Mária Bieliková Gerhard Friedrich
Georg Gottlob Stefan Katzenbeisser
György Turán (Eds.)

LNCS 7147

SOFSEM 2012: Theory and Practice of Computer Science

38th Conference on Current Trends
in Theory and Practice of Computer Science
Špindlerův Mlýn, Czech Republic, January 2012
Proceedings

Springer

Lecture Notes in Computer Science 7147

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

Mária Bieliková Gerhard Friedrich
Georg Gottlob Stefan Katzenbeisser
György Turán (Eds.)

SOFSEM 2012: Theory and Practice of Computer Science

38th Conference on Current Trends
in Theory and Practice of Computer Science
Špindlerův Mlýn, Czech Republic, January 21-27, 2012
Proceedings

Springer

Volume Editors

Mária Bieliková
Slovak University of Technology in Bratislava, Slovakia
E-mail: bielik@fiit.stuba.sk

Gerhard Friedrich
Alpen-Adria-Universität Klagenfurt, Austria
E-mail: gerhard.friedrich@uni-klu.ac.at

Georg Gottlob
University of Oxford, UK
E-mail: georg.gottlob@cs.ox.ac.uk

Stefan Katzenbeisser
Technische Universität Darmstadt, Germany
E-mail: skatzenbeisser@acm.org

György Turán
University of Illinois at Chicago, IL, USA
and University of Szeged, Hungary
E-mail: gyt@uic.edu

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-27659-0 e-ISBN 978-3-642-27660-6
DOI 10.1007/978-3-642-27660-6
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011944431

CR Subject Classification (1998): F.2, F.1, D.2, H.2-3, C.2, H.4, D.4, G.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume contains the *invited* and *contributed* papers selected for presentation at SOFSEM 2012, the 38th Conference on Current Trends in Theory and Practice of Computer Science, held January 21–27, 2012 in OREA Hotel Horal, Špindlerův Mlýn, in the Krkonoše Mountains of the Czech Republic.

SOFSEM (originally: SOFtware SEMinar) is devoted to leading research, and fosters the cooperation among researchers and professionals from academia and industry, in all areas of computer science. As a well-established and fully international conference, SOFSEM maintains the best of its original Winter School aspects, like a high number of invited talks (this year again 11) and an in-depth coverage of novel research results in selected areas within computer science. SOFSEM 2012 was organized around the following four tracks:

- *Foundations of Computer Science* (Chair: György Turán - University of Illinois at Chicago, USA, and University of Szeged, Hungary)
- *Software and Web Engineering* (Chair: Mária Bieliková - Slovak University of Technology in Bratislava, Slovakia)
- *Cryptography, Security, and Verification* (Chair: Stefan Katzenbeisser - Technical University of Darmstadt, Germany)
- *Artificial Intelligence* (Chair: Gerhard Friedrich - Alpen-Adria-University Klagenfurt, Austria)

In memory of Alan Turing, whose 100^{th} anniversary is celebrated in 2012, SOFSEM 2012 hosted a session on Turing machines. The session consisted of invited and contributed talks on Turing machines as the basic model of computability and complexity. SOFSEM 2012 was among the official Centenary Events of The Alan Turing Year.

With its four tracks, SOFSEM 2012 covered the latest advances in research, both theoretical and applied, in leading areas of computer science. The SOFSEM 2012 Program Committee consisted of 105 international experts from 25 different countries, representing the 4 track areas with outstanding expertise. An integral part of SOFSEM 2012 was the traditional *SOFSEM Student Research Forum* (Chair: Roman Špánek - Institute of Computer Science of the Academy of Sciences of the Czech Republic in Prague - ICS ASCR), organized with the aim to present student projects in the theory and practice of computer science and to give students feedback on both the originality of their scientific results and on their work in progress. The papers presented at the Student Research Forum were published in a local proceedings volume.

In response to the calls for papers, SOFSEM 2012 received 140 submissions by 297 authors, coming from 38 different countries of 5 continents (Americas, Asia, Australia and Europe): full papers were provided for 121 (87 %) of them. From these, there were 66 submissions in the Foundations of Computer Science Track (55 %) and 55 submissions (45 %) in the remaining three tracks: 26 in

the Software and Web Engineering Track (21 %), 18 in the Artificial Intelligence Track (15 %) and finaly 11 in the Cryptography, Security, and Verification Track (9 %).

After a detailed review process (using the EasyChair Conference System) with at least three reviewers per paper, a careful electronic selection procedure was carried out within each track between August 29 and September 20, 2011. In total 43 papers were selected for presentation at SOFSEM 2012, following strict criteria of quality and originality. From these, there were 24 submissions in the Foundations of Computer Science Track (56 %) and 19 (44 %) in the other 3 tracks: 9 in the Software and Web Engineering Track (21 %), 6 in the Artificial Intelligence Track (14 %) and finally 4 in the Cryptography, Security, and Verification Track (9 %).

Of the 43 accepted papers, 12 papers were submitted as student papers. In fact, two student papers, namely, the paper by Peter Damaschke and Azam Sheikh Muhammad and the one by Jiří Isa, Zuzana Reitermanová and Ondřej Sýkora, received the highest evaluations of all submissions in their respective tracks, the first one even the highest of all accepted papers.

Furthermore, 12 student papers were selected for the SOFSEM 2012 Student Research Forum, based on the recommendations of the Chair of the SRF, and with the approval of the Track Chairs.

As editors of these proceedings, we are grateful to everyone who contributed to the scientific program of the conference, especially the invited speakers and all authors of contributed papers. We thank all authors for their prompt responses to our editorial requests.

SOFSEM 2012 was the result of a considerable effort by many people. We would like to express our special thanks to:

- The SOFSEM 2012 Program Committees of the four tracks and all additional referees for their precise and detailed reviewing of the submissions
- Roman Špánek, of the ICS ASCR, for his preparation and handling of the Student Research Forum
- The SOFSEM Steering Committee headed by Július Štuller (ICS ASCR), for guidance and support throughout the preparation of the conference
- Springer's LNCS series, for its great support of the SOFSEM conferences

We are also greatly indebted to:

- The SOFSEM 2012 Organizing Committee consisting of Július Štuller (Chair), Pavel Tyl, Martin Řimnáč, and Dana Kuželová (all from the ICS ASCR) for the support and preparation of all aspects of the conference
- The Action M Agency, in particular Milena Zeithamlová and Pavlína Březinová, for the local arrangements of SOFSEM 2012

We especially thank Július Štuller, for his assistance in all duties and responsibilities of the Track Chairs and the PC/General Chair.

Finally, we are very grateful for the financial support of our sponsors, which enabled us to compose a high-quality program of invited speakers and helped us to keep the student fees low. We thank the Institute of Computer Science of the Academy of Sciences of the Czech Republic in Prague, for its invaluable support of all aspects of SOFSEM 2012.

October 2011

Mária Bieliková
Gerhard Friedrich
Georg Gottlob
Stefan Katzenbeisser
György Turán

Organization

Steering Committee

Mária Bieliková	Slovak University of Technology in Bratislava, Slovakia
Bernadette Charron-Bost	Ecole Polytechnique, France
Keith Jeffery	STFC Rutherford Appleton Laboratory, UK
Antonín Kučera	Masaryk University, Brno, Czech Republic
Jan van Leeuwen	Utrecht University, The Netherlands
Branislav Rovan	Comenius University, Bratislava, Slovakia
Július Štuller, Chair	Institute of Computer Science, Academy of Sciences, Czech Republic
Petr Tůma	Charles University in Prague, Czech Republic

Program Committee

PC Chair

Georg Gottlob	University of Oxford, UK

Track Chairs

Mária Bieliková	Slovak University of Technology in Bratislava, Slovakia
Gerhard Friedrich	Alpen Adria University Klagenfurt, Austria
Stefan Katzenbeisser	Technische Universität Darmstadt, Germany
György Turán	University of Illinois at Chicago, USA, and University of Szeged, Hungary

Student Research Forum Chair

Roman Špánek	Institute of Computer Science, Academy of Sciences, Czech Republic

PC Members

Isolde Adler	Goethe University Frankfurt am Main, Germany
Andris Ambainis	University of Latvia, Latvia
Ioannis Anagnostopoulos	University of Central Greece, Greece
Frederik Armknecht	University of Mannheim, Germany
Roman Barták	Charles University in Prague, Czech Republic
Petr Berka	University of Economics, Prague, Czech Republic
Miklos Biro	Corvinus University of Budapest, Hungary

Roderick Bloem	Graz University of Technology, Austria
Přemek Brada	University of West Bohemia, Czech Republic
Ivan Bratko	University of Ljubljana, Slovenia
Gerd Brewka	Leipzig University, Germany
Levente Buttyan	Budapest University of Technology and Economics, Hungary
Sven Casteleyn	Polytechnic University of Valencia, Spain
Sourav Chakraborty	Chennai Mathematical Institute, India
Ferdinando Cicalese	University of Salerno, Italy
Ondřej Čepek	Charles University in Prague, Czech Republic
Ivana Černá	Masaryk University, Brno, Czech Republic
Florian Daniel	University of Trento, Italy
Bhaskar Dasgupta	University of Illinois at Chicago, USA
Peter Dolog	Aalborg University, Denmark
Agostino Dovier	University of Udine, Italy
Johann Eder	University of Klagenfurt, Austria
Uwe Egly	Vienna University of Technology, Austria
Michael Elkin	Ben-Gurion University of the Negev, Israel
Wolfgang Faber	University of Calabria, Italy
Andreas Falkner	Siemens AG Österreich, Austria
Johann Gamper	Free University of Bolzano, Italy
Martin Gebser	University of Potsdam, Germany
Martin Grohe	Humboldt University Berlin, Germany
Vince Grolmusz	Eötvös University, Budapest, Hungary
Hele-Mai Haav	Institute of Cybernetics at TUT, Estonia
Kristoffer Hansen	Aarhus University, Denmark
Lisa Hellerstein	Polytechnic Institute of NY, USA
Eelco Herder	L3S Research Center, Germany
Pavel Herout	University of West Bohemia, Czech Republic
Jan Hidders	Delft University of Technology, The Netherlands
Martin Homola	Comenius University, Bratislava, Slovakia
Juraj Hromkovič	ETH Zurich, Switzerland
Dietmar Jannach	Dortmund University of Technology, Germany
Marina Jirotka	University of Oxford, UK
Dimitris Karagiannis	University of Vienna, Austria
Przemyslaw Kazienko	Wroclaw University of Technology, Poland
Johannes Kinder	EPFL, Switzerland
Petr Kosina	Brno University of Technology, Czech Republic
Daniel Král	Czech Republic
Rastislav Královič	Comenius University, Bratislava, Slovakia
Milos Kravcik	RWTH Aachen University, Germany
Petr Kroha	Chemnitz University of Technology, Germany
Klaus Kursawe	University of Nijmegen, The Netherlands
Miroslaw Kutylowski	Wroclaw University of Technology, Poland

Michal Laclavík	Institute of Informatics, Slovak Academy of Sciences, Slovakia
Viliam Lisý	Czech Technical University in Prague, Czech Republic
Martin Lopez-Nores	University of Vigo, Spain
Shachar Lovett	The Weizmann Institute of Science, Israel
Leszek Maciaszek	Wrocław University of Economics, Poland
Frederic Magniez	LIAFA, University of Paris 7, CNRS, France
Václav Matyáš	Masaryk University, Brno, Czech Republic
Marius Minea	Politechnic University of Timisoara, Romania
Victor Mitrana	Rovira i Virgili University, Tarragona, Romania
Tadeusz Morzy	Poznan University of Technology, Poland
Jerzy Nawrocki	Poznan University of Technology, Poland
Pavol Návrat	Slovak University of Technology in Bratislava, Slovakia
Ronald Ortner	University of Leoben, Austria
Jan Paralič	Technical University of Kosice, Slovakia
Michal Pěchouček	Czech Technical University in Prague, Czech Republic
Reinhard Pichler	Vienna University of Technology, Austria
Jaroslav Pokorný	Charles University in Prague, Czech Republic
Axel Polleres	Siemens AG Österreich, Austria
Bart Preneel	Catholic University, Leuven, Belgium
Daniel Reidenbach	Loughborough University, UK
Karel Richta	Charles University in Prague, Czech Republic
Branislav Rovan	Comenius University, Bratislava, Slovakia
Ahmad-Reza Sadeghi	Technische Universität Darmstadt, Germany
Davide Sangiorgi	University of Bologna, Italy
Ulrich Schöpp	LMU Munich, Germany
Radu Sion	Stony Brook University, USA
Boris Skoric	Eindhoven Technical University, The Netherlands
Markus Stumptner	University of South Australia, Australia
Vojtěch Svátek	University of Economics, Prague, Czech Republic
Balázs Szörényi	University of Szeged, Hungary
Petr Šaloun	Technical University Ostrava, Czech Republic
Daniel Štefankovič	University of Rochester, USA
Olga Štěpánková	Czech Technical University in Prague, Czech Republic
Július Štuller	Institute of Computer Science, Academy of Science, Czech Republic
Massimo Tisi	Polytechnical University of Milan, Italy
Sophie Tison	University of Lille 1, LIFL, France

Hans Tompits	Vienna University of Technology, Austria
Helmut Veith	Vienna University of Technology, Austria
Elad Verbin	University of Aarhus, Denmark
Heribert Vollmer	Hannover University, Germany
Valentino Vranić	Slovak University of Technology in Bratislava, Slovakia
Vincent Wade	Trinity College Dublin, Ireland
Manuel Wimmer	Vienna University of Technology, Austria
Franz Wotawa	Graz Technical University, Austria
Marek Zaionc	Jagiellonian University, Poland
Jaroslav Zendulka	Brno University of Technology, Czech Republic
Wieslaw Zielonka	LIAFA, University of Paris 7, France
Stanislav Živný	University of Oxford, UK

Organization

38th SOFSEM 2012 was organized by:

Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague
Action M Agency, Prague

Organizing Committee

Július Štuller, *Chair*	Institute of Computer Science, Prague, Czech Republic
Pavel Tyl	Institute of Computer Science, Prague, Czech Republic
Martin Řimnáč	Institute of Computer Science, Prague, Czech Republic
Dana Kuželová	Institute of Computer Science, Prague, Czech Republic
Milena Zeithamlová	Action M Agency, Prague, Czech Republic
Pavlína Březinová	Action M Agency, Prague, Czech Republic

Supported by

ČSKI – Czech Society for Cybernetics and Informatics

SSCS – Slovak Society for Computer Science

Table of Contents

Regular Papers

Foundations of Computer Science

Software and Web Engineering

Cryptography, Security, and Verification

Artificial Intelligence

The Legacy of Turing in Numerical Analysis

Felipe Cucker*

Department of Mathematics, City University of Hong Kong, 83 Tat Chee Avenue, Hong Kong, P.R. of China
macucker@cityu.edu.hk

Abstract. Alan Mathison Turing is revered among computer scientists for laying down the foundations of theoretical computer science via the introduction of the Turing machine, an abstract model of computation upon which, an elegant notion of cost and a theory of complexity can be developed. In this paper we argue that the contribution of Turing to "the other side of computer science", namely the domain of numerical computations as pioneered by Newton, Gauss, &c, and carried out today in the name of numerical analysis, is of an equally foundational nature.

1 Introduction

Alan Mathison Turing is mostly known, among scientists, for his 1936 paper where what we call today the *Turing machine* was first introduced [19].[1] This abstract model of computation became central in the theoretical foundations of the young computer science. A reason of this success is that, unlike other abstract models of computation leading to the same class of computable functions, the Turing machine provides a natural framework to define, and subsequently study, issues of complexity.

The development of computer languages, compilers, editors, web browsers, &c, brought to light a myriad of combinatorial problems demanding efficient algorithmic solutions and the complexity theory built upon the Turing machine has been pivotal in the phenomenal progress this development has seen.

Yet, maybe surprisingly, the intellectual context within which Turing wrote his 1936 paper had no connection whatsoever with digital computers. It hinged, instead, on issues of mathematical logic and foundations of mathematics. More surprisingly, however, is the fact that by the end of WWII, with digital computers already established in a number of research laboratories, Turing's attention did not focus on the Turing machine and the theory ultimately built upon it but on theoretical issues of numerical analysis. Turing worked at that time in the National Physical Laboratory and an account of his life and scientific endeavors between 1946 and 1948 is (appropriately) given by James Wilkinson in his 1970's Turing Lecture [23]. Our point above is made clear by Wilkinson:

* Partially supported by GRF grant CityU 100810.

[1] I say 'among scientists' because for most laymen it is his role in code breaking that makes his fame.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 1–13, 2012.

> Turing's international reputation rests mainly on his work on computable numbers but I like to recall that he was a considerable numerical analyst, and a good part of his time from 1946 onwards was spent working in this field [...]

The goal of this paper is to briefly survey some of the influence of Turing's work in todays' numerical analysis which, we believe, is as fundamental as it is in theoretical computer science. A part of this influence is due to one of his works [20] completed while at the NPL. This, once Turing's post-war interests are made clear, is to be expected. But we will also point out that his theory of computability, through its resounding success in theoretical computer science, eventually influenced numerical analysis as well.

2 Complexity and Accuracy

The following passage by (none less than) Gauss will serve us to start our discussion:

> Since none of the numbers we take out from logarithmic or trigonometric tables admit of absolute precision, but are all to a certain extent approximate only, the results of all calculations performed by the aid of these numbers can only be approximately true. [...] It may happen, that in special cases the effect of the errors of the tables is so augmented that we may be oblidged to reject a method, otherwise the best, and substitute another in its place.
>
> Carl Friedrich Gauss, *Theoria Motus* (cited in [11] p. 258).

There are a number of points made in these sentences. The first and most obvious is that real number computations are polluted by errors. These errors may arise by the use of tables (as in Gauss' quote), or by measurement limitations, or by, nowadays, the use of floating-point numbers in digital computers. Leaving the causes aside, the point made by Gauss is that errors accumulate during the computation and the final error may be of such a magnitude to make irrelevant the result of the computation.

The second point made by Gauss is more subtle. He remarks that in some cases we may be forced to reject an algorithm where errors accumulate badly, but which is *otherwise the best*, and adopt another algorithm with a better behavior regarding errors. He did not spell out the meaning of being "the best" in this context but I think we can safely infer that Gauss had complexity considerations in mind: the best would mean that which requires the smallest amount of work.

The third point is only implicit. It follows from Gauss' words that the process of finding an algorithmic solution to a computational problem passes through a two-step exercise. Firstly, one designs an algorithm which solves the problem. No error (or finite precision) considerations are present at this time. For instance, we think on solving linear systems of equations by Gaussian elimination, or by using Cramer's rule, or by substitution, ... In all cases we think within a mental framework where real numbers are available as they are the usual arithmetic operations on them. It is within this framework that we can compare algorithms

(and, in the example above, come to the conclusion that Gaussian elimination is better, i.e. faster, than substitution). In a second stage, finite precision enters the scene and one analyzes the error-accumulation behavior of the different algorithms solving a problem. It is at this second stage that one may need to replace the fastest algorithm by a slower one because of the error-magnifying properties of the former.

I am not claiming here that every single numerical analyst follows this two-stage exercise (though some may do it). But I do believe that this is a correct picture of the way algorithms in numerical analysis find their way into common practice. But let us return to Turing's work at the NPL, and with it his legacy within the first stage in Gauss' scheme.

3 Accuracy

According to Wilkinson,

> some time after my arrival, a system of 18 equations arrived in Mathematics Division and after talking around it for some time we finally decided to abandon theorizing and to solve it. [...] we decided on Gaussian elimination with complete pivoting. Turing was not particularly enthusiastic, partly because he was not an experienced performer on a desk machine and partly because he was convinced that it would be a failure.

Wilkinson then goes explaining how, instead, the computation was a total success and that it was this success (against Turing's initial apathy) that

> set him thinking afresh on the problem of rounding errors in elimination processes. About a year later he produced his famous paper [...]

The famous paper is [20] and it is fit that we devote some time to understand its contents.

A finite-precision algorithm working with a *machine precision* $\varepsilon_{\mathsf{mach}}$, $0 < \varepsilon_{\mathsf{mach}} < 1$, replaces, during the computation, all numbers x by a number $\tilde{x}$ such that $\tilde{x} = x(1-\delta)$ with $|\delta| \leq \varepsilon_{\mathsf{mach}}$. In a digital computer the number $\tilde{x}$ is obtained by keeping in the representation of x a fixed number of bits (or digits) in the mantissa. If $a \in \mathbb{R}^n$ is approximated by $\tilde{a}$ we may define the (normwise) *relative error* of this approximation by taking

$$\mathsf{RelError}(a) = \frac{\|a - \tilde{a}\|}{\|a\|}.$$

Now assume we have a function $\varphi : \mathbb{R}^n \to \mathbb{R}^m$ and an algorithm $\mathscr{A}$ meant to compute it. That is, $\mathscr{A}$ actually computes a function $\varphi^{\mathscr{A}}$ which depends on $\varepsilon_{\mathsf{mach}}$ and which coincides with φ under infinite precision ($\varepsilon_{\mathsf{mach}} = 0$). The central question underlying Gauss' quote can be rephrased as follows:

$$\text{How big is } \mathsf{RelError}(\varphi(a))?$$

Implicit in Turing's paper is the realization that the answer to this question relies on two different factors: on the one hand, the nature of the algorithm $\mathscr{A}$, and on the other hand, a magnification factor depending solely on a and φ. A rigorous explanation of this statement can be given in terms of the so called backward error analysis, vigorously pioneered by Wilkinson in the 1960s. But, in this paper, we do not need to enter into these details. It will suffice for our purposes to remark that we can define the magnification factor mentioned above by taking the worst-case magnification in $\varphi(a)$ of small errors in a. More formally, we define

$$\mathsf{cond}^{\varphi}(a) = \lim_{\delta\to 0} \sup_{\mathsf{RelError}(a)\leq\delta} \frac{\mathsf{RelError}(\varphi(a))}{\mathsf{RelError}(a)}.$$

In case A is a square matrix and $\varphi(A) := A^{-1}$ one can prove that $\mathsf{cond}^{\varphi}(A) = \|A\|\|A^{-1}\|$. For linear equation solving, that is, for $\varphi(A, b) := A^{-1}b$, we do not have such an exact expression but one can prove that $\|A\|\|A^{-1}\| \leq \mathsf{cond}^{\varphi}(A, b) \leq 2\|A\|\|A^{-1}\|$. It follows that the quantity

$$\kappa(A) := \|A\|\|A^{-1}\|$$

measures, maybe up to a small factor, the worst case magnification in A^{-1} or in $x = A^{-1}b$ of small errors in the input A (resp. (A, b)). This quantity is known as the *condition number* of A and you may correctly guess that the origin of this name is in Turing's 1948 paper.

Indeed, Section 8 of the paper carries the title "Ill-conditioned matrices and equations" and starts with the following statement:

> When we come to make estimates of errors in matrix processes we shall find that the chief factor limiting the accuracy that can be obtained is 'ill-conditioning' of the matrices involved

and continues

> It is characteristic of ill-conditioned sets of equations that small percentage errors in the coefficients given may lead to large percentage errors in the solution.

He then defined $\kappa(A)$ "as a measure of the degree of ill-conditioning in a matrix" and proceeded to derive error estimates for some algorithmic methods[2].

Turing was not alone in his endeavour. Independently of him, John von Neumann and Herman Goldstine were pursuing similar thoughts in [21]. In a sequel [22] to this paper, von Neumann and Goldstine introduced a theme which,

[2] In our exposition above we were sloppy in specifying a norm on the space $\mathbb{R}^{n\times n}$ of matrices. The relations between $\mathsf{cond}(A)$ and $\kappa(A)$ we described above hold true for operator norms. In [20], Turing used instead the max-norm given by $\max|a_{ij}|$ and the Frobenius norm $\sqrt{\sum a_{ij}^2}$. These differences are, nonetheless, inessential to the argument.

subsequently championed by Steve Smale (see [16]), would become central in the foundations of numerical analysis: the probabilistic analysis of condition numbers. The motivation is clear. Error bounds (and, we will see it in the next section, complexity bounds as well) depend on the condition $\mathsf{cond}^{\varphi}(a)$ of the input a. But we do not know this quantity and it has been observed that its computation requires, essentially, to compute $\varphi(a)$ (see [15]). A way out from this vicious circle is to estimate the expectation of $\mathsf{cond}^{\varphi}(a)$ (or, more commonly, of its logarithm) for random a. Goldstine and von Neumann did not go that far. But Alan Edelman did, proving that for random $n \times n$ matrices (with independent standard Gaussian entries) one has

$$\mathbb{E}(\log \kappa(A)) = \log n + C + o(1), \qquad \text{as } n \to \infty,$$

with $C = 1.537$ for real matrices and $C = 0.982$ for complex matrices [9]. This result produces upper bounds on the expected loss of precision for matrix inversion and linear equation solving.

Even though desirable, it is not feasible to describe here the extensions, occurrences, and applications of condition numbers. We should nevertheless mention that a condition number (indeed, a close relative of κ above) played a fundamental role in the recent advances [1,2,7] towards a solution of the 17th of the problems posed by Steve Smale for the mathematicians of the 21st century [17]. A comprehensive exposition of the aspects of conditioning mentioned above can be found in [5].

4 Complexity

After the overview on Turing's contribution to the first stage in Gauss' scheme, we can start considering the same for the second stage. Here Turing's role is bigger and with more disparate origins. It is not controversial today to state that his 1936 paper [19] is at the center of contemporary complexity theory. Two features of the paper stand out. Firstly, the introduction of the Turing machine, an abstract model of computation allowing for a formal definition of the *cost* of a computation. Secondly, the use of *reductions* between computational problems as a means to state that one problem requires no less resources (i.e., cost) than another for its algorithmic solution. Turing introduced the latter with the goal of classifying undecidable problems but his ideas eventually translated into a classification of decidable problems (in terms of resource requirements). Thus, for instance, the class P is that of all sets decidable within a cost polynomial in the size of the input. Similarly, the class NP is that of the sets decidable within a non-deterministic polynomial cost (I skip the formal definition, I assume the reader is familiar with it). Problems that can be proved to be in $\mathsf{NP} \setminus \mathsf{P}$ are considered to be intrinsically difficult to solve and the best candidates in this class are the so called NP-*complete* problems, defined in terms of a variation on the reductions introduced by Turing. The first example of such a problem was independently exhibited by Steve Cook [8] and Leonid Levin [14]. Shortly after, Richard Karp proved NP-completeness for 23 problems in different areas of

computation showing that the phenomenon was much more general than Cook and Levin's results suggested [12] and hundreds of problems have joined the list since then. Paradoxically, however, no proof has been found that NP-complete problems are not in P (or, in other words, that $\mathsf{P} \neq \mathsf{NP}$). The evidence pointing to this fact is overwhelming but its proof is elusive. The 3rd of the problems proposed by Smale (mentioned above) is to determine whether $\mathsf{P} = \mathsf{NP}$.

These considerations need to be contextualized in what we could term as *discrete computations*. Basically, these are computations whose intervening objects can be precisely described with a finite word over a finite alphabet. Typical such objects are graphs, trees, rational numbers, and, obviously, finite words over a finite alphabet (as those forming this text). In opposition to these computations are those of *continuous mathematics*, that is, those involving real or complex numbers, commonly known as *numerical algorithms*. These can only be approximated by finite words over a finite alphabet (and hence Gauss' tribulations on the accumulation of errors).

It is not surprising that Turing's 1948 paper begins with a short section on cost. Indeed, its Section 1, bearing the title "Measure of work in a process", can be quoted in its entirety:

> It is convenient to have a measure of the amount of work involved in a computing process, even though it be a very crude one. We may count up the number of times that various elementary operations are applied in the whole process and then give them various weights. We might, for instance, count the number of additions, subtractions, multiplications, divisions, recordings of numbers, and extractions of figures from tables. In the case of computing with matrices most of the work consists of multiplications and writing down numbers, and we shall therefore only attempt to count the number of multiplications and recordings. For this purpose, a reciprocation will count as a multiplication. This is purely formal. A division will then count as two multiplications; this seems a little too much, and there may be other anomalies, but on the whole substantial justice should be done.

That is, Turing is disregarding a measure of cost in terms of elementary bit operations (as would correspond to a Turing machine) and is suggesting instead what is usually called as *algebraic cost*, in which floating-point numbers are considered as undivisible units and arithmetic operations between them as elementary machine operations.

This pondering was not meant to build a complexity theory as the one underlying the P vs NP problem. It barely aimed to provide a framework within which an idea of cost for numerical algorithms can be agreed on and algorithms can therefore being compared w.r.t. cost as implicit in Gauss' quotation. The definition of the class P did not arrive until the 60s (see [10]) and the structural approach to complexity until the early 70s (with the papers of Cook, Levin, and Karp), and all this for discrete computations.

For numerical computations, it is not until the late 80s that a paper by Lenore Blum, Michael Shub, and Steve Smale [4] would put together the two streams of thought initiated by Turing we mentioned above. Blum, Shub and Smale

define an abstract model of computation, which they call *real Turing machine* (but has since been refered to as *BSS-machine*), and associate to it a notion of cost which essentially coincides with that of Section 1 of [20]. Furthermore, they extended the notion of nondeterminism, defined the class $\mathsf{NP}_{\mathbb{R}}$ of sets decidable in nondeterministic polynomial time and exhibited an $\mathsf{NP}_{\mathbb{R}}$-complete problem (a task for which, Turing's idea of reduction was again of the essence). We will close this article by giving some details of this complexity theory over the reals.

We denote by $\mathbb{R}^n$ the n-fold cartesian product of $\mathbb{R}$ and by $\mathbb{R}^\infty$ the disjoint union $\bigsqcup_{n\geq 1}\mathbb{R}^n$. This is the space where inputs are taken from in numerical computations and, in fact, BSS-machines compute functions $\varphi : \mathbb{R}^\infty \to \mathbb{R}^\infty$. In case a machine M computes a function $\varphi^M : \mathbb{R}^\infty \to \{0,1\}$ we say that M *decides* the set $S := \{a \in \mathbb{R}^\infty \mid \varphi(a) = 1\}$. For $a \in \mathbb{R}^\infty$ we write $|a| = n$ when $a \in \mathbb{R}^n$ and we say that n is the *size* of a. The class $\mathsf{P}_{\mathbb{R}}$ is then defined as the class of sets S for which there is a machine M deciding S and such that, for every $a \in \mathbb{R}^\infty$, the cost $\mathsf{cost}^M(a)$ of the computation of M with input a is polynomial in $|a|$. That is,

$$\mathsf{P}_{\mathbb{R}} := \{S \subseteq \mathbb{R}^\infty \mid \exists M\, \forall a\, (\varphi^M(a) = 1 \iff a \in S)\ \&\ \mathsf{cost}^M(a) = |a|^{\mathcal{O}(1)}\}.$$

We can similarly define the class $\mathsf{NP}_{\mathbb{R}}$. A set S is in $\mathsf{NP}_{\mathbb{R}}$ if and only if there exists $B \subseteq \mathbb{R}^\infty \times \mathbb{R}^\infty$, $B \in \mathsf{P}_{\mathbb{R}}$, such that, for all $a \in \mathbb{R}^\infty$,

$$a \in S \iff \exists y\, (a, y) \in B. \tag{1}$$

The point $y \in \mathbb{R}^\infty$ is said the be a *witness* for the membership of a to S. Clearly, one can suppose that $|y| = |a|^{\mathcal{O}(1)}$ since otherwise M cannot even read y.

An example of set in $\mathsf{NP}_{\mathbb{R}}$ is the set 4FEAS of polynomials of degree 4 in several variables which possess a real zero. In this case, the input a corresponds to a polynomial system f in n variables and the witness y to a point $\xi \in \mathbb{R}^n$. The computation performed by M amounts to evaluating $f(\xi)$ and returning 1 if and only if this evaluation yields zero. The size of f (which in this case the number of coefficients of f) is $\Theta(n^4)$ and the evaluation can be done with cost polynomial in this size. More importantly, Blum, Shub, and Smale proved that 4FEAS is $\mathsf{NP}_{\mathbb{R}}$-complete pointing towards a fundamental difficulty in the problem of deciding whether a polynomial (or a polynomial system) has a real zero. This is one of the main results in [4].

Unlike the situation in the context of discrete computations, there was no avalanche of $\mathsf{NP}_{\mathbb{R}}$-complete problems as a sequel of the publication of [4]. An unarguably reason is a smaller pool of problems (reasonable candidates will have to be problems dealing with polynomial systems and sets defined by them). A deeper reason, which explains why, nonetheless, problems in this pool had not been classified as complete in *any* complexity class over the reals is the fact that the catalog of complexity classes over the reals was not broad enough. This was brought to the light in [6].

To describe the ideas in [6] it will be convenient to return to the definition of $\mathsf{NP}_{\mathbb{R}}$ above. An immediate extension of this definition is the following.

Definition 1. *Let $\mathcal{C}$ be a complexity class of decision problems. A set S is in $\exists\mathcal{C}$ if and only if there exists $B \subseteq \mathbb{R}^\infty \times \mathbb{R}^\infty$, $B \in \mathcal{C}$, such that, for all $a \in \mathbb{R}^\infty$,*

$$a \in S \iff \exists y\, (a, y) \in B.$$

A set S is in $\forall\mathcal{C}$ if and only if there exists $B \subseteq \mathbb{R}^\infty \times \mathbb{R}^\infty$, $B \in \mathcal{C}$, such that, for all $a \in \mathbb{R}^\infty$,

$$a \in S \iff \forall y\, (a, y) \in B.$$

The class $\mathsf{NP}_{\mathbb{R}}$ is therefore $\exists\mathsf{P}_{\mathbb{R}}$ and we will also denote it simply by $\exists$. Also, the class $\mathsf{coNP}_{\mathbb{R}}$ is defined to be $\forall\mathsf{P}_{\mathbb{R}}$ and we will denote it by $\forall$.

Definition 1 allows to alternate the use of the quantifiers $\exists$ and $\forall$ to obtain classes such as $\exists\forall\exists$, which is usually denoted by $\Sigma^3_{\mathbb{R}}$. The union of all the complexity classes obtained this way, starting from $\mathsf{P}_{\mathbb{R}}$, is the *polynomial hierarchy* over $\mathbb{R}$. Complexity classes in this hierarchy are natural counterparts of the classical (i.e., discrete) polynomial hierarchy.

The finding in [6] is that the classes in the polynomial hierarchy over $\mathbb{R}$ are not sufficient to accommodate many of the problems naturally arising when considering polynomial systems and the sets they define. We next describe some of these problems.

We first recall that an *algebraic circuit* over $\mathbb{R}$ is an acyclic directed graph where each node has indegree 0, 1 or 2. Nodes with indegree 0 are either labeled as *input nodes* or with elements of $\mathbb{R}$ (we shall call them *constant nodes*). Nodes with indegree 2 are labeled with the binary operators of $\mathbb{R}$, i.e. one of $\{+, \times, -, /\}$. They are called *arithmetic nodes*. Nodes with indegree 1 are either *sign nodes* or *output nodes*. All the output nodes have outdegree 0. Otherwise, there is no upper bound for the outdegree of the other kinds of nodes. Occasionally, the nodes of an algebraic circuit will be called *gates*.

To a circuit $\mathscr{C}$ with n input gates and m output gates one associates a function $f_{\mathscr{C}} : \mathbb{R}^n \to \mathbb{R}^m$ (see [6] for details). This function may not be total since divisions by zero may occur (in which case, by convention, $f_{\mathscr{C}}$ is not defined on its input).

We say that an algebraic circuit is a *decision* circuit if it has only one output gate whose parent is a sign gate (and hence $f_{\mathscr{C}} : \mathbb{R}^n \to \{0, 1\}$). The set *decided* by the circuit is

$$S_{\mathscr{C}} = \{a \in \mathbb{R}^n \mid f_{\mathscr{C}}(a) = 1\}.$$

Subsets of $\mathbb{R}^n$ decidable by algebraic circuits are known as *semialgebraic sets*. They are defined as those sets which can be written as a Boolean combination of solution sets of polynomial inequalities $\{a \in \mathbb{R}^n \mid f(a) \geq 0\}$.

Semialgebraic sets will be inputs to our problems. They will be given either by a Boolean combination of polynomial equalities and inequalities or by a decision circuit. If not otherwise specified, we mean the first variant. In this case, polynomials are encoded with the so called *dense encoding*, i.e., they are represented by the complete list of their coefficients (including zero coefficients, as in 4FEAS above).

Partial functions $f : \mathbb{R}^n \to \mathbb{R}^m$ computable by algebraic circuits are known as *piecewise rational*. These are the functions f for which there exists a semialgebraic partition $\mathbb{R}^n = S_0 \cup S_1 \cup \ldots \cup S_k$ and rational functions $g_i : S_i \to \mathbb{R}^m$, $i = 1, \ldots, k$ such that g_i is well-defined on S_i and $f_{|S_i} = g_i$. Note that f is undefined on S_0. We will also consider piecewise rational functions as inputs to some problems. They will be encoded by algebraic circuits.

We next define the following problems:

FEAS$_{\mathbb{R}}$ (*Polynomial feasibility*) Given a polynomial $f \in \mathbb{R}[X_1, \ldots, X_n]$, decide whether there exists $x \in \mathbb{R}^n$ such that $f(x) = 0$.

DIM$_{\mathbb{R}}(d)$ (*Semialgebraic dimension*) Given a semialgebraic set S and $d \in \mathbb{N}$, decide whether $\dim S \geq d$.

EADH$_{\mathbb{R}}$ (*Euclidean Adherence*) Given a semialgebraic set S and a point x, decide whether x belongs to the Euclidean closure $\overline{S}$ of S.

EDENSE$_{\mathbb{R}}$ (*Euclidean Denseness*) Given a decision circuit $\mathscr{C}$ with n input gates, decide whether $\overline{S_{\mathscr{C}}} = \mathbb{R}^n$.

ERD$_{\mathbb{R}}$ (*Euclidean Relative Denseness*) Given semialgebraic sets S and V, decide whether S is included in $\overline{V}$.

LERD$_{\mathbb{R}}$ (*Linearly restricted Euclidean Relative Denseness*) Given a semialgebraic set $V \subseteq \mathbb{R}^n$ and points $a_0, a_1, \ldots, a_k \in \mathbb{R}^n$, decide whether $a_0 + \langle a_1, \ldots, a_k \rangle$ is included in $\overline{V}$.

ZADH$_{\mathbb{R}}$ (*Zariski Adherence*) Given a semialgebraic set S and a point x, decide whether x belongs to the Zariski closure $\overline{S}^Z$ of S.

ZDENSE$_{\mathbb{R}}$ (*Zariski Denseness*) Given a decision circuit $\mathscr{C}$ with n input gates, decide whether $\overline{S_{\mathscr{C}}}^Z = \mathbb{R}^n$.

UNBOUNDED$_{\mathbb{R}}$ (*Unboundedness*) Given a semialgebraic set S, is it unbounded?

LOCDIM$_{\mathbb{R}}$ (*Local Dimension*) Given a semialgebraic set $S \subseteq \mathbb{R}^n$, a point $x \in S$, and $d \in \mathbb{N}$, is $\dim_x S \geq d$?

ISOLATED$_{\mathbb{R}}$ (*Isolated*) Given a semialgebraic set $S \subseteq \mathbb{R}^n$ and a point $x \in \mathbb{R}^n$, decide whether x is an isolated point of S.

EXISTISO$_{\mathbb{R}}$ (*Existence of isolated points*) Given a semialgebraic set $S \subseteq \mathbb{R}^n$, decide whether there exist a point x isolated in S.

BASICCLOSED$_{\mathbb{R}}$ (*Closedness for basic semialgebraic sets*) Given a basic semialgebraic set S, is it closed?

BASICCOMPACT$_{\mathbb{R}}$ (*Compactness for basic semialgebraic sets*) Given a basic semialgebraic set S, is it compact?

SOCS$_{\mathbb{R}}(k)$ (*Smallest Order Coefficient Sign, k variables*) Given a division-free straight-line program Γ in k input variables $X_1, \ldots, X_k$, decide whether the smallest-order coefficient (w.r.t. the ordering $X_1 \succ X_2 \succ \ldots \succ X_k$) of f_Γ (the polynomial in X computed by Γ) is positive.

LOCSUPP$_{\mathbb{R}}$ (*Local Support*) Given a circuit $\mathscr{C}$ with n input nodes and a linear equation $\ell(x) = 0$, decide whether there exists $x_0 \in \mathbb{R}^n$ and $\delta > 0$ such that $S_{\mathscr{C}} \cap \{\ell < 0\} \cap B(x_0, \delta) = \emptyset$ and $\dim(\overline{S_{\mathscr{C}}} \cap \{\ell = 0\} \cap B(x_0, \delta)) = n - 1$.

TOTAL$_{\mathbb{R}}$ (*Totalness*) Given a circuit $\mathscr{C}$, decide whether $f_{\mathscr{C}}$ is total.

INJ$_{\mathbb{R}}$ (*Injectiveness*) Given a circuit $\mathscr{C}$, decide whether $f_{\mathscr{C}}$ is injective.

SURJ$_{\mathbb{IR}}$ (*Surjectiveness*) Given a circuit $\mathscr{C}$, decide whether $f_{\mathscr{C}}$ is surjective.

IMAGEZDENSE$_{\mathbb{IR}}$ (*Image Zariski Dense*) Given a circuit $\mathscr{C}$, decide whether the image of $f_{\mathscr{C}}$ is Zariski dense.

IMAGEEDENSE$_{\mathbb{IR}}$ (*Image Euclidean Dense*) Given a circuit $\mathscr{C}$, decide whether the image of $f_{\mathscr{C}}$ is Euclidean dense.

DOMAINZDENSE$_{\mathbb{IR}}$ (*Domain Zariski Dense*) Given a circuit $\mathscr{C}$, decide whether the domain of $f_{\mathscr{C}}$ is Zariski dense.

DOMAINEDENSE$_{\mathbb{IR}}$ (*Domain Euclidean Dense*) Given a circuit $\mathscr{C}$, decide whether the domain of $f_{\mathscr{C}}$ is Euclidean dense.

CONT$_{\mathbb{IR}}$ (*Continuity*) Given a circuit $\mathscr{C}$, decide whether $f_{\mathscr{C}}$ is continuous.

CONT$_{\mathbb{IR}}^{\mathrm{DF}}$ (*Continuity for Division-Free Circuits*) Given a division-free circuit $\mathscr{C}$, decide whether $f_{\mathscr{C}}$ is continuous.

CONTPOINT$_{\mathbb{IR}}^{\mathrm{DF}}$ (*Continuity at a Point for Division-Free Circuits*) Given a division-free circuit $\mathscr{C}$ with n input gates and $x \in \mathbb{R}^n$, decide whether $f_{\mathscr{C}}$ is continuous at x.

LIPSCHITZ$_{\mathbb{IR}}(k)$ (*Lipschitz-k*) Given a circuit $\mathscr{C}$, and $k > 0$, decide whether $f_{\mathscr{C}}$ is Lipschitz-k, i.e., whether for all $x, y \in \mathbb{R}^n$, $\|f(x) - f(y)\| \leq k\|x - y\|$.

LIPSCHITZ$_{\mathbb{IR}}$ (*Lipschitz*) Given a circuit $\mathscr{C}$, decide whether $f_{\mathscr{C}}$ is Lipschitz, i.e., whether there exists $k > 0$ such that $f_{\mathscr{C}}$ is Lipschitz-k.

To classify the complexity of these problems some new complexity classes, with no counterparts in the discrete context, are necessary.

Definition 2. *Let $\mathcal{C}$ be a complexity class of decision problems. A set A belongs to $\mathsf{H}\mathcal{C}$ if there exists $B \subseteq \mathbb{R} \times \mathbb{R}^\infty$, $B \in \mathcal{C}$, such that, for all $a \in \mathbb{R}^\infty$,*

$$a \in A \iff \exists \mu > 0 \, \forall \varepsilon \in (0, \mu) \, (\varepsilon, a) \in B.$$

A set A belongs to $\forall^*\mathcal{C}$ if there exist a polynomial p and a set $B \subseteq \mathbb{R}^\infty \times \mathbb{R}^\infty$, $B \in \mathcal{C}$, such that, for all $a \in \mathbb{R}^\infty$,

$$a \in A \iff \dim\{z \in \mathbb{R}^{p(|a|)} \mid (z, a) \notin B\} < p(|a|).$$

If $\mathcal{C}$ is a complexity class we denote by $\mathcal{C}^{\mathsf{c}}$ the class of its complements, i.e., the class of all sets A such that $A^{\mathsf{c}} \in \mathcal{C}$. We define $\exists^*\mathcal{C} = (\forall^*\mathcal{C}^{\mathsf{c}})^{\mathsf{c}}$.

We note that A belongs to $\exists^*\mathcal{C}$ if and only if there exist a polynomial p and a set $B \subseteq \mathbb{R}^\infty \times \mathbb{R}^\infty$, $B \in \mathcal{C}$, such that, for all $a \in \mathbb{R}^\infty$,

$$a \in A \iff \dim\{z \in \mathbb{R}^{p(|a|)} \mid (z, a) \in B\} = p(|a|).$$

The quantifier H was introduced in [6]. It captures the notion of "for all sufficiently small numbers." The quantifiers $\forall^*$ and $\exists^*$ capture the notions of "for almost all points" and "for sufficiently many points" in a specific sense. They were first introduced by Koiran in [13].

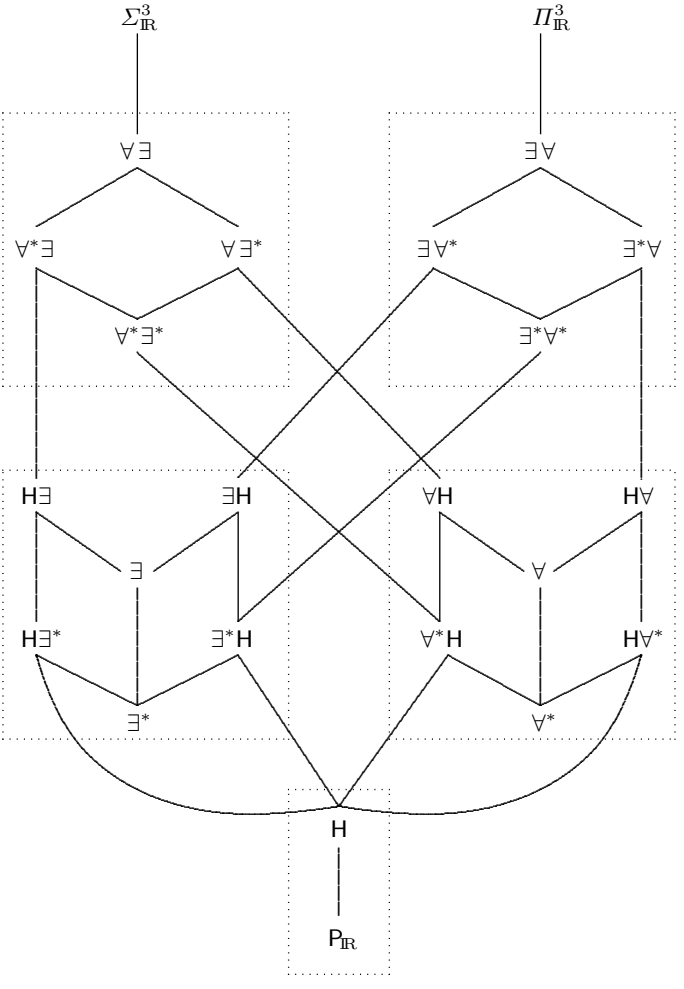
Using these operators we may define many new complexity classes. Notations such as $\exists^*\forall$, $\mathsf{H}\forall$, or $\exists^*\mathsf{H}$ denote some of the newly created complexity classes in an obvious manner. To avoid a cumbersome notation, we also write H instead of $\mathsf{HP}_{\mathbb{IR}}$.

It is beyond the goal of this article to describe in detail all of the structural properties of the classes defined above. We just mention that H is closed by complements and that, for any class $\mathcal{C}$ as above, we have the inclusion

$$\exists^*\mathcal{C} \subseteq \exists\mathcal{C}.$$

The following diagram gives a landscape of complexity classes in the lower levels of the polynomial hierarchy over $\mathbb{R}$. All upward lines mean inclusion. Note that not all possible classes below $\Sigma^3_{\mathbb{R}}$ or $\Pi^3_{\mathbb{R}}$ are in the diagram. We restricted attention to those standing out (e.g., because of having natural complete problems).

Boxes enclosing groups of complexity classes do not have a very formal meaning. They are rather meant to convey the informal idea that some classes are "close enough" to be clustered together.



We finally summarize the complexity sorting of the list of problems given above in the following table. The meaning of each row should be clear.

Problem	Complete in	Lower bound	Upper bound
$\text{FEAS}_{\mathbb{R}}$	$\exists$		
$\text{DIM}_{\mathbb{R}}(d)$	$\exists$		
$\text{SOCS}_{\mathbb{R}}(k)$	H^k		
$\text{ZDENSE}_{\mathbb{R}}$	$\exists^*$		
$\text{DOMAINZDENSE}_{\mathbb{R}}$	$\exists^*$		
$\text{EDENSE}_{\mathbb{R}}$	$\forall^*$		
$\text{DOMAINEDENSE}_{\mathbb{R}}$	$\forall^*$		
$\text{IMAGEZDENSE}_{\mathbb{R}}$	$\exists$		
$\text{TOTAL}_{\mathbb{R}}$	$\forall$		
$\text{INJ}_{\mathbb{R}}$	$\forall$		
$\text{LIPSCHITZ}_{\mathbb{R}}(k)$	$\forall$		
$\text{ZADH}_{\mathbb{R}}$		$\exists$	?
$\text{EADH}_{\mathbb{R}}$	$\mathsf{H}\exists$		
$\text{UNBOUNDED}_{\mathbb{R}}$	$\mathsf{H}\exists$		
$\text{LOCDIM}_{\mathbb{R}}$	$\mathsf{H}\exists$		
$\text{ISOLATED}_{\mathbb{R}}$	$\mathsf{H}\forall$		
$\text{CONT}_{\mathbb{R}}$		$\forall$	$\mathsf{H}^3\forall$
$\text{CONT}_{\mathbb{R}}^{\text{DF}}$		$\forall$	$\mathsf{H}^2\forall$
$\text{CONTPOINT}_{\mathbb{R}}^{\text{DF}}$	$\mathsf{H}\forall$		
$\text{LIPSCHITZ}_{\mathbb{R}}$		$\forall$	$\mathsf{H}\forall$
$\text{LOCSUPP}_{\mathbb{R}}$	$\exists^*\mathsf{H}$		
$\text{EXISTISO}_{\mathbb{R}}$		$\mathsf{H}\forall$	$\exists\forall$
$\text{BASICCLOSED}_{\mathbb{R}}$	$\mathsf{H}\forall$		
$\text{BASICCOMPACT}_{\mathbb{R}}$	$\mathsf{H}\forall$		
$\text{LERD}_{\mathbb{R}}$	$\forall^*\exists$		
$\text{IMAGEEDENSE}_{\mathbb{R}}$	$\forall^*\exists$		
$\text{ERD}_{\mathbb{R}}$		$\forall^*\exists$	$\forall\exists$
$\text{SURJ}_{\mathbb{R}}$	$\forall\exists$		

An exposition of what was the state-of-the-art in complexity theory over the reals at the end of the 90s can be found in [3].

References

1. Beltrán, C., Pardo, L.M.: Smale's 17th problem: average polynomial time to compute affine and projective solutions. J. Amer. Math. Soc. 22(2), 363–385 (2009)
2. Beltrán, C., Pardo, L.M.: Fast linear homotopy to find approximate zeros of polynomial systems. Found. Comput. Math. 11(1), 95–129 (2011)
3. Blum, L., Cucker, F., Shub, M., Smale, S.: Complexity and Real Computation. Springer, Heidelberg (1998)
4. Blum, L., Shub, M., Smale, S.: On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. Bulletin of the Amer. Math. Soc. 21, 1–46 (1989)
5. Bürgisser, P., Cucker, F.: Condition. Forthcoming book

6. Bürgisser, P., Cucker, F.: Exotic quantifiers, complexity classes, and complete problems. Found. Comput. Math. 9, 135–170 (2009)
7. Bürgisser, P., Cucker, F.: On a problem posed by Steve Smale. In: To Appear at Annals of Mathematics (2011)
8. Cook, S.: The complexity of theorem proving procedures. In: 3rd Annual ACM Symp. on the Theory of Computing, pp. 151–158 (1971)
9. Edelman, A.: Eigenvalues and condition numbers of random matrices. SIAM J. of Matrix Anal. and Applic. 9, 543–556 (1988)
10. Edmonds, J.: Paths, trees, and flowers. Canadian J. of Mathematics 17, 449–467 (1965)
11. Goldstine, H.H.: A History of Numerical Analysis from the 16th through the 19th Century. Springer, Heidelberg (1977)
12. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R., Thatcher, J. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press (1972)
13. Koiran, P.: The real dimension problem is $\mathrm{NP}_{\mathrm{IR}}$-complete. J. of Complexity 15, 227–238 (1999)
14. Levin, L.: Universal sequential search problems. Probl. Pered. Inform., IX 3, 265–266 (1973) (in Russian); (English translation in Problems of Information Trans. 9,3; corrected translation in [18])
15. Renegar, J.: Is it possible to know a problem instance is ill-posed? J. of Complexity 10, 1–56 (1994)
16. Smale, S.: Complexity theory and numerical analysis. In: Iserles, A. (ed.) Acta Numerica, pp. 523–551. Cambridge University Press (1997)
17. Smale, S.: Mathematical problems for the next century. In: Arnold, V., Atiyah, M., Lax, P., Mazur, B. (eds.) Mathematics: Frontiers and Perspectives, pp. 271–294. AMS (2000)
18. Trakhtenbrot, B.A.: A survey of russian approaches to perebor (brute-force search) algorithms. Annals of the History of Computing 6, 384–400 (1984)
19. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proc. London Math. Soc., Ser. 2 42, 230–265 (1936)
20. Turing, A.M.: Rounding-off errors in matrix processes. Quart. J. Mech. Appl. Math. 1, 287–308 (1948)
21. von Neumann, J., Goldstine, H.H.: Numerical inverting matrices of high order. Bulletin of the Amer. Math. Soc. 53, 1021–1099 (1947)
22. von Neumann, J., Goldstine, H.H.: Numerical inverting matrices of high order, II. Proceedings of the Amer. Math. Soc. 2, 188–202 (1951)
23. Wilkinson, J.: Some comments from a numerical analyst. Journal ACM 18, 137–147 (1971)

Turing Machines for Dummies
Why Representations Do Matter

Peter van Emde Boas

ILLC, FNWI, Universiteit van Amsterdam, P.O. Box 94242, 1090 GE Amsterdam
Bronstee.com Software & Services B.V., Heemstede
Dept. Comp. Sci. University of Petroleum, Chang Ping, Beijing, P.R. China
peter@bronstee.com

Abstract. Various methods exists in the litearture for denoting the configuration of a Turing Machine. A key difference is whether the head position is indicated by some integer (mathematical representation) or is specified by writing the machine state next to the scanned tape symbol (intrinsic representation).

From a mathematical perspective this will make no difference. However, since Turing Machines are primarily used for proving undecidability and/or hardness results these representations do matter. Based on a number of applications we show that the intrinsic representation should be preferred.

1 The Turing Machine Model

Given the nature of the meeting I expect that the dummies mentioned in my title will not be present in the audience. Still I believe that it is useful to start with a description of the Turing Machine model as we are supposed to know it.

The simplest version of the Turing machine is defined in mathematical terms by a tuple $M = \langle K, \Sigma, P, q_0, q_f, b, \Delta \rangle$. The machine has only a single one-dimensional tape, with tape alphabet Σ, and a set of internal states K. The program P of the finite control consists of a set of quintuples $\langle q, s, q', s', m \rangle \in K \times \Sigma \times K \times \Sigma \times \Delta$. Here the set $\Delta = \{L, 0, R\}$ denotes the set of possible head moves : Left, stay put or Right. The meaning of this quintuple is: if in state q the head is scanning symbol s then print symbol s', perform move m and proceed to state q'. The states q_0 and q_f are two special elements in K denoting the initial and the final state respectively. The symbol b is a special tape symbol called blank which represents the contents of a tape-cell which never has been scanned by the head.

In this single tape model there is no special input or output tape. The input is written on the unique tape in the initial configuration with the unique head scanning the leftmost input symbol. When started the computation will perform applicable instructions on the configuration up to the point in time where some termination condition is satisfied (if such a configuration arises at all). Various termination conditions are used in the literature. Absence of an applicable instruction is a possible termination condition, but one can also use specially

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 14–30, 2012.

designed halting states to which one can ascribe a quality of being accepting or rejection. Another possibility is to modify the instruction format allowing the state from the machine to disappear, leaving a configuration consisting of tape symbols only.

If one wants the single tape model to produce output one obtains such output by an ad-hoc convention from the final configuration (for example, the output consists of all non-blank tape symbols written to the left of the head in the final configuration).

Note that if we denote configurations of the single tape machine in the format $\$\Sigma^* K \Sigma^*\$$, with the state symbol written in front of the currently scanned tape symbol, the transitions between two successive configurations are described by a very simple context sensitive grammar. In this grammar one includes for example for the instruction $\langle q, s, q', s', R\rangle$ the production rules $(qst, s'q't)$ for every $t \in \Sigma$, together with the rule $(qs\$, s'q'b\$)$ for the blank symbol b. Similar rules encode the behaviour of left-moving instructions or instructions where the head doesn't move.

Aside from this basic model there are various more extended models containing multiple tapes, multiple heads on a single tape, semi-infinite tapes, multi dimensional tapes, or extensions of the instruction repertoire by allowing heads on the same tape to jump to each other's position in constant time. In the multi tape version there can be special tapes reserved for the input or output, subject to special restrictions like right moving heads only, no printing on input and no rewriting of any previously printed symbol on the output.

Fact is that all these models are equivalent from the perspective of Theoretical Computer Science; they all satisfy the Invariance Thesis stating that they simulate each other with polynomial time and constant factor space overheads. For a more detailed discussion of the issues involved I refer to my chapter in the 1990 Handbook of Theoretical Computer Science [24].

Turing and His Model

Given the large number of variations of the Turing Machine model, one might consider to go back to the master himself, and accept the definition given in his 1936 paper [21] to be the official one. This approach, however, is not going to work.

Turing's original paper reads as the work of an engineer or programmer avant la lettre, rather than that of a mathematician or logician. He starts with an intuitive description and justification (which is expanded upon in a later section of the paper). The model he describes is more general than the standard models in the literature since he allows a finite sequence of atomic instructions rather than a single atomic step to be the action provoked by a state-tape symbol observation. The reader should observe here that for Turing a configuration means such a state-tape symbol pair; what we call a configuration (also called instantaneous description) is called a full configuration by Turing. In the paper Turing proceeds by outlining a macro-language allowing for iterated instructions

in order to perform the required sweeps, copying actions, and searches for symbol occurrences which are found in all explicit programs for this device.

The model as presented is actually a linearisation of a two track single tape model where one track is used for the real data (digits 0 and 1) and the other track is used for auxiliary symbols called markers.

With all his flexibility there is an extension generally accepted today which is explicitly rejected: the use of nondeterminism. In the beginning of Section 2 Turing considers *ambiguous* configurations where the machine has a choice but such a choice should be made by an external operator. As argued elsewhere [17] nondeterminism became an accepted notion only in the 1950-ies presumably in order to obtain the desired characterisations in Automata Theory.

The more restricted model which allows atomic actions only is introduced for the purpose of constructing the Universal Turing Machine. This construction then opens the way for the famous results on the unsolvability of the Halting problem and the undecidability of the Hilbert Calculus. These sections read almost like a research plan outline, the details of which can be found in later textbooks like Kleene [9].

Using the Model

Why is the Turing Machine model so popular in Computer Science? It is not because of its close resemblance to real computers, since the Random Access Machine provides us with an idealised but more realistic model. It is also not popular because it is easy to write programs for Turing Machines. If you have ever taught a basic Theory class you will know that programming exercises on a Turing machine are very easy to state, but turn out to be rather cumbersome to solve, and boring to grade.

Instead I believe that the main reasons are that the model is very easy to understand and yet it is a model with universal computational power. Considering its conceptual simplicity it is hard to believe at first sight that the model is universal. But as it turns out it is not to hard to prove that (after having made the right choice of representation) one can simulate real computations as given by Kleene schemes of recursive functions, as well as alternative models of computation. The hardest part of proving the equivalence of the standard universal computational models after all is creating the coding mechanisms in Arithmetic (using Gödel numberings) in order to show that recursive function schemes can simulate symbol manipulating devices like Turing Machines.

There are a number of deep results on solving actual problems on Turing Machines. My favourite examples are the Hennie-Stearns oblivious simulation of multi tape machines on two tapes [6], Slisenko's algorithms for recognizing palindromes [16] and related string properties in real-time, and the the oblivious real-time minimal space simulation of a finite collection of counters by Vitányi [27]. However the dominant use of the Turing Machine model in Theory are negative results: Undecidability and/or Hardness results. Due to the undecidability of the Halting problem we can derive that every formal system capable of coding Turing machine computations in such a way that termination of these computations

becomes expressible within the system will inherit this curse of undecidability. Similarly any formalism which can express the fact that some Turing Machine computation terminates within a given number of steps will have a satisfiability or validity problem which can't be solved in much less steps, provided this expression is sufficient succinct. It is in the context of the construction of this type of reductions that the advantage of the combinatorial simplicity of the Turing Machine model (particularly in its most simple form: the single tape version) become prominent.

Reductions as sketched above are a main topic in Theoretical Computer Science. They are used as a tool for measuring the complexity of various problems. A problem A is reduced to a problem B if there exists some explicit, efficient and sufficiently succinct procedure for transforming instances of problem A into one (or several) instances of problem B such that solving the obtained B-instances will provide an effective answer to the original A-instance. The standard use of such a reduction is to show that problem B is difficult, by reducing some problem A to B where it is already known that A is difficult. But how do we know already that problem A is hard? This knowledge arises from the fact that we (or someone else) has previously reduced some other hard problem to A. This chain of reductions must originate in some problem generally accepted to be hard, and that role is played by termination problems on Turing Machines. It is therefore useful to single out these reductions which start with problems on Turing Machines; I will call them *Master Reductions*.

Representing Machine Configurations

In this presentation I will illustrate that, in order to obtain really efficient Master Reductions, one more ingredient is required: the choice of the right representation of Machine configurations.

A Turing Machine configuration is fully described by its three components: the state of the machine, the complete contents of the machine tapes, and the positions of the reading heads on the various tapes. Hence for a Mathematician it is evident how to represent this configuration: a tuple containing these three components is all we need, so the representation will become an object like $< q, x_1 x_2 \ldots x_k, i >$ with q denoting the state, $x_1 x_2 \ldots x_k$ denoting the tape contents and i denoting the head position. However, we have already encountered the more graphical representation which has become standard in our community. It is a representation where the state is inserted in the string of tape symbols, preceding the scanned symbol, or alternatively, printed above or below the scanned symbol (but this will make the printing and/or typesetting much harder). In our example configuration the representation will become $x_1 x_2 \ldots q x_i \ldots x_k$; moreover in order to make explicit where the used segment of the tape begins and ends this representation frequently is extended using endmarkers, resulting in a string like $\$x_1 x_2 \ldots q x_i \ldots x_k\$$.

In the sequel I will call representations of the first kind *Mathematical Representations*, whereas those of the second kind will be called *Intrinsic Representations*.

Given a sequence of successive configurations which occur during some computation, one can represent this section of the computation simply by the sequence of their encodings. However, a much clearer representation is obtained by writing these configurations below each other thus giving rise to the so called *time-space diagram* representation of the computation. Both representations can be used for this purpose.

If the mathematical representation is used it is evident what a proper allignment of these configurations should be: the content of some tape square may change, but the square itself maintains its identity. So each column in the diagram corresponds to a single fixed tape cell. To the left of the diagram one writes the state symbol and an index of the position of the tape head, which index also indicates the unique region in the diagram where during the transition connecting the two configurations changes may occur.

If we use the intrinsic representation we face the problem that the state symbol requires an additional position which moreover wanders through the configuration during the computation. One can solve this problem by introducing extra empty columns in the time-space diagram used only for storing the state symbol if the head arrives at that position. The easiest solution is to combine the state symbol and the scanned tape symbol into a pair which becomes a new symbol in an extended alphabet. Now the tape cells remain within their column in the diagram and yet the effect of the transition becomes entirely local.

There exist also versions of the time-space diagram where the state symbol/head position remains in a fixed column, while the tape symbols are moving around. This diagram illustrates a version of the machine where the head remains fixed but the tape moves; something which is not very realistic given the fact that the tape is supposed to be infinite.

In the literature one finds also some intermediate representation which I will call the *Semi-Intrinsic Representation*. Here the state symbol is written before the sequence of tape symbols, but the scanned tape cell is indicated by some marker (an arrow preceding the scanned symbol, or the scanned symbol is underlined).

I believe that the first time the advantage of the use of an intrinsic representation was explicitly mentioned in the literature is in a simple lemma (2.14) on page 38 in the thesis of Larry Stockmeyer [19]. Stockmeyer introduces for his standard model (the nondeterministic multi tape version with input and output tape) a version of the mathematical representation (page 20). Later he introduces the single tape version as a "technical useful model", and for encoding configurations of the latter model he uses a version of the intrinsic representation (page 34-35).

The lemma states that there exists some compatibility relation between triplets of the symbols used in the time-space diagram such that one row represents a proper successor configuration of the row above it, if and only if all triplet pairs formed by three successive symbols in the same position in the two rows

belong to this compatibility relation[1]. So consider three successive symbols in some row and the three symbols below it; if it is always the case that these triplets are compatible then the entire diagram describes a segment of the computation of our machine. Moreover, this compatibility relation is completely determined by the program of our machine. Note that this locality condition does not hold for the semi-intrinsic representation, since the state symbol information is located at a distance.

I now can state the thesis I want to discuss in this presentation: *For the construction of Master Reductions the Intrinsic Representation is by far more useful than the Mathematical Representation*. Stated otherwise: if you are looking for a Master reduction, use the intrinsic representation and life will be easy.

In the sequel of this paper we will illustrate the advantage of the intrinsic representation in relation to the following topics. We first reconsider the relation between machine computations and grammar derivations on which the fundamental characterisation of the Chomsky Hierarchy in basic Automata theory is based. Next we consider the two most common versions of a master reduction for the class NP: the Cook-Levin reduction to Satisfiability and the reduction to tiling problems. We discuss how Stockmeyer used his locality lemma in order to prove hardness results in the theory of (Extended) Regular Expressions. The final part of the paper illustrates the importance of the intrinsic configuration for proving that various models for Parallel Computation satisfy the Parallel Computation Thesis which states that such models recognize in polynomial time exactly what sequential models recognize in polynomial space. I hope that these examples which seem harder if not impossible to perform using a mathematical representation will convince the audience of the validity of my thesis.

2 The Chomsky Hierarchy and the Corresponding Automata

The core topic of an undergraduate course on Automata Theory is to provide a proof of the machine based characterisations of the four levels of the Chomsky Hierarchy: Regular Grammars vs. Finite Automata, Context Free Grammars vs. Push Down Machines, Context Sensitive Grammars vs. Linear Bounded Automata and finally Unrestricted Grammars vs. Turing Machines.

Proving these characterisations (once the required mathematical concepts have been introduced) requires a proof in two directions: one must show that the machine can simulate the grammar, and conversely that the machine computations can be simulated by grammar rules.

One may look therefore into the influence of the choice of representation of machine configurations on the proofs of these characterisations. It is evident that the intrinsic representation for this purpose is the right tool: individual transitions are fully described by context sensitive rules involving no more than

[1] Note that Stockmeyer speaks in this lemma about a compatibility *function*, but in his language functions are partial and multivalued so he intends this to be a relation.

three symbols on the left hand side (two symbols if the state symbol is paired with the scanned tape symbol - the second symbol is required for moving the head).

Given this insight the characterization of the type-0 languages becomes almost trivial. Turing machines are symbol manipulators, so it is not difficult - given some grammar - to write a Turing Machine program which starts out writing the start symbol, performing substitutions allowed by the grammar until the resulting string appears. The Turing machine can erase (or insert) symbols by shifting parts of the tape contents one square to the left (right). Conversely, given the fact that the machine configurations are derived by means of context sensitive rules, it is easy to construct a grammar which first generates an initial configuration and subsequently simulates the Turing Machine computation towards its accepting state. Since in this final configuration a substantial number of auxiliary symbols still may remain written on the tape, a final cleanup sweep where the undesired symbols are erased is required.

A similar proof will work for the context sensitive grammars vs. the linear bounded automata. However the prohibition of erasing rules requires a careful treatment of the boundary markers of the tape segment containing the input. These boundary markers are required since the machine must be capable of feeling the end of the tape, while on the other hand the machine is not allowed to leave the input string. This problem can be solved by pairing the end marker with the first(last) symbol and rewriting these marked symbols at the end of the production.

A comparable verbatim simulation between the machine configurations and the intermediate phrases of the derivation process is not possible for the two remaining cases of the context free and regular languages. The main reason is that in the grammar based world during the generation process only the initial part of the generated word is present, whereas the complete word exists already at the start of the machine computation.

One can however preserve the flavour of such a simulation. The problem is resolved by removing from the machine configuration the part of the input word which still has te be read in the future. The configuration consists of the part of the input already read (the part of the output already generated) followed by a machine state (nonterminal symbol). For the context free case this machine state is paired with the topmost stack symbol and the remaining stack symbols are concatenated in reverse order (paired up with the intermediate machine state attained when that stack symbol is eventually removed).

As is well known the choice freedom on the grammar side results in the machines becoming nondeterministic. This nondeterminism subsequently can be eliminated in the regular grammar case and for the unrestricted Turing Machines. For the context free grammar case nondeterminism has been shown to be required, whereas its necessity for the linear bounded machines is known as the famous LBA problem which still is unsolved.

We conclude that the intrinsic representation is used in Automata Theory as we know it today. This is not a formal proof that we can't build a version of

Automata Theory based on the mathematical representation, but let me just observe that I have never encountered such a treatment in the literature.

3 Master Reductions for NP

The two master reductions which I will investigate in this section are the Cook-Levin reduction to a version of the Satisfiability problem for Propositional Logic and the reduction based on Tilings.

Propositional logic is a language which is extremely flexible if you want to state properties of finite combinatorial structures, provided you are willing to introduce a sufficiently large collection of propositional variables. In the Cook-Levin reduction these variables encode the complete time-space diagram of an accepting Turing machine computation on the given input. The reduction is performed in such a way that it establishes a one-one correspondence between accepting computations and satisfying assignments of these propositional variables.

Let some language L in NP be accepted by some nondeterministic Turing Machine M in polynomial time. That means that for some input string x it holds that x belongs to L if and only if we can find a time-space diagram of size T by T which describes an accepting computation according to M where T is moreover bounded by $P(|x|)$ for some fixed polynomial P. The time-space diagram is encoded using propositional variables $p[i, j, k]$ expressing that *at position* $< i, j >$ *in the diagram symbol* σ_k *is written*.

The Cook-Levin formula is the conjunction of a collection of sub-formula's which express the required properties of the diagram like:

1. At every position in the diagram some symbol is written.
2. At every position in the diagram at most a single symbol is written.
3. The diagram starts with the encoding of the initial configuration on the input x.
4. The diagram terminates in some accepting configuration (which can be tweaked to be unique if one desires it to be so).
5. successive rows in the diagram are connected by legal transitions of the machine M.

If the intrinsic representation is used we know (by Stockmeyer' lemma) that the last condition can be expressed by enforcing the local compatibility condition on all 3 by 2 sub-windows in the diagram. This can be expressed by writing some clause excluding an illegal combination of symbols within such a window for all illegal combinations and all proper positions of this window in the diagram (a nice way of expressing this condition if one aims at obtaining a Cook-Levin formula in Conjunctive Normal Form).

It is not difficult to design a Cook-Levin formula in case the Mathematical Representation is used. In this case the state and the head position are denoted outside the diagram but we can introduce additional variables $s[i, l]$ expressing *at time* i *the machine is in state* q_l and $h[i, j]$ expressing *at time* i *the head is*

located at position j. The Cook-Levin formula now will include additional clauses expressing that at every time the state and head position are uniquely determined. The revised correctness conditions require that at some distance from the head position nothing changes and that the changes in the direct neighbourhood of the head positions conform to the given program. Details can be found in any textbook containing a full proof of the Cook-Levin result.

The question becomes whether there is an advantage here of using the intrinsic representation. I claim there is; it is recognized by a simple estimation of the size of the Cook-Levin formula's obtained.

For both representations the number of variables required is $O(T^2K)$ where K is some constant equal to the number of symbols which may occur in the time-space diagram. The number of additional state and head variables required for the Mathematical representation are of order $O(TK)$ and $O(T^2)$ respectively, and these numbers are small compared to the number of variables used for the diagram anyhow.

However if we consider the size of the various sub-formula's one observes that the five conditions in case we use the intrinsic representation are of sizes $O(T^2K)$, $O(T^2K^2)$, $O(T)$, $O(T)$ and $O(T^2K^6)$ respectively. However, when using the mathematical representation, the additional formula expressing the fact that the head always resides at a single position turns out to be of size $O(T^3)$ which is a factor T larger than all the other contributions and becomes the dominant term in the size estimate of the resulting formula in propositional logic (note that K is determined by the program only and is independent of the length of the input).

Hence the penalty for using the mathematical representation in the Cook-Levin result is that the size of the formula produced by the reduction becomes cubic rather than quadratic in the running time of the simulated machine. Yet, this unnecessary overhead has not prevented well known authors, including Cook [2] and Garey & Johnson [4] to use the mathematical representation for their proof of the Cook-Levin Theorem.

Tiling Reductions

The tiling reduction, used for NP-reductions originally by Levin [10] and Harry Lewis [11,12] is based on covering a region of the plane using square tiles which are divided in four triangles each being coloured. Tiles are to be selected from a fixed catalogue of tile types, and may not be rotated or reflected. When two tiles are placed adjacently (horizontally or vertically) the colours along a shared edge must be equal. Boundary conditions are enforced by fixing colours at the boundary of the region to be tiled; alternatively one can assign a first move to the devil by placing a single tile somewhere in the plane and demanding that the tiling must be extended to the full region.

Tilings allow a direct encoding of a time-space diagram if the Intrinsic Representation is used. The successive configurations appear encoded in colours along horizontal lines of the tiled region. We need tile types which express that a tape symbol is passed unchanged from one configuration to the next one. Other tile

types express directly the instructions of the program. A third class of tile types allows some tape symbol to become scanned in the next configuration if the head enters from an adjacent column. One must however restrict the Turing Machine program in order to ensure that the machine when moving to some state q can't move in both directions, since this would allow the creation and/or annihilation of phantom pairs of heads in the time-space diagram simulated by the tiling.

A more detailed description of the construction and its use can be found in [22,25]. The nice properties of the tiling reduction are that there is a complete separation between the encoding of the Turing Machine Program (which determines the catalogue of tile types) and the input (which is encoded in the boundary condition). If we allow the boundary condition to be specified in some more succinct form (I.E., if we can express the size of the boundary rather than listing all edge segments) the reduction shows hardness for higher complexity classes like PSPACE and NEXPTIME. Chlebus [3] has shown how alternating Turing Machines can be reduced to a two player game version of the Tiling problem.

As mentioned the tiling reduction works nicely for the intrinsic representation. It is not to difficult to design a tiling simulation for the semi-intrinsic representation (one uses signals transmitting the state information through a horizontal line) but I never have seen a simulation starting from the mathematical representation, which would require some internal mechanism for performing binary to unary conversion of numbers to start with.

Starting with the tiling reduction as a master reduction problems like Satisfiability but also Knapsack like problems are easily reached by further reductions [14]. But also a Hilbert 10 reduction can be obtained [22,25][2].

To my opinion the Tiling reduction is more suitable for educational use compared to the original Cook-Levin reduction. In my classes I have always used the example of a simple Turing Machine which increments a binary counter, a program of 6 instructions. The resulting catalogue of tile types contains 15 types. In 1992 my institute ordered the construction of a wooden demonstration model of the resulting puzzle to be used for educational events. I believe that it represents the most inefficient computer in the world which was ever built. After the move of the institute the puzzle was saved with my archives, but it is locked away in a storage room. The puzzle is available today in digital form on the web [26].

Note also that the combined reduction to Satisfiability using the tiling reduction as an intermediate step achieves the same $O(T^2)$ overhead which is obtained by the direct Cook-Levin reduction in case the intrinsic representation is used.

Our conclusion is that for NP-reductions the use of the Intrinsic Representation is not an absolute requirement, but the alternatives have some disadvantages.

[2] This reduction was originally constructed at a workshop in Paderborn in October 1982 in response and rebuttal to a presentation by J.P. Jones who presented with Yuri Matijasevič an improved version of the reduction of Machine termination to the solvability of exponential Diophantine Equations based on register machines, and claimed that such a reduction based on Turing Machines was not possible.

4 Stockmeyer and His Work on Regular Expressions

The standard theory of Regular Expressions deals with expressions generated by a grammar based on three types of generators and three operations. The generators are:

1. 0 denoting the empty language,
2. λ denoting the singleton language containing only the empty word,
3. σ for each σ in the alphabet Σ under consideration, denoting the singleton language containing the single letter word σ.

The operators are the + denoting union of languages, . for concatenation of languages and $*$ denoting the Kleene star iteration operation; the $*$ operator is monadic, whereas the + and . are binary operators.

Beyond this standard language of regular expressions a number of additional operators are considered by Stockmeyer: ^{2}, denoting Squaring, I.E., concatenation of a language with itself, $\cap$ denoting intersection and $\sim$ denoting complementation. It is known that the family of regular languages is closed under these operators, hence, in principle, regular expressions involving such operators can be rewritten into standard expressions. However there is no direct algebraic method for doing so. The detour by construction of the corresponding automata and deriving the regular expressions corresponding to these automata will produce unmanageable large expressions.

In chapter 4 of his thesis (the largest chapter in this book) Stockmeyer investigates how these additional operators affect the complexity of decision problems on generalized Regular Expressions. Decision problems considered are:

1. $NEC(\phi, \Sigma)$: does the expression ϕ denote the set of all possible words over the alphabet Σ?
2. $EQ(\phi, \psi), INEC(\phi, \psi)$: do the two expressions denote the same (different) languages?

Evidently these problems are inter-reducible, provided operators like complementation and intersection are available, but since also languages without these operators are considered we need them all.

The hardness results in this chapter are obtained by a master reduction. Consider a rectangular time-space diagram of an accepting computation of some nondeterministic single tape Turing Machine. The correctness of such a diagram is expressed by the conjunction of a number of conditions expressing syntactic well-formedness (consisting of the right sort of symbols in the right positions), correct start (with the intended initial configuration on the input word), correct termination (in some final accepting configuration), and correct computation (enforced by application of Stockmeyer's 3 by 2 window compatibility check throughout the diagram).

The diagram is a two dimensional object, but it can be linearised into a string by printing all rows in the diagram behind each other, separated by a suitable extra marker. So one might look for some generalized regular expression

describing precisely those strings which encode a correct time-space diagram. Note that we now must enforce the additional condition that the segments in the linearised diagram all should have the same length.

We need our expression to encode the conjunction of all the conditions which must be enforced. This is hard to express if we don't have the operator of intersection in our language. Therefore Stockmeyer migrates to the complementary world where he constructs an expression which intends to denote all strings which *fail* to encode a correct time-state diagram. The expression becomes a *Syllabus Errorum* stating all possible sources of an error. This explains the use of the decision problem NEC in his investigations: if the accepting time-space diagram exists there exists an error-free string, and therefore the described language will have a non-empty complement. Otherwise all strings are erroneous and the expression will be equivalent to the language Σ^*.

The hardest error type to be described is a violation of the 3 by 2 window compatibility relation. In the time-space diagram the symbols are written closely together but in the linearisation they are separated by a substring whose length is equal to the width of the diagram (up to a small additive constant). This explains the importance of yardsticks: sub-expressions of the form Σ^K for large values of K. Since the width of the time-space diagram equals the space consumed by the simulated computation it becomes relevant to invent succinct representations for these yardsticks: the more succinct such a representation becomes the higher the (nondeterministic) spacebound for which a hardness proof is obtained.

If we have no additional operators the size of the expression for a yardstick is linear in K. Thus hardness is obtained for linear bounded automata. Please keep in mind that at the time the thesis was written the Immerman-Szelepsényi result [7,20] yielding closure under complementation of nondeterministic space bounded complexity classes had not yet been proven, whence Stockmeyer had to navigate carefully around issues involving complementation. Today we understand his result as a proof of hardness for PSPACE.

Adding the operation ^{2} of squaring reduces the size of the expression for a yardstick to $(O(log(K))$, and hardness for NEXPSPACE is obtained (by Savitch' result [15] NEXPSPACE = EXPSPACE). Removing the $*$ operator eliminates the possibility to talk about arbitrary long computations, and therefore hardness results are obtained for nondeterministic time classes (NP respectively NEXPTIME depending on whether squaring is available or not). The hardest part of the theory is section 4.2 where the impact of the complementation operator is shown: each increase by one in the complementation depth of the regular expressions allows for an exponential increase of the succinctness of the yardstick expression. Therefore the hardness results are raised to non-elementary space and/or time bounded complexity classes.

From our perspective the key ingredient in all constructions is the encoding of a compatibility violation in the diagram by an expression listing the violating pair connected by a yardstick expression. This simulation is made possible by the use of the intrinsic representation and it must be hard if not impossible to obtain a similar construction based on the mathematical representation.

5 The Impact of the Intrinsic Representation on Machine Models in the Second Machine Class

The Second Machine Class [23,24] consists of those models for machines supporting some form of parallel processing for which the *Parallel Computation Thesis*, expressed by the equalities //PTIME = //NPTIME = PSPACE is true: what the parallel model can do in polynomial time, deterministically or nondeterministically, is what can be achieved in the sequential world in Polynomial Space.

Machine models of this nature were investigated in the 1970-ies. There are various parallel versions of the Random Access Machines, and versions of Turing Machines supporting parallel branching. More surprising was the discovery that some sequential models which may operate on very large data objects also are second machine class members: typical examples are the Vector Machines introduced by Pratt and Stockmeyer [13] and the Random Access machine extended with multiplicative instructions described by Hartmanis and Simon [5]. Also the Alternating Turing Machine [1] belongs to this class, be it that there exists no nondeterministic version of this device.

Proving that some device indeed satisfies the above equalities uses some methods which by now have been well understood. The inclusion //NPTIME $\subseteq$ PSPACE is shown by guessing an accepting computation trace of the parallel device, and validating this trace using some recursive procedure which will evaluate the state of the elementary hardware components of this device at any time during the computation. A key argument is that the parameters of such a recursive procedure can be written down in polynomial space.

Such a proof can be given only when the parallel model is reasonable: it can activate in polynomial time an exponential amount of hardware (but not more) and the nondeterminism must be Uniform (the same choices are made on all parallel paths in the computation).

For the inclusion PSPACE $\subseteq$ //PTIME nowadays various strategies are avialable: one can show that the parallel device can simulate an Alternating machine, or one can construct a Polynomial Time algorithm for the PSPACE complete problem QBF [18]. However in the mid 1970-ies the Alternating machine had not yet been invented and nobody had proposed the idea of exploiting the QBF problem. The early proofs were all based on a master simulation of a PSPACE bounded Turing machine on the parallel machine.

The idea used in this master simulation is the reduction of the existence of an accepting computation to a connectivity problem on a huge (exponentially large) *Computation graph*. This graph has all possible configurations of the Turing Machine on the allowed amount of space as nodes, and the transitions between these configurations as edges. The initial configuration in the graph is just some special node, and so is the final accepting configuration (which may be assumed to be unique). Computations become paths in this computation graph. Hence the existence of an accepting computation is reduced to the existence of a path connecting the start node with the target node.

This connectivity problem can be solved by computing the transitive closure of the relation given by the edges (transitions). A convenient algorithm for computing this transitive closure uses the mathematical representation of the *Adjacency Matrix*: row and column indices represent nodes (configurations) and the presence of an edge from node i to node j is denoted by assigning the value 1 to matrix element at position $< i, j >$. The diagonal entries in the matrix obtain also value 1 (every node is reachable from itself by a path of length 0).

By iteratively squaring this matrix (over the Boolean algebra where $1.1 = 1 + 1 = 1$) one determines which pairs of nodes in the graph are connected: after t iterations all connections by some path of length $\leq 2^t$ are found. Since cycles don't contribute to connections and the number of the nodes is bounded by $2^{O(spacebound)}$ a polynomial number of iterations is sufficient. At the end of the computation the answer is found in the desired matrix element; the rest of the matrix is discarded.

The details of this simulation depend on the precise model considered. Generic tasks are the construction of some object representing a list of all integers in the range $0 \ldots 2^M$ for some large value of M. The entries in this list represent all possible configurations of the machine. Think of the numbers as being written down as binary numbers and consider the resulting bit-string to be the representation in binary of the string of symbols in the configuration. There is no guarantee that these numbers (digit strings) satisfy reasonably syntactic conditions like not containing more than one state symbol. However, getting rid of such junk configurations is not needed; they can't do any harm. In fact removing them may be harmful in the sequel of the proof, because it would create gaps in the sequence of configurations at positions which are hard to predict.

The next task is to construct the Cartesian product of this list with itself, yielding an object storing all configuration pairs.

Given this object we must determine for all these pairs of configurations whether they are equal or connected by a transition or not. Moreover this has to be done in parallel, given the exponential size of this object. This is precisely the point where it is crucial that these numbers are understood to encode configurations in the intrinsic representation. Equality is easy to test but the test for a transition requires that in the binary representation the digit block is identified where the two strings are different. This block represents the three symbols of Stockmeyer' 3 by 2 window. The contents of these blocks in the two configurations must obey the compatibility relation. Moreover, outside these blocks the two configurations must be identical.

Once this test has been performed the Adjacency matrix is obtained. The computation then can proceed by implementing the iterated multiplication of the matrix with itself. This is yet another complex task, but it is less model dependent.

The details of the above computation are different for parallel versions of the Turing Machine (which is symbol manipulation oriented) and the Parallel Random Access devices. For the RAM based models one must invoke some

mechanism which will allow an efficient method for converting numbers into bit-strings. Inspection of the constructions proposed in the literature shows that for all RAM based parallel models some form of string manipulation or some mildly multiplicative operation like division by 2 is inserted in the instruction code. Such instructions are not available in the basic RAM model - the result by Hartmanis and Simon indicate that you can't add to much multiplicative power to the RAM without creating a model which is to powerful.

Our conclusion is that in these early simulations the fact that the numbers encode configurations in the intrinsic representation is a key ingredient for the correctness of the proof. It seems hard, if not impossible to find such a construction if the Mathematical representation is used.

6 Conclusion - Is There a Dragon Out There?

I hope that the examples in the preceding sections have convinced the reader that the use of the Intrinsic Representation of Turing Machine configurations has been the enabler for several fundamental results in Theoretical Computer Science. The question remains whether this observation should affect our behaviour as theoreticians. Stated otherwise: do we need to start a Crusade against the use of the Mathematical Representation? Is there a dragon out there which should be slayed?

While preparing this presentation I have searched the leftovers of what in the past used to be a well equipped Mathematical Library in my institute[3]. Inspection of some 25 textbooks on introduction in Computer Science or Computation Theory yielded the following results: Many authors give no formal definition of a configuration but informally they present something resembling the intrinsic or a semi-intrinsic representation. This also holds for the Wikipaedia page on Turing machines. I found a formal definition of the Mathematical representation for single tape machines only in the 1969 edition of Hopcroft and Ullman, and in the 1981 edition of Lewis and Papadimitriou. The later authors however immediately continue with a semi-intrinsic representation as an illustrative tool - no wonder, since in this textbook a master reduction based on tilings is presented. Other authors give the Mathematical representation for multi tape machines but move towards the intrinsic representation for the single tape model, and that is the model used in all the hardness and undecidability proofs. Turing himself uses an Intrinsic representation by way of illustration. So do Kleene and Davis.

Evidently in practice our colleagues have throughout the last 70 years followed their intuition and have made the right choice. But except for Stockmeyer I have not found anybody who explicitly has looked into the advantages of this decision.

The conclusion is that we can continue and live and work in peace. Dragons remain a rare species which should be protected rather than persecuted.

[3] Victim of the curse of digitalisation.

References

1. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. Assoc. Comput. Mach. 28, 114–133 (1981)
2. Cook, S.A.: The complexity of theorem proving procedures. In: Proc. ACM Symposium Theory of Computing, vol. 3, pp. 151–158 (1971)
3. Chlebus, B.G.: Domino-tiling games. J. Comput. Syst. Sci. 32, 374–392 (1986)
4. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Co. (1979)
5. Hartmanis, J., Simon, J.: On the power of multiplication in random access machines. Proc. IEEE Switching and Automata Theory 15, 13–23 (1974)
6. Hennie, F.C., Stearns, R.E.: Two-way simulation of multi-tape Turing machines. J. Assoc. Comput. Mach. 13, 533–546 (1966)
7. Immerman, N.: Nondeterministic space is closed under complementation. SIAM J. Comput. 17, 935–938 (1988)
8. Jones, J.P., Matijasevič, Y.V.: Register machine proof of the theorem on exponential Diophantine representation of enumerable sets. J. Symb. Logic 49, 818–829 (1984)
9. Kleene, S.C.: Introduction to Metamathematics. Noth Holland Publ. Cie (1952)
10. Levin, L.A.: Universal'nie zadachi perebora. Problemi Peredachi Informatsie IX, 115–116 (1973) (in Russian)
11. Lewis, H.R.: Complexity of solvable cases of the decision problem for the predicate calculus. In: Proc. IEEE FOCS, vol. 19, pp. 35–47 (1978)
12. Lewis, H.R., Papadimitriou, C.H.: Elements of the Theory of Computation. Prentice-Hall (1981)
13. Pratt, V.R., Stockmeyer, L.J.: A characterization of the power of vector machines. J. Comput. Syst. Sci. 12, 198–221 (1976)
14. Savelsberg, M.P.W., van Emde Boas, P.: BOUNDED TILING, an alternative to SATISFIABILITY? In: Wechsung, G. (ed.) Proc. 2nd Frege Memorial Conference, Schwerin. Mathematische Forschung, vol. 20, pp. 401–407. Akademie Verlag (September 1984)
15. Savitch, W.J.: Relations between Deterministic and Nondeterministic tape Complexities. J Comput. Syst. Sci. 12, 177–192 (1970)
16. Slisenko, A.O.: A simplified proof of the real-time recognizability of palindromes on Turing Machines. J. Mathematical Sciences 5(1), 68–77 (1981)
17. Spaan, E., Torenvliet, L., van Emde Boas, P.: Nondeterminism, fairness and a fundamental analogy. EATCS Bulletin 37, 186–193 (1989)
18. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time. In: Proc. ACM STOC, vol. 5, pp. 1–9 (1973)
19. Stockmeyer, L.: The complexity of decision problems in automata theory and logic, Report MAC-TR-133, MIT (1974)
20. Szelepsényi, R.: The method of forcing for nondeterministic automata. Bull. EATCS 33, 96–100 (1987)
21. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proc. London Math. Soc. Ser. 2 42, 230–265 (1936)
22. van Emde Boas, P.: Dominoes are forever. In: Priese, L. (ed.) Proc. 1st GTI Workshop, Paderborn, Rheie Theoretische Informatik UGH Paderborn, vol. 13 (1982)
23. van Emde Boas, P.: The second machine class 2, an encyclopedic view on the parallel computation thesis. In: Rasiowa, H. (ed.) Mathematical Problems in Computation Theory, vol. 21, pp. 235–256. Banach Center Publications (1987)

24. van Emde Boas, P.: Machine models and simulations. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. A, pp. 3–66. North Holland Publ. Comp. (1990)
25. van Emde Boas, P.: The convenience of tiling. In: Sorbi, A. (ed.) Complexity, Logic and Recursion Theory. Lect. Notes in Pure and Appled Math., vol. 187, pp. 331–363 (1997)
26. van Emde Boas, H.: Turing Tiles, Web application located at, http://www.squaringthecircles.com/turingtiles/
27. Vitányi, P.M.B.: An optimal simulation of counter machines. SIAM J. Comput. 14, 1–33 (1985)

What Is an Algorithm?

Yuri Gurevich

Microsoft Research
gurevich@microsoft.com

Abstract. We attempt to put the title problem and the Church-Turing thesis into a proper perspective and to clarify some common misconceptions related to Turing's analysis of computation. We examine two approaches to the title problem, one well-known among philosophers and another among logicians.

We must, incidentally, make it clear from the beginning that if a thing is not a science, it is not necessarily bad. For example, love is not a science. So, if something is said not to be a science, it does not mean that there is something wrong with it; it just means that it is not a science.

Richard Feynman

1 Introduction

Two articles in a recent book [10] present two approaches to the title problem and offer different answers. Article [19] presents an approach developed by Yiannis Moschovakis. "A characteristic feature of this approach is the adoption of a very abstract notion of algorithm that takes *recursion* as a primitive operation and is so wide as to admit 'non-implementable' algorithms" [19]. The article starts thus.

> In the sequence of articles . . . , Moschovakis has proposed a mathematical modeling of the notion of algorithm — a set-theoretic "definition" of algorithms, much like the "definition" of real numbers as Dedekind cuts on the rationals or that of random variables as measurable functions on a probability space.

We discuss this definition of algorithms in §6.

Article [22] presents an approach originally developed by Robin Gandy, a student of Alan Turing, in a 1980 article [9]. Gandy intended to complement Turing's analysis of human computers with an analysis of computation by mechanical devices. He came up with an axiomatically defined class of computation devices, later named Gandy machines. The approach was adopted by Wilfried Sieg. In article [22], Sieg uses Gandy machines to "dispense with [Church's and Turing's] theses". The article starts thus.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 31–42, 2012.

> Church's and Turing's theses dogmatically assert that an informal notion of effective calculability is adequately captured by a particular mathematical concept of computability. I present an analysis of calculability that ... dispenses with theses. ... The analysis leads to axioms for discrete dynamical systems (representing human and machine computations) and allows the reduction of models of these axioms to Turing machines.

We discuss this axiomatization of discrete dynamical systems and dispensing with the theses in §4.

In §2, we discuss whether it is possible at all to define algorithms. (There is also a question why bother to define algorithms. Well, understanding what algorithms are should — and does — have practical applications, to software specification and model-based testing in particular, as well as theoretical application, like semantics of software or algorithmic completeness of computation models. But that is a different issue to be addressed elsewhere.)

In §3, we discuss and clarify a couple of misconceptions related to Turing's analysis of computations. In §4 we discuss Gandy machines. In §5, we discuss what kind of entities algorithms are; this discussion is closely related to §6 where we discuss Moschovakis's recursor theory.

This article can be seen as a companion to our older article [5].

2 Can the Notion of Algorithm Be Rigorously Defined?

Two articles [19] and [22], mentioned in §1, give different answers to the question in the title of this article. The two answers are not at all equivalent. A question arises whether the notion of algorithm can be defined at all. The answer is yes and no. Let us explain.

The Negative Answer

In our opinion, the notion of algorithm cannot be rigorously defined in full generality, at least for the time being. The reason is that the notion is expanding.

Concerning the analogy of algorithms to real numbers, mentioned in §1, Andreas Blass suggested a better analogy: algorithms to numbers. Many kinds of numbers have been introduced throughout history: positive integers, natural numbers, rationals, reals, complex numbers, quaternions, infinite cardinals, infinite ordinals, etc. Similarly many kinds of algorithms have been introduced. In addition to classical sequential algorithms, in use from antiquity, we have now parallel, interactive, distributed, real-time, analog, hybrid, quantum, etc. algorithms. New kinds of numbers and algorithms may be introduced. The notions of numbers and algorithms have not crystallized (and maybe never will) to support rigorous definitions.

The Positive Answer

But the problem of rigorous definition of algorithms is not hopeless. Not at all. Some strata of algorithms have matured enough to support rigorous definitions.

This applies to classical (or classical sequential or just sequential) algorithms, essentially the only algorithms in use from antiquity to the 1950s. "Algorithms compute in steps of bounded complexity", wrote Andrei Kolmogorov in 1953 [14]. This is a good informal definition of sequential algorithms.

An axiomatic definition of sequential algorithms have been given in [12]. That definition was used to derive the Church-Turing thesis from first principles in [8]. The derivation presumes that, at the time, Church and Turing (and Gödel and other experts) had in mind only sequential algorithms, which we believe they did. The axiomatic definition was extended to synchronous parallel algorithms in [3] and to interactive sequential algorithms in [6,7].

The Status of the Church-Turing Thesis

As far as the input-output relation is concerned, synchronous parallel algorithms and interactive sequential algorithms can be simulated by Turing machines. This gives additional confirmation of the Church-Turing thesis.

None of the other known kinds of algorithms seem to threaten the thesis but the thesis has not been dispensed with and probably never will be. The question whether some algorithm of a currently unknown kind would allow us to compute a function from natural numbers to natural numbers that is not Turing computable remains open, possibly forever. And even if we had a satisfactory axiomatic definition of algorithms in full generality, the thesis would not be dispensed with. It would be just reduced to the first principles enshrined in the axioms.

3 Remarks on Turing's Analysis of Computation

Turing's analysis of computation [24] was a stroke of genius. The analysis is extremely famous, and yet it is often misunderstood.

Some people think that every computable function, total or partial, can be computed by a Turing machine. This is not so, and here are some counter-examples. Consider Euclid's algorithm for computing the greatest common divisor $d = \gcd(a, b)$ of two natural numbers a, b.

```
let M = max(a, b),  m = min(a, b)
while M > m do
    M, m := max(M − m, m), min(M − m, m)
d := M.
```

The gcd function on natural numbers is of course Turing computable, but the algorithm was also applied — in theory and in practice — to the lengths of segments of a straight line, which gives rise to a computable partial function

(the algorithm does not terminate if the two given lengths are incommensurate) that is not Turing computable because you cannot place an arbitrary length on the Turing tape. More generally, the functions computed by ruler-and-compass algorithms are not Turing computable. And let us emphasize that ruler and compass were practical tools in ancient Greece and that a number of ruler-and-compass algorithms were practical algorithms.

It is common in mathematics to consider algorithms that work on abstract objects. The functions computed by these algorithms may not be Turing computable. One example is Gaussian elimination. Here is another example: a bisection algorithm that, given a real $\varepsilon > 0$ and a continuous function f on a real segment $[a, b]$ with $f(a) < 0 < f(b)$, computes a point $c \in [a, b]$ with $|f(c)| < \varepsilon$.

```
while |f((a + b)/2)| ≥ ε do
    if f((a + b)/2) < 0 then a := (a + b)/2 else b := (a + b)/2
c := (a + b)/2
```

One can argue that these functions are not truly computable, that in practice we can only approximate them. But then there are analog computers *in practical use* that work in real time and compute functions that are not Turing computable.

Of course Turing would not be surprised by our examples. He explicitly restricted his analysis to "symbolic" (symbol-pushing, digital) algorithms. He implicitly restricted his analysis to sequential algorithms, essentially the only algorithms in his time. It is interesting that it turned out easier to axiomatize all sequential algorithms [12], whether symbolic or not, including the ruler-and-compass algorithms, Gaussian elimination and the bisection algorithm (but excluding analog algorithms which are not sequential in our sense).

What about quantum algorithms? Do they compute functions that are not Turing computable? Erich Grädel and Antje Nowack checked that the quantum computing models in the literature can be faithfully simulated by parallel abstract state machines [11]. And, as we mentioned above, functions computed by parallel ASMs are Turing computable.

There is also a rather common misunderstanding that Turing defined the notion of algorithm, albeit restricted to symbolic sequential algorithms. Let us restrict attention to such algorithms for a moment. Suppose that your computation model (e.g. a programming language) is Turing complete. Does it mean that the model allows you to express all algorithms? Not necessarily. Turing machines simulate faithfully only the input-output behavior of algorithms. But there may be much more to algorithms than their input-output behavior. Turing completeness does not mean algorithmic completeness. It means only that, for every Turing computable function f, the language allows you to program an algorithm that computes f.

For illustration consider Turing machines with one tape that may be multidimensional. The model is obviously Turing complete. On any such machine, the problem of palindrome recognition requires $\Theta(n^2/\log n)$ time [2]. But the

problem is trivially solvable in linear time on a Turing machine with two one-dimensional tapes. For a deeper dive into algorithmic completeness, see [26, §3].

4 Gandy's Analysis of Mechanisms

Robin Gandy argues in [9] that "Turing's analysis of computation by a human being does not apply directly to mechanical devices." The reason is that humans compute sequentially but machines can perform parallel computations. In this connection, Gandy analyzed computations by mechanical devices and introduced (what we call now) Gandy machines.

> A set-theoretic form of description for discrete deterministic machines is elaborated and four principles (or constraints) are enunciated, which, it is argued, any such machine must satisfy. ... It is proved that if a device satisfies the principles then its successive states form a [Turing] computable sequence. [9, p. 123]

Note "successive states". Gandy machines work in sequential time. This type of parallelism is called synchronous. In the rest of this section, parallelism will by default be synchronous.

Gandy pioneered the use of axioms in the analysis of computation. He came up with four principles (or constraints, or axioms) satisfied, he claimed, by all discrete deterministic machines. Contrast this with Turing's analysis. While Turing's analysis was convincing, it is hard to isolate first principles that, in Turing's opinion, are satisfied by all symbolic sequential computations.

Wilfried Sieg adopted Gandy's approach and reworked Gandy's axioms; see [23] and references there.

Critical Remarks

In a 2002 article [20], Oron Shagrir suggests that "there is an ambiguity regarding the types of machines that Gandy was postulating". He offers three interpretations: "Gandy machines as physical machines", "Gandy machines as finite-physical machines", and "Gandy machines as a mathematical notion". Shagrir concludes that none of the three interpretations "provides the basis for claiming that Gandy characterized finite machine computation." This agrees with our own analysis. By the way, for our purposes, there is no difference between Gandy's original axioms and Sieg's versions of the axioms. So we will just speak about Gandy's axioms.

• *What real-world devices satisfy Gandy's axioms?* Probably very few do. One problem is the form of the states of Gandy machines: a collection of hereditary finite sets. Another problem is the requirement that state transitions are synchronous. You, the reader, may say that we have not proved our point. Well, the burden of proof is on the proponents of the approach. And there are precious few examples in the papers of Gandy and Sieg, and none of the examples is a

real-world device. The most prominent example in Gandy's paper is the cellular automaton known as Conway's game of life. Note that a cellular automaton can grow without any bound. In the real-world, such a cellular automaton would not stay synchronous.

It seems obvious that Gandy abstracts from material and views discrete deterministic machines as algorithms, abstract algorithms. So Gandy's claim can be restated thus: parallel algorithms satisfy the axioms.

• *What algorithms satisfy Gandy's axioms?* Typical parallel or even sequential algorithms do not satisfy the axioms. Consider for example a factorial algorithm. The state of the algorithm is naturally infinite and consists of natural numbers. There is of course a Gandy machine that simulates the factorial algorithm. Note that, in addition to simulating the factorial algorithm, the simulating machine may be forced to construct set representations of additional numbers.

In our view, Gandy's axioms are really used just to define another parallel computation model. (By the way, it is our ambition in [3] that parallel algorithms, on their natural abstraction levels, satisfy our axioms.)

• *How does Gandy's parallel computation model compare to other parallel computation models?* By now, there are numerous models of synchronous parallelism in the literature, e.g. parallel random access machines, circuits, alternating Turing machines, first-order logic with the least fixed-point operator, and parallel abstract state machines. What are the advantages, if any, of Gandy's model over the other models? Neither Gandy nor Sieg addressed this question. Gandy's model seems quite awkward for programming or specifying algorithms.

• *Dispensing with the Church-Turing thesis* Gandy proved that his machines can be simulated by Turing machines. This is another confirmation of the Church-Turing thesis. But is it a good ground for dispensing with the thesis? We do not think so, even if we restrict attention to parallel algorithms and forget other kinds of algorithms. By the first two bullets above, Gandy's theorem does not imply that his axioms are satisfied by all discrete mechanical devices or by all parallel algorithms.

5 What Kind of Entities Are Algorithms?

One point of view is that the question about algorithm entities is of no importance. We quoted already in §1 that "Moschovakis has proposed ... a set-theoretic 'definition' of algorithms, much like the 'definition' of real numbers as Dedekind cuts" [19]. The quotation marks around the word definition make good sense. There is another familiar definition of real numbers, as Cauchy sequences. Dedekind cuts and Cauchy sequences are different entities, yet the two definitions are equivalent for most mathematical purposes. The question of importance in mathematics is not what kind of entities real numbers are but what structure they form. Either definition allows one to establish that real numbers form a complete Archimedean ordered field.

The analogy in the quotation is clear: concentrate on mathematical properties of algorithms rather than on what kind of entities they are. The analogy makes good sense but it is far from perfect because much more is known about algorithm entities than real-number entities. Let us sketch another point of view on algorithm entities.

Consider algorithms that compute in sequential time. This includes sequential algorithms as well as synchronous parallel algorithms. A sequential-time algorithm is a state transition system that starts in an initial state and transits from one state to the next until, if ever, it halts or breaks. The very first postulate in our axiomatizations of sequential and synchronous parallel algorithms [12,13] is that the algorithms in question are sequential time.

The question arises what kind of entities states are. In our view, rather common in computer science, algorithms are not humans or devices; they are abstract entities. According to the second postulate in the axiomatizations of sequential and synchronous parallel algorithms, the states are (first-order) structures, up to isomorphism. This admittedly involves a degree of mathematical modeling and even arbitrariness. A particular form of structures is used; why this particular form? But this is a minor detail. Structures are faithful representations of states, and that is all that matters for our purposes. It is convenient to declare that states *are* structures, up to isomorphism; but there is no need to do so.

The point of view that sequential-time algorithms are state transition systems extends naturally to other classes of algorithms. In particular, a sequential-time interactive algorithm (until now we considered non-interactive algorithms) is a state transition system where a state transition may be accompanied by sending and receiving messages. A distributed algorithm is an ensemble of communicating sequential-time interactive algorithms.

6 Moschovakis's Recursion-Based Approach

We start with basics. In recursion-based approaches you write recursive equations that specify a function. Typically the equations define a monotone operator, and semantics is given by means of the least fixed point of the operator. For example, equations

$$\exp(x+1,0) = 1$$
$$\exp(x,y+1) = \begin{cases} 0 & \text{if } x = 0 \\ x \times \exp(x,y) & \text{if } x > 0 \end{cases}$$

specify exponentiation $\exp(x,y) = x^y$ on natural numbers. The equations define a monotone operator on extensions of the standard arithmetical structure with partial binary function exp. Accordingly the following process gives meaning to exponentiation. Initially exp is nowhere defined. Apply the equations, obtaining $\exp(x,0) = 1$ for every $x > 0$; then apply the equations again, obtaining additionally $\exp(x,1) = x$ for all x, and so on. After ω steps (where ω is the first infinite ordinal), you reach a fixed point; now $\exp(x,y)$ is defined for all x, y

except $x = y = 0$. Often the evolution toward the fixed point involves not only the function that you are computing but also some auxiliary functions.

In 1934, Gödel formulated a recursion-based calculus of (in general partial) numerical functions. Gödel's calculus can be seen as a specification language where a specification of a function f is a system of recursive equations that, taking into account some global conventions, suggests a particular (possibly inefficient) way to compute f. Church's thesis (extended to partial functions by Kleene) asserts that every "effectively calculable", that is computable by an algorithm, function on natural numbers is programmable in Gödel's calculus.

Recursive specification of functions has much appeal. It is declarative and abstracts from computation details. It is often concise. There has been much progress since the 1930s. Logicians developed recursion theory. McCarthy created a functional (that is recursion-based) programming language LISP, and many other functional languages followed.

The key ideas of Moschovakis's approach appear already in the 1984 article [16] that seems to be the very first publication on the subject.

> If, by Church's Thesis the precise, mathematical notion of *recursive function* captures the intuitive notion of *computable function*, then the precise, mathematical notion of *recursion* ... should model adequately the mathematical properties of the intuitive notion of *algorithm*. [16, p. 291]

Moschovakis discusses Euclid's algorithm for the greatest common divisor of two natural numbers. Then he says:

> Following the drift of the discussion, we might be expected at this point to simply identify the Euclidean algorithm with the functional `gcd`. We will not go quite that far, because the time-honored intuitive concept of algorithm carries many linguistic and intensional connotations (some of them tied up with *implementations*) with which we have not concerned ourselves. Instead we will make the weaker (and almost trivial) claim that *the functional* `gcd` *embodies all the essential mathematical properties of the Euclidean algorithm*. [16, p. 295]

He gives recursive equations for the mergesort algorithm on a set X and proceeds to prove that at most $n \cdot \log_2(n)$ comparisons are required to sort n elements.

Moschovakis's views have been evolving.

> When algorithms are defined rigorously in Computer Science literature (which only happens rarely), they are generally identified with abstract machines, mathematical models of computers. ... My aims here are to argue that this does not square with our intuitions about algorithms and the way we interpret and apply results about them; to promote the problem of defining algorithms correctly; and to describe briefly a plausible solution, by which algorithms are recursive definitions while machines model implementations, a special kind of algorithms. [17, p. 919].

The main technical notion in Moschovakis's approach is that of *recursor* which is a generalization of function specification in Gödel's calculus. The most recent

published definition of recursor is found in [19, p. 95]. Semantics is given by means of the least fixed point of a monotone operator. In some cases, the least fixed point is not achieved in $\leq \omega$ steps; then the recursor is *infinitary* and cannot be implemented by abstract machines. For illustration, see "the infinitary Gentzen algorithm" in [18]. Moschovakis formulates this slogan:

> The theory of algorithms is the theory of recursive equations. [18, p. 4]

Critical Remarks

• *Recursors vs. algorithms* We think that Moschovakis was right the first time around, in [16, p. 295] when he refrained from identifying (what he later called) recursors with algorithm "because the time-honored intuitive concept of algorithm carries many linguistic and intensional connotations" which are contrary to such identification.

Consider the system of two recursive equations (and thus a recursor) for the exponentiation in the beginning of this section. Is it an algorithm or not? The recursor certainly looks like an algorithm, and in many functional programming languages, this recursor would be a legitimate program (modulo syntactic details of no importance to us here). Typically $\exp(x^y)$ would be interpreted as a function call and, for example, the evaluation of 3^2 would proceed thus:

$$3^2 = 3 \cdot 3^1 = 3 \cdot (3 \cdot 3^0) = 3 \cdot (3 \cdot 1)) = 3 \cdot 3 = 9.$$

But the recursor theory is different. The meaning of a recursor is given by the least fixed point construction, and there is *nothing else*. In the case of the exponentiation recursor, the only "computation" is the process that we described above: start with the nowhere defined exp function, compute $\exp(x^0)$ for *all* $x > 0$, compute x^1 for *all* x, etc. What should we do in order to compute 3^2? Should we wait until the "computation" of exp is completed and then apply exp, or should we wait only to the end of stage 3 when all x^2 are computed? The recursor theory says nothing about that.

It is not our goal to make the recursor theory look ridiculous. In fact we agree that recursors are useful for mathematical analysis of algorithms. We just see no good reason to identify them with algorithms. Paraphrasing Richard Feynman, if thing is not an algorithm, it is not necessarily bad.

• *The abstraction level of imperative algorithms* It seems to us that recursor theorists underestimate the abstraction capabilities of imperative programming. Imperative programs, and in particular abstract state machines, can be as abstract as needed. We addressed this point once [4]. Here let us just quickly say this. Yes, an algorithm comes with a program for executing the algorithm. But this does not mean that the program necessarily addresses low-level computational details. Every algorithm operates on its natural level of abstraction. This level may be very low but it may be arbitrarily high.

• *Declarative specifications* Recursion is appealing. A part of the appeal comes from the declarative nature of recursion. That declarative nature is by itself a limitation for software specification; and note that every piece of software is an algorithm. Declarative specification of software was very popular in the 1980s and 1990s, but it was discredited to a large extent. As software is developed, it evolves. A book with a declarative specification quickly becomes obsolete. If specification is not executable, you cannot experiment with it.

• *Recursion is but one aspect of an algorithm* The theory of algorithms does not reduce to recursion. For one thing, there are clever data structures. For many linear-time algorithms, for example, it is crucially important that an algorithm does not manipulate large objects directly; instead it manipulates only pointers to those objects. Such aspects of complexity analysis seem below the abstraction level of recursors.

• *Distributed algorithms* The recursor approach does not seem to extend to distributed algorithms, and the number of useful distributed algorithms is large and growing.

• *Monotonicity limitation* Here is something that the recursor theory should be able to cover but doesn't. The current recursor theory is limited to recursors with semantics given by the least fixed point of a monotone operator. That is a serious limitation.

For a simple example consider Datalog with negation [1]. The operator defined by a Datalog-with-negation program is not monotone but it is inflationary, and semantics is given by the inflationary fixed point [13].

For illustration, here is a Datalog-with-negation program computing the complement C of the transitive closure T of a nonempty binary relation R on a finite domain [1, Example 3.3].

$$
\begin{aligned}
&T(x,y) \leftarrow R(x,y)\\
&T(x,y) \leftarrow R(x,z), T(z,y)\\
&U(x,y) \leftarrow T(x,y)\\
&V(x,y) \leftarrow T(x,y), R(x',z'), T(z',y'), \neg T(x',y')\\
&C(x,y) \leftarrow \neg T(x,y), U(x',y'), \neg V(x',y')
\end{aligned}
$$

Explanation. At every step all rules are fired. By the first two rules, the computation of T proceeds in the usual way. Since the domain is finite, the computation of T completes after some number k of steps. The pairs of T are stored in U with a delay of one step, so the computation of U completes after $k+1$ steps. The computation of V is identical to that of U, except that at the step $k+1$, when U is completed, the last batch of pairs from T is not stored in V. The final rule is idle during the first k steps but on step $k+1$ it stores the complement of T into C.

• *More examples, please.* It would be much useful to have more example of recursors of interest to computer scientists. All current examples of that sort seem to be present already in the 1984 article [16].

Acknowledgments. Many thanks to Andreas Blass for numerous illuminating discussions, and to Serge Grigorieff and Oron Shagrir for useful suggestion.

References

1. Abiteboul, S., Vianu, V.: Datalog extensions for database queries and updates. J. of Computer and System Sciences 43, 62–124 (1991)
2. Biedl, T., Buss, J.F., Demaine, E.D., Demaine, M.L., Hajiaghayi, M., Vinař, T.: Palindrome recognition using a multidemensional tape. Theoretical Computer Science 302, 475–480 (2003)
3. Blass, A., Gurevich, Y.: Abstract state machines capture parallel algorithms. ACM Transactions on Computational Logic 4(4), 578–651 (2003); Correction and extension, same journal 9(3) article 19 (2008)
4. Blass, A., Gurevich, Y.: Algorithms vs. machines. Bull. European Association for Theoretical Computer Science 77, 96–118 (2002)
5. Blass, A., Gurevich, Y.: Algorithms: A quest for absolute definitions. In: Current Trends in Theoretical Computer Science, pp. 195–225. World Scientific (2004); also in Olszewski, A., et al. (eds): Church's Thesis after 70 Years, pp. 24–57. Ontos Verlag (2006)
6. Blass, A., Gurevich, Y.: Ordinary interactive small-step algorithms. ACM Trans. Computational Logic (Part I), 7(2), 363–419 (2006); plus 8(3), articles 15 and 16 (Parts II, III) (2007)
7. Blass, A., Gurevich, Y., Rosenzweig, D., Rossman, B.: Interactive small-step algorithms. Logical Methods in Computer Science 3(4), papers 3 and 4 (Part I and Part II) (2007)
8. Dershowitz, N., Gurevich, Y.: A natural axiomatization of computability and proof of Church's thesis. Bull. of Symbolic Logic 14(3), 299–350 (2008)
9. Gandy, R.: Church's thesis and principles for mechanisms. In: Barwise, J., et al. (eds.) The Kleene Symposium, pp. 123–148. North-Holland (1980)
10. Cooper, S., Löwe, B., Sorbi, A. (eds.): New Computational Paradigms: Changing Conceptions of what is Computable. Springer, Heidelberg (2008)
11. Grädel, E., Nowack, A.: Quantum Computing and Abstract State Machines. In: Börger, E., Gargantini, A., Riccobene, E. (eds.) ASM 2003. LNCS, vol. 2589, pp. 309–323. Springer, Heidelberg (2003)
12. Gurevich, Y.: Sequential Abstract State Machines Capture Sequential Algorithms. ACM Transactions on Computational Logic 1(1), 77–111 (2000)
13. Gurevich, Y., Shelah, S.: Fixed-point extensions of first-order logic. Annals of Pure and Applied Logic 32, 265–280 (1986)
14. Kolmogorov, A.N.: On the concept of algorithm. Uspekhi Mat. Nauk 8(4), 175–176 (1953) (in Russian); English translation in [25]
15. McCarthy, J.: A basis for a mathematical theory of computation. In: Brafford, P., Herschberg, D. (eds.) Computer Programming and Formal Systems, pp. 33–70. North-Holland (1963)
16. Moschovakis, Y.N.: Abstract recursion as a foundation of the theory of algorithms. In: Computation and Proof Theory. Lecture Notes in Mathematics, vol. 1104, pp. 289–364. Springer, Heidelberg (1984)

17. Moschovakis, Y.N.: What is an algorithm? In: Engquist, B., Schmid, W. (eds.) Mathematics Unlimited – 2001 and Beyond, pp. 919–936. Springer, Heidelberg (2001)
18. Moschovakis, Y.N.: Algorithms and implementations. Tarski Lecture 1 (2008), http://www.math.ucla.edu/~ynm/lectures/tlect1.pdf
19. Moschovakis, Y.N., Paschalis, V.: Elementary algorithms and their implementations. In: [10], pp. 87–118
20. Shagrir, O.: Effective computation by humans and machines. Minds and Machines 12, 221–240 (2002)
21. Sieg, W.: Calculations by man & machine: Mathematical presentation. In: Proceedings of the Cracow International Congress of Logic, Methodology and Philosophy of Science, pp. 245–260. Kluwer (2002)
22. Sieg, W.: Church without dogma – Axioms for computability. In: [10], pp. 139–152
23. Sieg, W.: On Computability. In: Irvine, A. (ed.) Handbook of the Philosophy of Mathematics, pp. 535–630. Elsevier (2009)
24. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proceedings of London Mathematical Society, Series 2 42, 230–265 (1936-1937); Correction, same journal 43, 544–546
25. Uspensky, V.A., Semenov, A.L.: Algorithms: Main Ideas and Applications. Kluwer (1993)
26. Valarcher, P.: Habilitation à Diriger des Recherches, Université Paris Est Créteil, LACL (EA 4219), Département d'Informatique, IUT Fontainebleau, France (2010), http://www.paincourt.net/perso/Publi/hdr.pdf

Strong Bridges and Strong Articulation Points of Directed Graphs

Giuseppe F. Italiano

University of Rome 'Tor Vergata'

Abstract. Given a directed graph G, an edge is a strong bridge if its removal increases the number of strongly connected components of G. Similarly, a vertex is a strong articulation point if its removal increases the number of strongly connected components of G. Strong articulation points and strong bridges are related to the notion of 2-vertex and 2-edge connectivity of directed graphs, which surprisingly seems to have been overlooked in the past. In this talk, we survey some very recent work in this area, both from the theoretical and the practical viewpoint.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, p. 43, 2012.

Towards Computational Models of Artificial Cognitive Systems That Can, in Principle, Pass the Turing Test*

Jiří Wiedermann

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

> *"I believe that in about fifty years' time it will be possible, to programme computers, with a storage capacity of about* 10^9*, to make them play the imitation game so well that an average interrogator will not have more than 70 per cent chance of making the right identification after five minutes of questioning. The original question, "Can machines think?" I believe to be too meaningless to deserve discussion. Nevertheless I believe that at the end of the century the use of words and general educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted."*
>
> A.M. Turing: Computing machinery and intelligence. Mind, 59, 433-460, 1950

Abstract. We will give plausible arguments in favor of a claim that we already have sufficient knowledge to understand the working of interesting artificial minds attaining a high-level cognition, consciousness included. Achieving a higher-level AI seems to be not a matter of a fundamental scientific breakthrough but rather a matter of exploiting our best theories of artificial minds and our most advanced data processing technologies. We list the theories we have in mind and illustrate their role and place on the example of a high-level architecture of a conscious cognitive agent with a potential to pass the Turing test.

1 Introduction

We are living in times for which the introductory Turing's notoriously known prediction should have become a reality. But is it really so? Certainly not. This holds despite the fact that the test conditions mentioned in the quotation are quite mild. First, the test admits up to 30% of errors in the identification accuracy which is not far from the random guessing. Second, the conversation time is limited. And third, an "average interrogator" is assumed. Yet there is no computer known able to pass such a test.

It is less known that Turing also made a prediction concerning the "strong version" of the Turing test, allowing unlimited conversation time, a negligible erroneous identification rate and an expert interrogator. In a 1952 BBC broadcast [42], the following

* This research was carried out within the institutional research plan AV0Z10300504 and partially supported by GA ČR grant No. P202/10/1333.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 44–63, 2012.

conversation took place in a discussion between Turing and one of his colleagues (M. H. A. Newman, professor of mathematics at the Manchester University):

Newman: I should like to be there when your match between a man and a machine takes place, and perhaps to try my hand at making up some of the questions. But that will be a long time from now, if the machine is to stand any chance with no questions barred?

Turing: Oh yes, at least 100 years, I should say.

This seems to be a much safer prediction.

However, it appears that asking for passing the strong-version of the Turing test is unnecessarily demanding a task. It would be quite acceptable if the machine betrayed itself by displaying more intelligence than one would expect from a man. Simply stated, the machine could be able to outperform humans in any intellectually challenging task. On the other hand, the machine could also betray itself by being less knowledgeable in the fields concerning human private, social and emotional life. Under such conditions, even for an expert a conversation with such a machine could certainly be more challenging than a chat with a machine of "average intelligence". In a sense, constructing such a machine may even be simpler than to construct machines passing the full Turing test by faithfully mirroring the human mind down to the last detail. The author also claims that we will be able to construct such machines sooner than the general public might believe. It is the goal of this paper to give a "technical" evidence for such a faith.

In the sequel, we will defend the stance that we already have a sufficient knowledge to understand the working of interesting artificial minds with a potential to pass the Turing test. Based on this, it seems that achieving Turing–test capable machines is not a matter of a fundamental scientific breakthrough but rather a matter of exploiting our best theories of artificial minds and our most advanced data processing techniques.

The structure of the paper is as follows.

In Section 2, we describe the status-quo of the field. First, in part 2.1, we illustrate the current state-of-the-art of practice in the field of artificial cognitive systems potentially aspiring to pass the Turing test using the example of the most recent and most visible representative of such systems — viz an IBM Corp.'s computer named Watson. Then, in part 2.2, we focus our attention on the selected examples of the most important recent achievements and trends of the field that will, at least to our mind, shape the field in near future. The most important among these features is a departure from biologism, automatically built internal world models, use of mirror neurons forming the basis for imitation leaning, realization of both phenomenal and functional consciousness utilizing the global workspace theory, exploitation of episodic memories, and real time massive data processing.

In Section 3 we briefly describe the architecture of a cognitive system incorporating the ideas mentioned in the previous section 2.2. This system is based on the author's previous model (cf. [47]) called "HUGO" in [46]. Attention will be paid mainly to aspects related to the framework from Section 2.2. As compared to the previous version of HUGO [47], the present version contains two main innovations. Namely, its mechanisms dealing with the episodic memories and phenomenal consciousness are made explicit.

Conclusions are in Section 4.

2 Status Quo

2.1 Watson the Computer

Recently, we have all witnessed a hallmark of the forthcoming era of machines that could pass the Turing test. In February 2011 news appeared about an IBM Corp.'s "Watson" computer defeating the top two Jeopardy! champions of all time during a practice round, showcasing the current state of artificial general intelligence. According to Wikipedia, Jeopardy! is an American television quiz show featuring trivia in history, literature, the arts, pop culture, science, sports, geography, wordplay, and more. The show has a unique answer-and-question format in which contestants are presented with clues in the form of answers, and must phrase their responses in question form.

The underlying Watson computer had no access to the Internet, but to a RAM memory containing about 200 millions of pages (4TB) of structured and unstructured text written in natural human language. Watson's entire software and data consumed about 16 TB (i.e., 16×10^{12} bytes or about 10^{14} bits) of RAM memory. This is about hundred thousands times more than the original Turing memory estimate. Moreover, this software ran on 90 powerful servers with a total of 2880 processors. Watson could process 500 GB, the equivalent of a million books, per second. Those are the numbers Turing probably could not dream about. The architecture of Watson has been described e.g. in [13].

But could Watson pass the Turing test? Only when restricting the test to the "Jeopardy!" domain. In such a case, the performance of Watson would be so good that "an average observer will not have more than 70 per cent chance of making the right identification after five minutes of playing Jeopardy!", to paraphrase Turing's words. A restricted form of Turing's test, which compares the machine against the abilities of experts in specific fields, is sometimes called Feigenbaum's test.

From a viewpoint of some people working in AI Watson's victory has not been a great feat since apparently neither breakthrough ideas, nor at lest new recent theoretical achievements have been used. Instead, a pragmatic approach based on massive data (pre)processing and natural language processing technologies known for years, undoubtedly pushed to their current limits, has been used.

In an interview about Watson [36], Noam Chomsky, the prominent American intellectual, linguist, philosopher and cognitive scientist, has said: *"Watson understands nothing. It's a bigger steamroller. Actually, I work in AI, and a lot of what is done impresses me, but not these devices to sell computers."* "A bigger steamroller" was apparently a reference to "a small steamroller", the Deep Blue computer of IBM Corp., which in 1997 defeated the world chess-master Garry Kasparov.

John Searle, professor of philosophy at the University of California, Berkeley, a renowned philosopher of the mind, the inventor of the famous Chinese room argument [34], in his newspaper article on Watson [35] essentially agreed with Chomsky: *"Watson did not understand the questions, nor its answers, nor that some of its answers were right and some wrong, nor that it was playing a game, nor that it won — because it doesn't understand anything"*.

Both gentlemen are, of course, right. Watson does not understand anything, Watson does not think. However, both gentlemen failed to appreciate the importance of

this victory in which a computer defeated humans in a game designed to be played between humans communicating in a natural language and following very general and only slightly restricting rules. This is to be compared with the previous success of AI — the "small steamroller" indirectly mentioned by Chomsky. The difference is tremendous: while chess is a play with very formal and rigorous rules, nothing like that holds in Jeopardy!. The jump in the level of informality is remarkable. One can ask how many of such leaps will be necessary in order for computers to pass the Turing test. In order to make an ultimate leap towards such computers, we may also ask how should such computers look like. Do we already have ideas about the principles by which computers, featuring a fully fledged human intelligence, will be based?

2.2 Winds of Change

The previous subsection has illustrated the "practical" status quo of the field of artificial cognitive systems using the most recent and most visible representative of such systems. Clearly, Watson the Computer is not a system reflecting the state-of-the-art in the theory and experimental development of the field of cognitive systems.

The general impression from studying the theoretical background of artificial cognitive systems is that it does not provide an easy survey. In spite of this, it appears that during the past two decades an interesting and promising body of new knowledge has accumulated in the theory which, when properly screened, selected and ordered, has a potential to offer more or less coherent ideas about algorithmic principles on which computational cognitive systems could be based. This knowledge has been scattered in the respective literature, the emerging trends are often not formulated explicitly and important contributions seem to penetrate only slowly into the general awareness of people working in the field.

Here, we will highlight the main reasons for believing that we already have enough knowledge to understand the algorithmic principles behind working of interesting artificial minds attaining a high-level cognition. In what follows we will list and comment the main ideas, trends and important theoretical achievements we have in mind.

2.2.1 Escaping the Turing Test

In theory, instead of considering artificial cognitive systems that can pass the Turing test, we more often consider a more general notion of *humanoid cognitive systems.* Humanoid cognitive systems are cognitive systems endowed with human-like intelligence, not necessarily with the intelligence that would be indistinguishable by Turing test. Namely, Turing test is explicitly, and unnecessarily, anthropomorphic. If our ultimate goal is to create machines that could help people in an intellectual domain, then it does not make sense to insist that the behavior of our machines must closely resemble that of people. Russell and Norvig [33] have invented a nice parable by noticing that *"aeronautical engineering texts do not define the goal of their field as 'making machines that fly so exactly like pigeons that they can fool other pigeons'".*

When speaking about humanoid cognitive systems, nowadays we usually have in mind *humanoid robotic systems.* This is a substantial deviation from Turing's original ideas [41] when he had in mind only "disembodied" computers, with no sensors, communicating with people only via a terminal.

In humanoid robotic systems, the adjective "humanoid" concerns both the form and the contents of such robotic systems. Physically, such systems should take the form of a human body, with as much sensors and actuators, mirroring those of the human body, as possible. As we shall see later, this is of utmost importance since practically all cognitive functions, higher-level function included, are derived from the sensorimotor interaction of a robot with its environment. Should the cognitive functions developing in a robot be of human-like nature, then the sensorimotor interaction of that robot, inclusively its environment, should be of similar nature as in the case of humans. We say that a humanoid robotic cognitive system should be *embodied* in a human-like body, and *situated*, via its sensors and effectors, in a human-like environment (cf. [29], [30] for more details).

As far as the "contents" of a humanoid robotic system, i.e., its control system is concerned, it appears there is no need to mirror the architecture of the human brain, only its functionality — see the next item.

2.2.2 Escaping Biologism

The next idea is escaping from anthropomorphism, or biologism in the design of control part of cognitive systems. It is amazing how many pictures and schemes of the human brain we see during a conference devoted to cognitive system design. Compare that to a similar situation in a conference devoted to the aircraft design, plagued by pictures of birds and their anatomy.

The following observation by Rodolfo Llinás, a prominent American neuroscientist, a founding father of modern brain science, adds a further aspect: *"I must tell you one of the most alarming experiences I've had in pondering brain function... that the octopus is capable of truly extraordinary feats of intelligence...most remarkable is the report that octopi may learn from observing other octopi at work. The alarming fact here is that the organization of the nervous system of this animal is totally different from the organization we have learned is capable of supporting this type of activity in the vertebrate brain... there may well be a large number of possible architectures that could provide the basis of what we consider necessary for cognition and qualia"* [24].

Indeed, there are many possible architectures for cognition, and it only seems natural when thinking about artificial minds to concentrate on solutions permitted, and enabled by our technologies while, of course, being inspired by nature, but not copying slavishly the human brain architecture.

2.2.3 Internal World Models

If a humanoid cognitive system has to communicate "intelligently" with people, it should obviously have information how the people's world looks like, what could be the abilities of people under various circumstance, etc. In short, such a system should possess a kind of internal model of the external world (inclusively that of the self), represented in whatever useful way.

Nowadays, it is generally believed that in order to open the road towards higher brain functions in humanoid cognitive systems we need automatic computational mechanisms that will augment the semantic knowledge acquired in the interaction of the system with its environment. These mechanisms often make use of internal world models. Presently, prevailing trends seem to prefer representations of the internal worlds in

form of neural nets rather than in form of rule-based symbol manipulation systems. For an overview of the recent state-of-the-art and a discussion on internal world models, cf. [19] or [10]. A cognitive system architecture exploiting the idea of a world model can be found, e.g., in [47].

2.2.4 Mirror Neurons

Mirror neurons were discovered during the 1990s (cf. [32]). As V. S. Ramachandran, the prominent neuroscientist, has put it, *"the discovery of mirror neurons in the frontal lobes of monkeys, and their potential relevance to human brain evolution is the single most important "unreported" (or at least, unpublicized) story of the decade. I predict that mirror neurons will do for psychology what DNA did for biology: they will provide a unifying framework and help explain a host of mental abilities that have hitherto remained mysterious and inaccessible to experiments"* [31]. Roughly speaking, the mirror neurons are neurons that fire if their owner performs a certain action as well if their owner observes the same species performing the same action. This can be interpreted as mirror neurons being a mechanism for "mind reading" of other subjects. Other researchers speculated on the existence of similar neurons also in primates and developed far-reaching conjectures on the importance of mirror neurons for understanding the intentions of other people, empathy, imitation learning and even for language readiness (cf. [4], [20], [31]). Surprisingly, until to date mirror neurons do not seem to find their way into generally accepted computational cognitive models. However, in [45] it was shown that mirror neurons can serve as a mechanism synthesizing the multimodal (i.e., motor, perceptional and proprioceptive) information and completing it, if necessary, so that an agent can remain situated even when parts of the multimodal information are missing. Such a mechanism forms a basis on which plausible explanation of the development of a host of mental abilities has been founded. These abilities range from imitation learning, communication via a sign language up to the dawn of thinking and consciousness. The respective results have built a bridge between the theory of embodied cognition and mirror neurons. These results have also justified the above mentioned hopes laid on the discovery of mirror neurons, indeed. The basic model from [45] has later been elaborated in a series of subsequent papers (cf. [46], [47]).

2.2.5 Global Workspace Theory

Global workspace theory (GWT) is a simplistic, very high-level cognitive architecture that has been developed by B. J. Baars by the end of the last century [5], [6] to explain emergence of a conscious process from large sets of unconscious processes in the human brain. Central to the theory is a model of information flow in which multiple, parallel, asynchronous specialist processes (corresponding to unconscious processes) compete and co-operate in an arena for access to a global workspace. A winning process then corresponds to a conscious process which is promoted to the global workspace and is allowed to broadcast information back to the arena. Based on this, the specialist processes invoke another set of unconscious processes and the whole cycle repeats itself. Note that the processes in the global workspace appear in a serial manner, one after the other, while each of them is the integrated product of parallel processing. The GWT can successfully model a number of characteristics of consciousness, such as its role in handling novel situations, its limited capacity, its sequential nature, and its

ability to trigger a vast range of unconscious brain processes. Unfortunately, the GWT neither does explain the mechanism how an originally unconscious process becomes a conscious one nor the mechanism of process competition.

The GWT has been incorporated into a number of computational models (cf. S. Franklin's IDA model [14]). It is perhaps interesting to observe that one "question/answer processing cycle" (cf. [13]) of Watson the Computer works, in fact, according to the GWT.

2.2.6 (Dis)solving the Hard Problem of Consciousness

For the past decade or two, the modern theory of consciousness has been stigmatized by the dichotomy between so-called *functional (or access) consciousness* and the so-called *phenomenal consciousness (or qualia).* These two notions were famously introduced by American philosopher of the mind, Ned Block [7] and subsequently also adopted by other important protagonists (e.g., David Chalmers [9]) in the field. Functional consciousness consists of that information globally available in the cognitive system for the purposes of reasoning, speech and high-level action control, whereas phenomenal consciousness consists of subjective phenomenal experience and feelings. Nowadays, we have relatively good ideas how to implement functional consciousness (cf. [47]). On the other hand, phenomenal consciousness seemed to present a nut hard to crack. The problem of explaining how and why we have qualitative phenomenal experiences (of form "what is it like") presents so-called *hard problem of consciousness* [9]. Some researchers even speculated that non-computational mechanisms for producing the subjective experiences of phenomenal consciousness (in the brain) must be found.

Nevertheless, recently theories pointing to a common evolutionary [17] and sensorimotor basis (cf. [27] and other works by this author) for both phenomenal and functional consciousness have appeared. According to O'Regan [27], in order to have a "raw feel" (qualia) it will suffice for a robot already possessing functional consciousness to engage in an embodied (sensorimotor) interaction with its environment. More specifically, qualia can be seen as an engagement in exercising a "fixed sensorimotor skill" accompanied by (functional) conscious attendance to that engagement and the skill's quality. The respective real-world interaction has to possess the properties of richness, bodiliness, insubordinateness and grabbiness. *Richness* characterizes abundance in details. *Bodiliness* or corporality requires that voluntary motions of a body systematically affect sensory inputs. *Insubordinateness* means that the world has its own dynamic that we can affect only partially (if at all) causing that bodiliness is never complete. And finally, *grabbiness* means that the perceptual stimuli have the alerting capacity — they can peremptorily interfere with cognitive processing (e.g., they can cause in interrupt).

2.2.7 Episodic Memory

Episodic memory is what people "remember", i.e., the contextualized information about autobiographical events (times, places, associated emotions), and other contextual knowledge that can be explicitly stated. It is obvious that such memory is important for an agent to know about its past. Therefore, as noted in [26], it is surprising that the vast majority of integrated intelligent systems ignore episodic memory, which often dooms them to what can be achieved by people with amnesia, which is demonstrably limited. Nowadays we frequently witness "add-ons" to the existing models of cognitive

systems architecture to account for episodic memory (cf. [26]). In [26] it is argued that episodic memory systems can support a vast number of cognitive capabilities which are mostly based on inspecting memories from the past that are "similar" to the present situation. Among these capabilities there is noticing novel situations, detecting repetitions, virtual sensing (reminded by some recall), future action modelling, planing ahead (cf. [48]), environment modelling, predicting success/failure, managing long term goals, etc. Incorporating episodic memories and their efficient management and especially their retrieval is a non-trivial matter and it is here where current massive data processing technologies can find their good use.

2.2.8 Real Time Massive Data Processing

In a sense, Watson the Computer can be seen as a crippled cognitive system specialized in doing efficient contextual retrieval invoked by clues over its preprocessed episodic memory. Its success was possible thanks to technological progress enabling maintenance of supercritical volumes of data and their searching and retrieval by supercritical speed. Could this be the case that we are witnessing the birth of a new paradigm? This paradigm states that *intelligence is not only a matter of suitable algorithms, but also, and mainly so, of the ability to accumulate (e.g., via learning and episodic memories storing), organize, and exploiting large data volumes representing knowledge, at a speed matching the timescale of the environmental requirements.* In the case of a robot, its sensorimotor interactions must also possess this quality, i.e., they must involve real time processing. Watson the Computer seems to be the first case where the real time aspect has boldly entered the game, enforcing a massively parallel solution which, eventually, has become the main factor in Watson's victory.

2.2.9 Comprehensive and Up-To-Date Models of Cognitive Systems

In recent years we have seen a number of proposals of cognitive systems architectures, cf. [2], [23], [37], or [43], to name a few of them. All these proposals have aimed if not towards implementation, then at least towards modelling and explanation of selected subsets of higher mental functions. Some of these systems have a long developmental history reaching to the 1980s through which they have evolved hand in hand with the theory towards higher level cognitive functions. Inevitably, the design of these systems reflects the spirit of the time of their creation. Most of them were meant as proposals of experimental architectures with the goal to verify their viability in solving basic learning cognitive tasks, trying to reach up to the higher cognitive functions. Some of these models have been quite complex, mirroring the known brain architecture, often with "modules" responsible for realization of certain tasks (like "anticipation", "motivation", "planning", "action selection", etc.), and with only vague ideas about the corresponding algorithmic mechanisms. A few of the models seem to be comprehensive enough, i.e., covering the whole range of cognitive abilities, from sensorimotor coupling, imitation learning, language evolution and acquisition, thinking, up to and inclusive functional consciousness. However, none of them seems to be on a par with the previous list of recent trends and achievements.

In the next section, we present a model that appears to be comprehensive in the previous sense and whose design reflects the previously mentioned "winds of change".

3 A Non-biological Model of a Conscious Cognitive System

In this section, we introduce a high-level model of a cognitive system that aspires to fulfill the requirements mentioned in the previous section.

The model is based on the author's model HUGO which, in the course of its existence, has passed through a few evolutionary phases [45], [46]. Its current state has been described in [47]. We present its abridged description stressing its up-to-date aspects.

Presenting the model, we make use of an approach used in software engineering in the design of large software systems. We start by sketching its architecture and giving an informal specification of its basic modules. This means that we define the type of data and data flow among the individual modules as well as the task of these modules in processing the data. Then, we give plausible arguments supporting the realization of processes mimicking higher cognitive tasks such as imitation learning and development of communication, language, thinking and consciousness.

The design of HUGO has not been influenced by the architecture of human brain, or that of any other animal. In that sense it is a non-biological model (cf. 2.2.2). However, in some aspects its operation bears quite a similarity to the operation of human brain. This indicates that the underlying principles are perhaps of a general nature which all models of cognition must capture.

3.1 HUGO

The internal structure of our model is depicted in Fig. 1. It consists of four main parts: there are sensorimotor units, the sensorimotor world model represented by a mirror net, the control unit, and the body. Arrows depict the data flow between these parts. All data transferred along the arrows are of digital nature.

Next, we specify the actions performed by the model's individual parts.

The *sensorimotor units* receive so-called *motor instructions* from the control unit. These are not only instructions for locomotive organs of the agent, but also instructions for pointing the sensors in a certain direction, for changing their settings, etc. At the same time, these instructions flow into the mirror net. The sensorimotor units deliver two kinds of data back to the mirror net.

The first kind of data is *exteroceptory data* that deliver information from the sensory units scanning the agent's environment. In this case, the sensory units act as a transformer of registered physical inputs (electromagnetic waves, sounds, pressure, etc.) into the digital form. In general, this transformation cannot be described mathematically since it depends on the physical/technical characteristics of the sensory units. The second kind of data is *proprioceptory data* delivering information from the internal sensors placed within the sensorimotor units or within the agent's body. For instance, this can be information about the current settings of the units or current conditions of the unit.

The next part of the model is the *mirror net.* It is a network of artificial mirror neurons which act analogously to (our ideas on) real mirror neurons. In each unit of this net (which might consists of several neurons), the exteroceptory and proprioceptory data from sensorimotor units (denoted as 'perception' in Fig. 1) meet with the motor instructions and (possibly) the *annotations* from the control unit. The annotations are

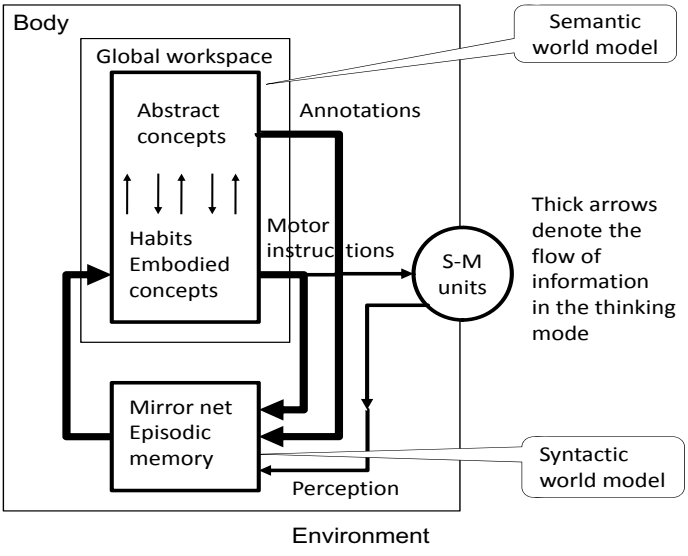


Fig. 1. The structure of a cognitive agent

the set of abstract concepts which are active at that time in the control unit. Then, the conjunction of motor instructions, annotations and of perceptual information — called 'units of S-M information' in Fig. 1 — is computed. This joint information is called *multimodal information.* The task of the mirror net is fourfold:

Learning: the net learns frequently occurring multimodal information and stores its representation;

Identification: the net finds multimodal information already stored in the net which is "most similar" to the incoming information;

Associative retrieval: given only partial multimodal information in which the inputs from some sensorimotor units are missing, or only an annotation of such information, the net finds the entire multimodal information of which the partial information or annotation is given.

Episodic memory formation: the incoming multimodal information from sensors is annotated by the contents of consciousness at that time, and stored.

Note that in our model the contents of episodic memories is the entire multimodal unit annotated at the occasion of paying conscious attention to the respective perception. This "conscious attention" can be elements of phenomenal consciousness, or, in more advanced cases, words in a natural agent's language. Annotations enable recall of the episodic memories that are most similar to the given annotation. The recalled memories are then further processed as specific multimodal units.

In order to work in this way, we must take measures that there is only a finite (albeit possibly a large) amount of "important" multimodal information stored in the mirror net. This can be achieved, e.g., by parameterizing motor instructions by a finite set of values and preprocessing the perceptory data by extracting the key features from it. For

such a purpose fuzzy techniques leading to a rough classification of multimodal data into clusters of similar information can be used. Even when such measures are taken in the mostly developed agents there can still be a huge amount of data stored in the mirror net and the episodic memory. It is here where the advanced data processing techniques mentioned in 2.2.8 could find their use.

The next requirement concerns the components of the multimodal information. In order that the associative recall can work well, the entire multimodal information must be uniquely determined by any of its significant components. For reasons that will be explained in the next section — namely in order the thinking mechanism to work — we assume that if there is a motor component in multimodal information, then this component alone determines the rest of multimodal information.

Each part of the mirror net specializes in learning and recognizing specific multimodal information corresponding to one "sensory-behavioral unit". Learning is done perpetually when complete multimodal information appears at the input to the mirror net. Such circumstance is called *standard learning mode.* Learning proceeds by Hebbian principles, i.e., by strengthening the weights of neurons representing the respective multimodal information each time when it is recognized.

Thus, in any case, irrespectively whether all parts or only a (significant) part of the multimodal information enters the net, the net outputs complete multimodal information which proceeds into the control unit. In the context of the control unit, the representations of multimodal information are called the *concepts.* Each concept can be represented by a tuple of values of the attributes which characterize the given concept.

The task of the control unit is, given the current multimodal information represented by the active concepts plus the incoming stream from the mirror net, to produce a new set of active concepts. The motor part of multimodal information and/or the annotations corresponding to the newly activated concepts is sent both to the sensorimotor units and to the mirror net. Clearly, the control unit determines the next action of an agent.

Within the control unit there are so-called *embodied concepts* corresponding to each occurrence of multimodal information in the mirror net. Moreover, new, so-called *abstract concepts* are formed from the existing (mainly embodied) concepts within the control unit. Associations of various strengths connect the concepts within it. The concepts and the associations among them are all stored in the control unit and form the agent's memory. The rules of forming new concepts and strengthening the associations among them are based on the following principles; the first three of them have been already identified by the 18th century Scottish philosopher D. Hume [21]:

Contiguity in space: two concepts get associated (or the respective association gets strengthened) if they frequently occur simultaneously; also, a new concept corresponding to the union of the two concepts gets formed;

Contiguity in time: two concepts get associated (or the respective association gets strengthened) if they frequently occur one after the other;

Similarity: a concept gets associated with another concept if the former is similar to the latter and vice versa; the notion of similarity must be appropriately defined (e.g., by requiring a sufficient overlap in multimodal information);

Abstraction: the common part of two "sufficiently" similar concepts forms an abstraction of the two; the respective "abstract" concept is added to the concepts represented in the control unit.

The control unit should work according to the following rules. At each time, some concepts in it should be in active state. These concepts represent the current *"mental state"* of the agent. When new multimodal information enters the control unit it activates a new set of concepts. Based on the current mental state and the set of newly activated concepts, a new set of concepts is activated. This set represents the new mental state of the agent and determines the next motor action of the unit.

Note that the new mental state is computed from an old one and from the new input. This mechanism is greatly reminiscent of control mechanism in the finite automata. The idea is that the new mental state should be computable via associations stored among the concepts. In detail, the currently and newly activated concepts jointly excite, via the associations, a set of passive concepts. This excitation strengthens all the respective associations by a little amount. At the same time, small amount weakens the remaining associations. This models the process of forgetting. From among the set of all the excited concepts, the set of the most excited concepts gets activated and the previously active concepts are deactivated. The set of currently active concepts is also strengthened. This set then represents the current mental state. The set of currently active concepts can be seen as the *short-term (operational) memory* of the agent. The set of all the concepts with all settings of associations and weights can be seen as a *long-term memory* of the agent. Obviously, the control unit can also be implemented by an artificial neural net. Since in any practical realization the net has to process a huge, "supercritical" number of concepts in real time, use of parallelism in this case seems to be unavoidable (cf. 2.2.8).

Based on the above mentioned principles the control unit is capable of solving simple cognitive tasks: learning *simultaneous occurrence* of concepts (by contiguity in space), their sequence, so-called *simple conditioning* (by contiguity in time), *similarity based behavior* and computing their *abstractions.* In fact, these are the unit's basic operations. The mechanism is also capable to realize *Pavlovian conditioning* (cf. [43], p. 217), in which the control unit can be conditioned to produce a response to an apparently unrelated stimulus.

If one wants to go farther in the realization of the cognitive tasks, one should consider special concepts called *affects.* The affects come in two forms: positive and negative ones. The basic affects are activated directly from the sensors. The ones corresponding to the positive feelings are positive whereas the ones corresponding to the negative feelings are negative. The excitatory (inhibitory) associations arise among positive (negative) affects and concepts. The role of the affects is to modulate the excitation mechanism. With the help of affects, one can simulate the reinforcement learning (also called operant conditioning) and the delayed reinforcement learning. Pavlovian conditioning, reinforcement learning and delayed reinforcement learning seems to be a minimal test, which a cognitive system aspiring to produce a non-trivial behavior should pass.

Affects by themselves do not correspond to what O'Regan [27] calls *"raw feelings",* but they create an important ingredient of phenomenal consciousness under fulfillment of other requirements described in 2.2.6.

In a stimulating environment during an agent's interaction with its environment concepts within the control unit start to self–organize, via property of similarity, into *clusters* whose centers are formed by abstract concepts. Moreover, by properties of time contiguity, chains of concepts, called *habits*, linked by associations start to emerge. The habits correspond to often performed activities. The behavior of agents governed by habits starts to prevail. In most cases such a behavior unfolds effortlessly. Only at the "crossings" of some habits an additional multimodal information from the mirror net (in an on–line or off–line mode—see the next section) is required directing the subsequent behavior. For more details concerning the work and cognitive abilities of the control unit, see the author's earlier paper [44] (and the references mentioned therein) where the control unit under the name "cogitoid" has been described. The operation of a cogitoid can be seen as a specific implementation of the global workspace theory (cf. 2.2.5).

The last component of our model is its body. Its purpose is to support the agent's sensorimotor units and to enclose all its parts into one protective envelope.

Now let us return to the question of internal models (cf. 2.2.3). Obviously, the mirror net can be seen as a specific kind of a static world model. In this model, the world is represented in the way as it is cognised by an agent's sensory and motor actions, i.e., by an agent's interaction with its environment. It can be termed as a *sensorimotor* model describing the "syntax" of the world. In the mirror net, the combinations of the exteroceptory and proprioceptory inputs jointly with motor actions fitting together, which "make sense" for the agent, are stored. Note that since the proprioceptory information is always a part of multimodal information, also elements of an agent's own model are in fact available in the mirror net.

On the other hand, the control unit is a specific model of the world capturing the "semantics" of the world. In this model, the relations among concepts are stored which, obviously, correspond to real relations among real objects and phenomena observed or generated by the agent during its existence. Similar relations are also maintained among the representations of these objects and phenomena. All this information represents a kind of a dynamic internal world model. One can also see this model as a depository of the "patterns of behavior which make sense in a given situation."

In the next section, we describe how the interaction of both world models leads to a more complex behavior.

3.2 Towards Higher Level Cognitive Functions

First we describe the mechanism of imitation learning which is a starting point for higher mental abilities, cf. [4], [20]. Imagine the following situation: agent A observes agent B performing a certain well distinguishable task. If A has in its repository of behavioral units multimodal information, which matches well the situation mediated by its sensors (which, by itself, is a difficult robotic task), then A's mirror net will identify the entire corresponding multimodal information (by virtue of associativity). At the same time, it will complement it by the flag saying *"this is not my own experience"* (because the proprioceptive part of information must have been filled in when completing the entire multimodal information) and deliver it to the central unit where it will be processed adequately. This can be seen as a germ of the self concept.

At that moment, A has information to its disposal what B is about to do, and hence, it can "forecast" the future actions of B. "Forecasting" is done by following the associations in the control unit starting in the current mental state. Agent A can even reconstruct the "feelings" of B (via affects) since they are parts of the retrieved multimodal information. This might be called *empathy* in our model. Moreover, if we endow our agent with the ability to memorize short recent sequences of its mental states, then A can repeat the observed actions of B. This, of course, is called *imitation.*

The same mechanism helps to form a more detailed *model of self.* Namely, observing the activities of a similar agent from a distance helps the observer to "fill in" the gaps in its own dynamic internal world model (i.e., in the control unit), since from the beginning an observer only knows "what it feels like" if it perceives its own part of the body while doing the actions at hand. At this stage, we are close to *primitive communication* done with the help of *gestures.* Indicating some action via a characteristic gesture, an agent "broadcasts" visual information that is completed by the observer's associative memory mechanism to the complete multimodal information. That is, with the help of a single gesture complex information can be mediated. A gesture acts like an element of a higher-level (proto)language. By the way, here *computational emotions* can enter the game as a component of the communication. Their purpose is to modulate the agent's behavior. Of course, for such a purpose the agents must be appropriately equipped (e.g., by specific mimics, possibility of color changes, etc.). Once we have articulating agents, it is possible to complement and subsequently even substitute gestures by *articulated sounds.* It is the birth of a spoken language. At about this time the process of stratification of abstract concepts from the embodied ones begins thanks to the abstraction potential of the control unit (cf. the rules of forming new concepts in the control unit). Namely, the agents "understand" their gestures (language), defined in terms of abstract concepts, via empathy in terms of their embodiment or grounding, in the same sensorimotorics and proprioception (i.e., in the same embodied concepts) [12], [15], and in a more involved case, in the same patterns of behavior (habits). Without having a body an agent could not understand its communication [30].

Returning to the previously mentioned process of concept stratification, the respective two classes of concepts can be seen as concepts on a symbolic and sub-symbolic level, respectively. This view offers an answer to an often mentioned problem, namely whether the mind works with the former or the latter class of concepts. Our model works with both classes of concepts by continuously passing from one class to the other class. The sub-symbolic level is necessary for understanding certain abstract concepts, especially those related to perception or to sensorimotor activities. The language is in fact a superstructure over the embodied concepts. There is one more aspect to be mentioned here: the abstract concepts at the symbolic level need not be grounded directly in the embodied concepts, at a sub-symbolic level. Rather, they can be "*tethered*" (this term has been coined in [38]) to other abstract concepts to which they are related by associations arising in the control unit. The network of certain abstract concepts can thus represent a "theory" within which an agent can "think" (see in the sequel) or act. In the case of artificial agents, such a net can be set-up in the control unit prior to an agent's first activation. That is, the respective behavior need not be learned from scratch by the agent, it can be, so to speak, built-in, or "innate". This mechanism would simulate

the result of a development that in living creatures has been reached by evolution. For similar ideas, cf. [38].

The transition from gestures or body language to articulation does not only mean that gestures get associated with the respective sounds but, above all, with the movements of speaking organs. In the further development this facilitates a voiceless "speaking to oneself" and later on, the transition towards thinking (see in the sequel).

The structure and functionality of the control unit and the mirror net and their cooperation realize, in fact, the solution to the symbol grounding problem in a similar spirit as described by Steels et al. in [40].

Having communication ability, an agent is close to thinking. In our model, *thinking means communication with oneself*, similarly as in cf. [11]. By communicating with oneself, an agent triggers the mechanism of discriminating between the external stimuli (I listen to what I am talking) and the internal ones. This mechanism may be termed as *self-awareness* in our model. By a small modification (from the viewpoint of the agent's designer), one can achieve that the still self-communication can be arranged without the involvement of speaking organs at all. In this case, the respective instructions will not reach these organs; the instructions will merely proceed to the mirror net (see Fig. 1). (Note that some people still move their speaking organs while thinking intensively.) Here they will invoke the same multimodal information as in the case when an agent directly hears spoken language or perceives its gestures via proprioception (here we make use of our assumption that a motor part of multimodal information is sufficient to determine its rest). Obviously, while thinking an agent "switches off" any interaction with the external world (i.e., both perception and motor actions). Thus, in Fig. 1 the parts of the schema connected by thick arrows depict an agent in a *"thinking mode";* this is captured by the cycle from the control unit to the mirror net and back to the control unit. In such a case, seen from the viewpoint of its internal mechanisms, an agent operates as in the case of standard learning mode, i.e., when it receives the "real" perceptory information and executes all motor instructions. In the thinking mode, the same processes go on, but this time they are based on the virtual, rather than real, information mediated by the mirror net. One can say that in the thinking mode an agent works "off-line", while in the standard mode it works "on-line". Note that once an agent has the power of "shutting itself off" from the external world in the thinking mode, then this agent in fact distinguishes between a thought and reality, and this is considered to be the hallmark of the consciousness [39].

In our model, we informally define functional consciousness much in the spirit of Minsky's idea [25] that *"consciousness is a big suitcase",* carrying many different mental abilities. A prologue to functional consciousness is communication and thinking. The following "definition" of functional consciousness assumes that the agents are able to communicate in a higher-level language. A *higher-level language* is an "abstract" language in which a relatively complex action (corresponding to a sequence of mental states) or an abstract concept is substituted by a word expression or a gesture. A language level is the higher the "richer" the language, i.e., the greater and more abstract is a set of things about which one can communicate. The agents can be thought of as being functionally conscious, as long as their language ability has reached such a level that

Cognitive functionality
Innate reflexes, affects, and grounding of elementary actions in sensory-motor coordination
Distinction between “self” and “others”, formation of the “self” concept, imitation of elementary actions
The birth of phenomenal consciousness
Imitation of sequences of actions via rehearsal
The birth of communication via gestures, body language, “mind reading” (i.e., intention discovering by observation), and empathy
Adding of vocalization to gestures and its association with the motorics of gestures and later with the motorics of the speech organs
Development of the episodic memories
Keeping of evolutionary equilibrium between the increase of a number of word stimuli and the brain size
Direct concept activation via listening to the corresponding words, development of abstract concepts
The dawn of thinking: first phase via talking aloud to oneself; subsequently only speaking organ movements prevail; later a gradual decay of these movements with the instructions to speaking organs getting directly associated with the semantics of words
Subjective experience of thinking as a proprioception of abstract concept activation and the self concept activation
Thinking as virtually situated damped sensory-motor actions
A fully developed functional and phenomenal consciousness

Fig. 2. The evolution of cognitive abilities

they are able to fable on a given theme (note the similarity with [8]). More precisely, the functionally conscious agents are able:

- to speak about, think of, and explain, from the first and the third person viewpoint, their past, present and future experience, feelings, intentions and observed objects, actions, and phenomena;
- to imitate the observed (or more generally: perceived, by whatever senses) activities of other agents, to describe them verbally and, vice versa, to perform activities given their description in a higher level language;
- extend their higher level language by new words or to learn a new langauge.

At this developmental level of functional consciousness we may also expect, according to O’Regan’s theory [27], that the agent will also be equipped by phenomenal consciousness. A prerequisite for this to happen is, of course, that the agent’s real-world interaction will possess the quality of richness, bodiliness, insubordinateness and grabbiness mentioned in 2.2.6.

Note that such a state of matters cannot be achieved without agents having an internal world model at their disposal along with the knowledge of the world’s functioning,

and that of their own functioning within this world; to that end the agents must be constructed so that they can learn. A prerequisite for consciousness to emerge is a "social interaction" among agents in a higher-level language with the same or similar semantics. Obviously, consciousness is not a property, which an entity does possess, or not. Rather, it is a continuous quality which ranges from rudimentary forms towards higher ones.

An agent can possess this quality in a various degree. For instance, from the previous requirements it follows that a conscious agent should recognize itself in a mirror. A "more conscious" agent should be able to tell apart lies from the truth. For instance, it can understand fairy tales and should "know" that these are fanciful stories. An even more conscious agent should be able to lie (not merely with the help of, e.g., mimic coloring, but in a higher language) and should still be aware of its lying.

And how do we know that a conscious agent "understands" its own actions and its world? Well — we simply ask it: the ability to answer such questions is the very idea of our "definition" of functional consciousness.

The above described notion of functional consciousness can be seen as a test to be applied by an entity to another entity in order to determine whether the other entity is functionally conscious according to that definition.

As a final remark, it is interesting to observe how our model deals with so-called *symbol detachment problem* investigated in [28]. This is a problem of how and why do the situated agents develop representations that are *detached* from their current sensorimotor interaction, but nevertheless preserve grounding and aboutness. Our model of an evolutionary path leading from the imitation learning capabilities through thinking up to the birth of consciousness offers a plausible explanation of the mechanisms and representations underlying the symbol detachment problem (cf. the emergence of the abstract concepts from the embodied concepts in the control unit) and stresses their role for the development of complex cognitive capabilities.

Figure 2 summarizes the evolution of higher cognitive abilities within an intelligent agent according to our model and the respective knowledge acquisition and learning processes. This evolution can be seen from two different viewpoints. First, it can be seen as a process of a development of an agent (e.g., of a human child) already possessing the respective mechanisms, in the course of agent's growing up and its education. Second, it can be seen as a stepwise 'upgrading' process over an evolutionary sequence of agents, each one of them possessing more advanced mechanisms and embodiments for managing more demanding cognitive skills than those obeyed by its predecessors.

Note the prominent role of mirror neurons in the above schema — starting with the second item at the latest, it is difficult to see how to cope with the corresponding cognitive tasks without having the mirror neurons and, indeed, the internal world model incorporated in the agent's model.

4 Conclusions

Undoubtedly, at present we can observe a change of attitude of the general publics towards the thinking machines. Turing might be quite happy that this part of his prediction (cf. the introductory quotation) has been fulfilled — people consider it quite acceptable

that machines can think. The question of the day is, how exactly, and when will the machines think.

We only have preliminary, but quite plausible ideas about the first part of the last question. It has been the goal of this article to throw some light on this matter.

And when will thinking machines appear? By extrapolating an exponential growth of technology over several decades, futurist Raymond Kurzweil predicted in 2005 that Turing test-capable computers would be manufactured in 2029 [22], and in 2009 he even made a $20,000 public bet that the test will be passed by then, indeed [3].

References

1. Aleksander, I., Dummall, B.: Axioms and Tests for the Presence of Minimal Consciousness in Agents. Journal of Consciousness Studies 10(4-5) (2003)
2. Anderson, M., Bothell, M., Byrne, D., Douglass, S., Lebiere, C., Qin, Y.: An integrated theory of the mind. Psychological Review 111(4), 1036–1060 (2004)
3. The Arena of Accountable Predictions: A Long Bet. By 2029 no computer – or machine intelligence – will have passed the Turing Test (2009), http://longbets.org/1/#adjudication_terms
4. Arbib, M.A.: The mirror system hypothesis: how did protolanguage evolve? In: Tallerman, M. (ed.) Language Origins: Perspectives on Evolution. Oxford University Press (2005)
5. Baars, B.J.: A cognitive theory of consciousness. Cambridge University Press, Cambridge (1988)
6. Baars, B.J.: In the theater of consciousness: The workspace of the mind. Oxford University Press, Oxford (1997)
7. Block, N.: On a Confusion About a Function of Consciousness. Brain and Behavioral Sciences 18, 227–247 (1995)
8. Blum, M., Williams, R., Juba, B., Humphrey, M.: Toward a high-level definition of consciousness. In: Invited Talk Presented at the Annual IEEE Computational Complexity Conference, San Jose, CA (2005)
9. Chalmers, D.: Facing Up To the Problem of Consciousness. J. of Consciousness Studies 2(3), 200–219 (1995)
10. Cruse, H.: The evolution of cognition–a hypothesis. Cognitive Science 27(1) (2003)
11. Dennett, D.: Consciousness Explained. The Penguin Press (1991)
12. Feldman, J.: From Molecule to Metaphor. MIT Press, Cambridge (2006)
13. Ferrucci, D., et al.: Building Watson: An Overview DeepQA Project. AI Magazine, 200–214 (Fall 2010)
14. Franklin, S.: IDA: A conscious artifact? Journal of Consciousness Studies 10(4-5), 47–66 (2003)
15. Harnad, S.: The symbol grounding problem. Physica D (42), 335–346 (1990)
16. Harnad, S.: Other bodies, other minds: a machine incarnation of an old philosophical problem. Minds and Machines (1), 43–54 (1991)
17. Harvey, I.: Evolving Robot Consciousness: The Easy Problems and the Rest. In: Fetzer, J.H. (ed.) Evolving Consciousness. Advances in Consciousness Research Series, pp. 205–219. John Benjamins, Amsterdam (2002)
18. Holland, O. (ed.): Journal of Consciousness Studies. Special Issue: Machine Consciousenss, vol. 10(4-5) (2003)
19. Holland, O., Goodman, R.: Robots with internal models: a route to machine consciousness? Journal of Consciousness Studies 10(4-5) (2003)

20. Hurford, J.R.: Language beyond our grasp: what mirror neurons can, and cannot, do for language evolution. In: Kimbrough, O., Griebel, U., Plunkett, K. (eds.) The Evolution of Communication systems: A Comparative Approach. The Viennna Series in Theoretical Biology. MIT Press, Cambridge (2002)
21. Hume, D.: Enquiry concerning human understanding. In: Selby-Bigge, L.A. (ed.) Enquiries Concerning Human Understanding and Concerning the Principles of Morals, 3rd edn. Clarendon Press, Oxford (2003); revised by Nidditch, P.H.
22. Kurzweil, R.: The Singularity is Near, p. 652. Viking Books (2005)
23. Langley, P.: An adaptive architecture for physical agents. In: Proceedings of the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, pp. 18–25. IEEE Computer Society Press, Compiegne (2005)
24. Llinàs, R.: I of the Vortex: From Neurons to Self. MIT Press (2001)
25. Minsky, M.: Consciousness is a big suitcase. EDGE (1998), http://www.edge.org/3rd_culture/minsky/minsky_p2.html
26. Nuxoll, A.M., Laird, J.E.: Extending Cognitive Architecture with Episodic Memory. In: Proceedings of the Twenty-Second Conference on Artificial Intelligence. AAAI Press, Vancouver (2007)
27. O'Regan, J.K.: How to Build Consciousness into a Robot: The Sensorimotor Approach. In: Lungarella, M., Iida, F., Bongard, J.C., Pfeifer, R. (eds.) 50 Years of Aritficial Intelligence. LNCS (LNAI), vol. 4850, pp. 332–346. Springer, Heidelberg (2007)
28. Pezzulo, G., Castelfranchi, C.: The symbol detachment problem. Cogn. Processes 8, 115–131 (2007)
29. Pfeifer, R., Scheier, C.: Understanding Intelligence. The MIT Press, Cambridge (1999)
30. Pfeifer, R., Bongard, J.: How the body shapes the way we think: a new view of intelligence. The MIT Press (2006)
31. Ramachandran, V.S.: Mirror neurons and imitation as the driving force behind 'the great leap forward' in human evolution. EDGE: The Third Culture (2000), http://www.edge.org/3rd_culture/ramachandran/ramachandran_p1.html
32. Rizzolatti, G., Fadiga, L., Gallese, V., Fogassi, I.: Premotor cortex and the recognition of motor actions. Cognitive Brain Research 3, 131–141 (1996)
33. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice Hall, Upper Saddle River (2003)
34. Searle, J.R.: Minds, brains, and programs. Behavioral and Brain Sciences 3(3), 169–225 (1980)
35. Searle, J.S.: Opinion: Watson Doesn't Know It Won on 'Jeopardy!'. The Wall Street Journal (2011), http://WSJ.com
36. Schmitt, G.: Conversation from January/February 2011, between myself and Noam Chomsky (2011), http://www.framingbusiness.net/archives/1287
37. Shanahan, M.P.: Consciousness, emotion, and imagination: a brain-inspired architecture for cognitive robotics. In: Proceedings AISB 2005 Symposium on Next Generation Approaches to Machine Consciousness, pp. 26–35 (2005)
38. Sloman, A.: Why symbol-grounding is both impossible and unnecessary, and why theory-tethering is more powerful anyway (2007), http://www.cs.bham.ac.uk/research/projects/cogaff/talks/models.pdf
39. Smee, A.: Principles of the human mind deduced from physical laws, N.Y (1849) (1853)
40. Steels, L., Loetzsch, M., Spranger, M.S.: Semiotic dynamics solves the symbol grounding problem. Nature Precedings (2007), http://hdl.nature.com/10101/npre.2007.1234.1

41. Turing, A.: Computing Machinery and Intelligence. Mind 59(236), 433–460 (1950)
42. Turing, A., et al.: Can Automatic Calculating Machines Be Said to Think? In: Shieber, S. (ed.) From a 1952 BBC Broadcast, Reprinted in The Turing Test: Verbal Behavior and the Hallmark of Intelligence. The MIT Press, Cambridge (1952)
43. Valiant, L.G.: Circuits of the mind. Oxford University Press, New York (1994)
44. Wiedermann, J.: Towards Algorithmic Explanation of Mind Evolution and Functioning. In: Brim, L., Gruska, J., Zlatuška, J. (eds.) MFCS 1998. LNCS, vol. 1450, pp. 152–166. Springer, Heidelberg (1998)
45. Wiedermann, J.: Mirror neurons, embodied cognitive agents and imitation learning. Computing and Informatics 22(6), 545–559 (2003)
46. Wiedermann, J.: HUGO: A Cognitive Architecture with an Incorporated World Model. In: Proc. of the European Conference on Complex Systems ECCS 2006. Said Business School, Oxford University (2006)
47. Wiedermann, J.: A high level model of a conscious embodied agent. In: Proc. of the 8th IEEE International Conference on Cognitive Informatics, pp. 448–456 (2009); expanded version appeared in International Journal of Software Science and Computational Intelligence (IJSSCI) 2(3), 62–78 (2010)
48. Zimmer, C.: The Brain: Memories Are Crucial for Looking Into the Future. DISCOVER Magazine (2011)

A Fully Generic Approach for Realizing the Adaptive Web

Paul De Bra and David Smits

Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands
debra@win.tue.nl, d.smits@tue.nl

Abstract. It is time for Adaptive Web (server) extensions to grow up and become generic. The GRAPPLE[1] (EU FP7) project aimed at integrating Learning Management Systems (LMS) with Adaptive Learning Environments (ALE) in order to support life-long learning. But instead of developing a dedicated Web-based ALE we developed an architecture containing a fully generic adaptive Web server, a distributed User Modeling Framework and a generic browser-based authoring environment for Domain Models and Conceptual Adaptation Models. The GRAPPLE architecture can be used for creating and serving any type of adaptive Web-based application. It supports content-, link- and presentation (layout) adaptation based (in any programmable way) on any desired user model information. In this paper we concentrate on GALE [21], the adaptation engine we renamed to the "Generic Adaptation Language and Engine". We describe the key elements that make GALE into a truly generic and highly extensible Web-based adaptive delivery environment.

Keywords: adaptation engine, adaptation rules, generic architecture.

1 Introduction

Vannevar Bush is often said to be the inventor of *hypertext* because of his article "As We May Think" [11]. (The term "hypertext" was not used by Bush and introduced much later by Ted Nelson [19].) Bush actually did more in that article, by envisioning that the user would build "*trails of his interest through the maze of materials available to him*". This type of user was called the "trailblazer". It is a first sign of personalization, aimed at facilitating *revisiting* information (either by the same user or by someone else). Another form of personalization was the addition of annotations: "*he inserts a page of longhand analysis of his own*". Adaptive hypermedia research, first summarized by Brusilovsky in 1996 [7] and updated in 2001 [8] aims at automating this trailblazing and annotating through *link adaptation* and *content adaptation*. Knutov et al [18] describe (in 2009) many new adaptation techniques developed to date and provide a list of challenges for creating a new *generic* adaptive hypermedia system, capable

[1] GRAPPLE stands for Generic Responsive Adaptive Personalized Learning Environment.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 64–76, 2012.

of dealing with *ontologies*, *open corpus adaptation*, *group adaptation*, *information retrieval* and *data mining*, *higher order adaptation*, *context awareness* and *multimedia adaptation*.

GALE [21] tackles some of Knutov's challenges directly through implemented functionality (that we will describe) and makes it possible to tackle more challenges by means of a highly extensible modular architecture. GALE builds on the experience we gained in the past when developing and using AHA! [3, 4, 6], the first general-purpose open source adaptive Web-server extension, used in different institutes all over the world.

This paper is organized as follows: Section 2 presents some background on adaptive Web-based systems, and positions GALE as a *generic* adaptive Web-server extension. Section 3 describes the GALE architecture, stressing the modular construction of communicating components. It also describes how GALE executes adaptation rules. Section 4 describes the adaptation language GAM, a simple textual representation of adaptation models. Section 5 concludes and points to needed and planned future work.

2 A Brief Overview of Adaptive Applications and Platforms

It is impossible to give a complete overview of the introduction of *adaptive functionality* in Web-based systems and applications. We will briefly mention some *influential* developments.

The fields of *intelligent tutoring systems* and *hypermedia* came together when the Web evolved to the point where it became feasible to add enough functionality to the Web server back-end. The award-winning ELM-ART[2] adaptive Lisp course [10] not only paved the way for later developments but also set a standard for *adaptive link annotations* by employing a "traffic light metaphor" of using green and red balls (and some intermediate colors like white and yellow) to indicate the "status" of links to course topics or pages. This metaphor was inherited by many later systems and can still be seen in some GALE applications as well.

When Brusilovsky created Interbook [9] he aimed at creating a platform that could be used for many courses (instead of the single Lisp course offered by ELM-ART). He used Microsoft Word as the authoring platform. An author would essentially write a textbook in Word. Fragments (paragraphs or sections) of text would be associated with some *outcome concepts* by means of a structured comment, and concepts would be indicated as being a *prerequisite* for other concepts, also by means of a comment. Converting a textbook (written in Word) into an adaptive on-line course was reduced to little more than a press of a button.

Adaptation is always based on information the system has about its user. Therefore *user modeling* is a key component in every adaptive system. Typically user modeling is based on rules that "translate" user actions into user

[2] ELM-ART stands for Episodic Learner Model, the Adaptive Remote Tutor.

information. Reading a course page means that you learn about a concept. That knowledge can be confirmed through a test. That knowledge is also used to check whether you satisfy *prerequisites* for studying other concepts. Knowledge levels of small concepts also "add up" to knowledge about chapters and whole courses. This description may give the false impression that an adaptive system really "knows" what goes on in the user's mind, with absolute certainty. There is however an alternative approach to user modeling, using Bayesian networks. This approach is taken by KBS-Hyperbook [16] for instance. User actions change the "belief" of the system that the user has certain knowledge. It is easier to deal with positive and negative "evidence" of the user's knowledge (or interest or any other type of information) in Bayesian networks than in systems that just use (event-condition-action) rules to update knowledge levels.

Recent versions of AHA! (the Adaptive Hypermedia Architecture) [3, 6] allow an author to define arbitrary *adaptation rules* (in fact *user model update rules*) and can thus support any model or interpretation of the information stored about users. Also, unlike Interbook's use of Microsoft Word for authoring, applications in AHA! are authored in HTML, and of course also delivered as HTML. This means that authors familiar with creating HTML can give an AHA! application any look and feel they desire. AHA! can for instance completely mimic the behavior and presentation of Interbook [5]. In order to be able to manage the complexity of having *arbitrary adaptation rules* and *arbitrary presentation* AHA! had to separate the authoring of the adaptation from the authoring of the content and presentation. Defining complex adaptation rules requires very different skills from writing the content of a course. We will see this in GALE as well.

ELM-ART, Interbook and AHA! are examples of systems that support *learning through adaptive information exploration*. The user receives guidance (through link annotation and in AHA! also through the *conditional inclusion of fragments*) but can browse through a course text in any desired way. More support for the *process* of navigation and the use of *services* (going beyond the single step of accessing a page) can be found for instance in APeLS, designed by Conlan and Wade et al [12].

Although adaptive applications in the field of elearning or "technology--enhanced learning" are the most common, there is also significant research in other application areas. Personalized advertising is becoming big business. This is highly visible in Google's personalized ads, but also in recommendations on shopping sites, like the "*people who bought this... also bought...*" messages on Amazon. Museums and other cultural institutes are offering adaptive personalized previews and guided tours or are studying this. The Dutch CATCH (Continuous Access to Cultural Heritage) research program has spawned a lot of research to improve access to cultural information. In one of its projects: CHIP[3] (Cultural Heritage Information Presentation and Personalization) a personalized art recommender was built for (and jointly with) the Rijksmuseum [22], as well as personalized virtual and physical (mobile) guided tours [13]. Italy is also strong in the adaptive access to information about culture and tourism, for

[3] See http://www.chip-project.org/ for more information and demos.

instance through UbiquiTO, a multi-device adaptive guide for Torino [2] and the work on Intelligent Inferfaces for Museum visitors from FBK in Trento [18]. The world of entertainment is following suit with personalized TV guides such as iFanzy[4] [1], a result of the ITEA Passepartout project.

The main difference between the mainstream adaptive educational applications and the others is the role of the *author*. Systems for culture, entertainment and business are typically large special-purpose Web-based Information Systems with an added personalization or adaptation component. There is also a body of research towards making Adaptive Web-based Information Systems more generic [16, 20]. The definition of the adaptation in Adaptive WIS is mostly automatically generated from semantic structures in the databases. In elearning a human author is involved in defining the adaptation at the "instance level", specifying in detail how which action of the learner leads to which change in the user model and how the user model state influences the adaptation. But this is not only the case in elearning. Adaptive (adventure) games for instance also require carefully crafted adaptation rules. GALE attempts to offer a *generic* solution to *authors* of adaptive websites. In the GRAPPLE project (EU FP7 STREP) the *adaptation language* and the graphical *authoring tools* played an important role. The GALE *engine* can be used to serve information where the adaptation rules are generated from (semantic) database structures, but in this paper we concentrate on manually authored applications.

3 The GALE Architecture

Figure 1 below shows the global architecture of GALE. A more elaborate description of GALE can be found in [21]. In this paper we can only present a brief summary of GALE.

Because of lack of space we will concentrate on the *processor stack* (bottom left of the figure) and the distributed way of *executing adaptation rules* (a collaboration between the adaptation engine, domain model and user model services. The former is responsible for performing the actual adaptation and the latter for performing user model updates.

3.1 GALE Processors and Modules

In GALE you can configure a set of *processors* that may operate on an outstanding request. Each processor is only active when the request has reached a state in which it can act upon it, and when finished it updates that state in order to notify the next processor that it can start processing the request. The actual adaptation functionality of GALE can thus be extended in arbitrary ways by adding processors to the stack. Here we briefly explain how a typical (http) request for a concept goes through the stack. We omit details like determining a layout for the presentation and logging for later use with data mining tools.

[4] See http://www.ifanzy.nl/.

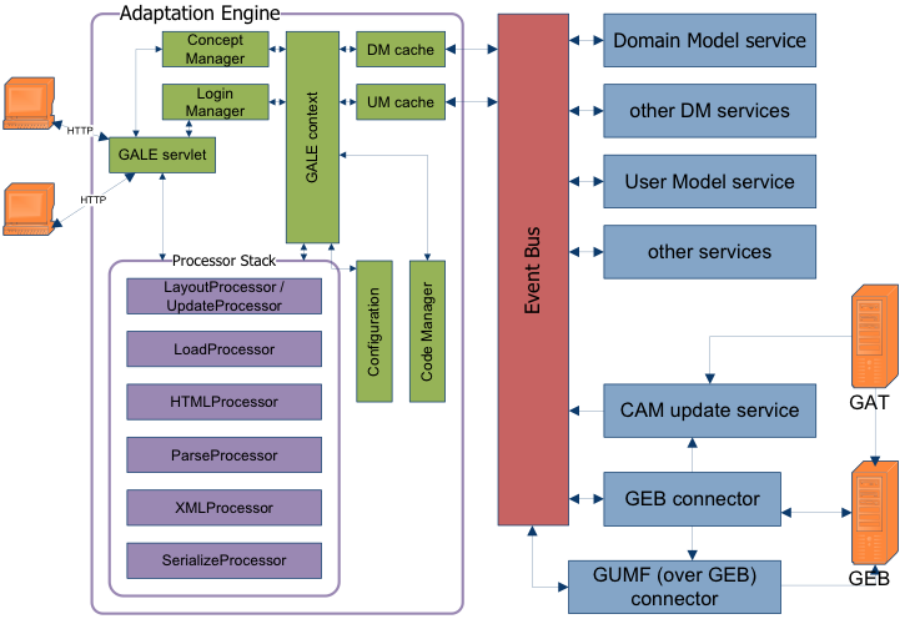


Fig. 1. The GALE architecture

1. When the top left part of the architecture has performed its duties of ensuring that the user is identified and a session is created the first processor to "touch" the request is the *UpdateProcessor*. It signals an *EventManager* that the "access concept" event has occurred. The default *EventAccessHandler* executes the *event code* of the requested concept, as defined in the application's "Domain Model" or DM. This part is concerned with updating the user model and will be described later. The *UpdateProcessor* will wait for the user model (UM) updates to be completed, so as to ensure that any further processing of the event is based on the *new* user model state. (For instance, when a learner requests a concept from a course, the knowledge update that should follow from reading the page associated with the concept is already performed before the page is retrieved, adapted and presented to the learner. It is important for an *adaptation rule designer* to know this order of events.
2. After the UM updates have been performed the *LoadProcessor* will retrieve the actual resource (file) associated with the concept. The name or url of resource may be dependent on UM, which is why UM updates must come first. The *LoadProcessor* retrieves the file (possibly by issuing an http request to a remote server) and creates an *InputStream* that can be used by subsequent processors to load and process the data. File name extensions are used to determine the mime type of the resource, and this may tell processors whether they should handle the request or not. At this point GALE could be extended with processors to handle all kinds of input formats, like images,

drawings, audio, video, etc. but by default GALE only provides processors to deal with HTML and XML.

3. Although GALE can handle HTML it really works with XHTML. When encountering HTML the *HTMLProcessor* uses the (open source) Tagsoup[5] converter to convert the file to XHTML. It creates a new *InputStream* that contains valid XHTML.
4. If the input is XML (also XHTML) the *ParseProcessor* converts the input into an in-memory DOM tree, using the open source dom4j[6] parser.
5. The *XMLProcessor* walks through the DOM tree in order to perform adaptation where needed. The *modules* that may be used to perform adaptation to certain tags are loaded by the *XMLProcessor*. GALE can be configured to associate different modules with different XML tags. The default modules are targeted towards handling XHTML, but any module for handling any kind of adaptation to elements associated with any tag can be added, thus facilitating adaptation to very different types of XML documents, like MusicXML, SMIL, etc. Modules can change the tag name, attributes of tags and the content of the XML elements. (We give some examples later.)
6. The *SerializeProcessor* converts the DOM tree back into the textual XML format and presents that to the *GaleServlet* as an *InputStream* so that it can be sent to the user's browser (as an http response).

In GALE *Modules* are associated with XML tags in order to perform adaptation. Because of space limitations we only explain a few modules here:

- The *IfModule* handles the <if> tag. It expects <if> to have an argument "expr" that is a Boolean expression in GALE code (see Sect. 4.) It expects one or two child elements: a <then> and optionally an <else> element. The module replaces the <if> subtree by either the content of the <then> subtree or the <else> subtree. The *IfModule* thus realizes what is known as the *adaptive inclusion of fragments* technique [17].
- The *AdaptLinkModule* handles the <a> tag which is used just like the HTML <a> tag, but referring to a *concept*, not a *page* or resource. The link adaptation consists of (optionally): adding an icon in front of the link anchor (e.g. a colored ball to implement the "traffic light metaphor"), adding an icon after the link anchor, and adding a "class" attribute to the <a> tag, which in combination with a style sheet changes the presentation of the link. The default stylesheet uses three classes: GOOD, NEUTRAL and BAD and associates these with the colors blue, purple and black, just like AHA! did. Unlike in AHA! the number and choice of classes, colors, icons and conditions for using which class are all unlimited and easily configurable.
- The *VariableModule* replaces the <variable> element by a variable form the user model or the result of an expression. The *AttrVariableModule* does the same for an attribute in the parent tag. In XML the attributes of a tag cannot contain XML tags, so using a <variable> tag inside an XML tag to

[5] See ccil.org/~cowan/XML/tagsoup/ for more information.
[6] See dom4j.sourceforge.net for more information on dom4j.

make an attribute adaptive is not allowed. To make the name of an image in the <img> HTML tag adaptive for instance you can write:

<img><attr-variable name="src" expr="..."></img>

where the expression (not filled in here) results in the name for the image.

3.2 The Execution of GALE Adaptation Rules

As Fig. 1 shows GALE has an "internal" Event Bus through which different components communicate with each other. There are two essential sources of information for adaptive information delivery: the *Domain Model* (DM) that describes the conceptual structure of an adaptive application and the *User Model* (UM) that stores all the information the system can gather about its users. DM and UM services in GALE are separate services that can (if desired) run on different machines.

The DM of an application consists of *concepts* and *relationships*. Concepts have properties, including a title and description but also names (urls) of resources and conditions for selecting which resource. Concepts also have associated *event code* that is executed by the adaptation engine when a user accesses the concept. To minimize communication the adaptation engine maintains a cache of the DMs of the applications it serves.

The UM contains for each user some (arbitrary) personal information and for each concept of each application the user has accessed it contains *event code and* some attribute values. (The *event code* is stored only once but the attribute values are stored for each user.) Since the event code may make use of DM information the UM service has a cache of the DM. And since the adaptation engine needs to frequently access UM information it has a cache of the UM data of its (active) users.

Figure 2 shows how GALE handles UM updates caused by internal *event code* and UM updates received from external sources, such as the GUMF user model service that is part of the GRAPPLE framework.

When *event code* associated with a concept (access) causes a UM update (for instance the access to a concept means that the user gains knowledge about the concept) that update changes an attribute-value in the UMCache. The UMCache wishes to synchronize this update with the UMService and sends a "setum" message to UMService through the EventBus. The change to the attribute-value may trigger *event code* associated with the attribute. This event code is executed within the UMService and results in more UM updates. These updates must be synchronized with the UM cache in the adaptation engine so the set of changes os returned to the UMCache. When the UMCache sends out some updates it always waits for "results" to come back, because the adaptation process must work with the *new* UM state.

When an external UM service sends a UM update (as a "setum" message) this may also cause additional updates, and an "updateum" going to the UMCache. GALE can thus handle UM updates arriving at any time from any service.

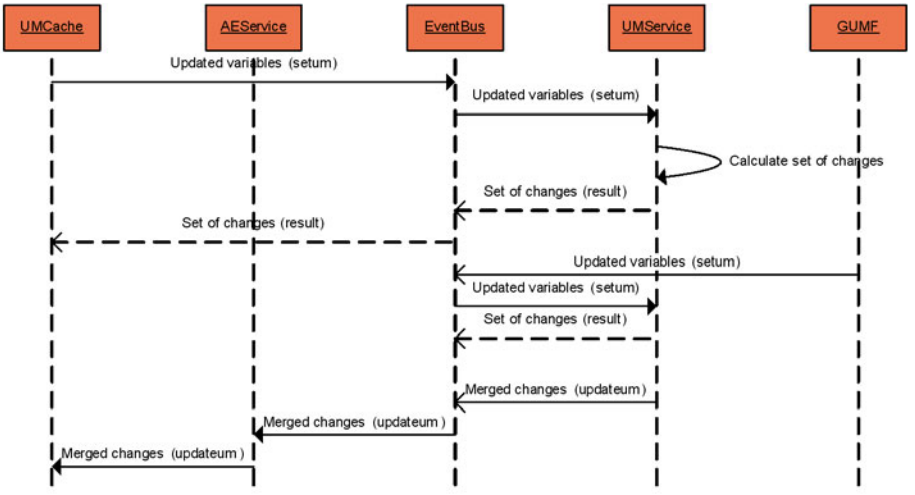


Fig. 2. The process of handling UM updates

4 GAM: The GALE Adaptation Model (GALE Code)

Creating an adaptive application requires that an author or adaptation designer defines *adaptation rules*. This process has been the Achilles heel of adaptive application design from the start. In Interbook [9] and early AHA! versions [4] an author would simply specify *prerequisite* and *outcome* relationships between concepts and the system would automatically generate adaptation rules that resulted in **the** adaptive behavior of the system. Authoring was easy, and flexibility was non-existent. As more flexibility was introduced, for instance in AHA! version 3 [6] a dilemma was born between simplifying authoring through graphical authoring tools [3] and empowering the author through a rich adaptation language. In the GRAPPLE project a graphical authoring environment was created [14] which allowed for easy adaptation design using templates (pedagogical relationship types) while empowering the author by making it possible to design arbitrarily many and complex new templates. In GRAPPLE it is thus possible to do little more than use *prerequisite* and *outcome* relationships, and at the same time it is possible to design the most complex adaptation you can imagine. Here we do not describe the graphical tool but only the underlying GAM language that can also easily be used by an author in a purely textual fashion. We introduce GAM by example (as a complete definition with examples would be far too long).

A GAM definition can be for a single concept or a number of concepts, and can be combined with (X)HTML content as well. Below is an example concept definition[7] for a concept http://gale.win.tue.nl/elearning.xhtml (that can of course be referred to as elearning.xhtml by other concepts on the same server):

[7] We omit some of the "escaping" of < and > symbols that is actually required by the XML syntax.

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns=http://www.w3.org/1999/xhtml
xmlns:gale="http://gale.tue.nl/adaptation">
<head>
<meta name="gale.dm" content="
{ #[visited]:Integer '0' {
event 'if (${#suitability} && ${#read} < 100)
#{#read, 100};
else if (!${#suitability} && ${#read} < 35)
#{#read, 35};'}
#knowledge:Integer ¡ avg(new Object[]
{${<=(parent)#knowledge},${#read}}).intValue()'
#[read]:Integer '0'
#suitability:Boolean 'true'
event '#{#visited, ${#visited}+1};' } " />
< /head>
<body>
<p>This page is a placeholder for the elearning
concept.</p>
< /body>
< /html>
```

The example code has the following semantics:

- #[visited]:Integer '0' means that this concept has a UM attribute called "visited"; it is an integer and is initialized with the value 0. The brackets [...] indicate that the value of this attribute is *stored* permanently.
- The code event '#{#visited, ${#visited}+1};' } means that when the concept is accessed the value of the "visited" attribute in increased by 1.
- When the value of "visited" changes its *event code* is executed which updates the "read" attribute.
- The attribute "read" is also an integer; it is also *stored* or *persistent*.
- The attribute "knowledge" is an integer which is not stored but calculated from the "read" value and the list of "knowledge" values of the children of the "elearning" concept.
- The attribute "suitability" is a Boolean, which is "true" by default. This too is not stored but calculated when needed. If there were prerequisites for the "elearning" concept there would be an expression that defines the condition for the concept to become suitable.

Another concept can "inherit" this adaptation (GAM) code as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns=http://www.w3.org/1999/xhtml
xmlns:gale="http://gale.tue.nl/adaptation">
<head>
<meta name="gale.dm" content= {->(extends)
```

```
http://gale.win.tue.nl/elearning.xhtml}" />
< /head>
<body>
<p>This page uses the elearning template.</p>
< /body>
< /html>
```

When a whole application domain is stored in a single file the "meta" element for the concepts/pages may look like:

```
<meta name='gale.dm' content='redirect:course.gam' />
and the file "course.gam" might have contents like:
welcome.xhtml {
->(extends)http://gale.win.tue.nl/elearning.xhtml
->(extends)layout.xhtml
< −(parent)gale.xhtml
< −(parent)gat.xhtml
}
gale.xhtml {
->(extends)welcome.xhtml
->(parent)welcome.xhtml
}
gat.xhtml {
-> (extends)welcome.xhtml
->(parent)welcome.xhtml
}
layout.xhtml {
#layout:String ‘
<struct cols="250px;*">
<view name="static-tree-view" />
<struct rows="60px;*;40px">
<view name="file-view" file="gale:/header.xhtml" />
<content />
<p><hr />Next suggested concept to study:
<view name="next-view" /></p>
< /struct>
< /struct> ‘
}
```

Again we do not explain this code but just illustrate that code can be shared between different concepts/pages, and can be placed in individual files or combined into a single GAM file. An adaptation designer can create the file "elearning.xhtml" and an application (or course) designer can use the adaptation defines by the designer by simply "extending" that definition and by defining relationships (like "parent") between concepts.

When authoring through GRAPPLE's graphical tool similar templates are used, but instead of "extending" concepts to inherit attribute definitions and adaptation rules templates are "instantiated" (through copying). In terms of adaptive functionality this makes no difference.

The event code in GAM is essentially arbitrary Java code, in which some shorthand notation is used to refer to *properties* and *attributes* of concepts. The shorthand can be summarized as follows:

- Attributes are accessed by using # and properties by using? as part of the syntax of URIs. http://gale.win.tue.nl/someconcept#knowledge refers to the knowledge value for "someconcept" for the current user and #knowledge refers to the knowledge value for the concept to which the code is associated. http://gale.win.tue.nl/someconcept?title refers to the title property of the concept.
- someconcept->(somerelation) represents the list of concepts to which "someconcept"' has the relation "somerelation".
- someconcept<-(somerelation) represents the list of concepts that have a "somerelation" relation to "someconcept".
- ${#knowledge} is the syntax used to *retrieve* the value of the knowledge attribute.
- #{#knowledge, 100} is the syntax used to *set* the value of the knowledge attribute (to 100 in this example).

Using this explanation we can now understand the Java code

```
GaleUtil.avg(new Object[]
{${<-(parent)#knowledge},${#read}}).intValue(),
```

which executes a GALE utility method "avg" on the list composed of all the "knowledge" values of all concepts with a parent relationship to the current concept and the "read" value of the current concept, and which then returns this value rounded to the nearest integer. Such expressions can be used not only in concept and adaptation rule definitions but also in pages, for instance in the expression of an <if> tag. Typically these expressions would be simple, like <if expr="${#visited}==1"><then>This appears on the first visit only </then></if>.

The easiest way to understand and create GAM adaptation code is to look at examples and tutorial material from the GALE website gale.win.tue.nl.

5 Conclusions and Further Work

In the quest for the ultimate powerful, flexible and easy to use adaptive application (authoring and delivery) platform GALE is the most recent attempt. The research area of adaptive Web-based hypermedia systems has evolved from easy to use but very rigid systems (e.g. Interbook [9]) to powerful, flexible and extensible systems (first AHA! [3, 4, 6], now GALE) for which unleashing its power and offering very simple authoring environments is an immense challenge.

In this paper we have explained the modular and highly configurable and extensible architecture of GALE and the powerful GAM adaptation language, and have shown how we have attempted to keep authoring simple by reusing adaptation definitions created by others. Within the GRAPPLE project a graphical authoring environment was created as well. In the future (planned before the presentation of this paper) we will investigate how the graphical and textual authoring methods compare in terms of acceptance by authors. As part of this process more templates (for both authoring environments) will be created and more modules and processors added to GALE as well.

GALE is already being used by researchers in different parts of the world and by companies wishing to start delivering adaptive training material. This will inevitably lead to the discovery of new requirements for more powerful functionality for the next generation adaptive systems, so the quest for the ultimate adaptive system continues.

Acknowledgments. This research was supported by The EU FP7 project GRAPPLE, ICT-2007.4.1, project 215434. Earlier research on AHA! was supported by the NLnet Foundation.

References

1. Akkermans, P., Aroyo, L., Bellekens, P.: iFanzy: Personalised Filtering Using Semantically Enriched TV-Anytime Content. In: Demo at the Third European Semantic Web Conference (2006)
2. Amendola, I., Cena, F., Console, L., Crevola, A., Gena, C., Goy, A., Modeo, S., Perrero, M., Torre, I., Toso, A.: UbiquiTO: A Multi-device Adaptive Guide. In: Brewster, S., Dunlop, M.D. (eds.) Mobile HCI 2004. LNCS, vol. 3160, pp. 409–414. Springer, Heidelberg (2004)
3. De Bra, P., Aerts, A., Berden, B., De Lange, B., Rousseau, B., Santic, T., Smits, D., Stash, N.: AHA! The Adaptive Hypermedia Architecture. In: Proceedings of the Fourteenth ACM Hypertext Conference, pp. 81–84. ACM Press (2003)
4. De Bra, P., Calvi, L.: AHA! An open Adaptive Hypermedia Architecture. New Review of Hypermedia and Multimedia 4, 115–139 (1998)
5. De Bra, P., Santic, T., Brusilovsky, P.: AHA! meets Interbook and more... In: Proceedings of the AACE ELearn 2003 Conference, pp. 57–64 (2003)
6. De Bra, P., Smits, D., Stash, N.: The Design of AHA! In: Proceedings of the Seventeenth ACM Conference on Hypertext and Hypermedia, pp. 133–134. ACM Press (2006), http://aha.win.tue.nl/ahadesign/
7. Brusilovsky, P.: Methods and Techniques of Adaptive Hypermedia. In: User Modeling and User-Adapted Interaction, vol. 6, pp. 87–129. Kluwer Academic Publishers (1996)
8. Brusilovsky, P.: Adaptive Hypermedia. In: User Modeling and User Adapted Interaction, vol. 11, pp. 87–110. Kluwer Academic Publishers (2001)
9. Brusilovsky, P., Eklund, J., Schwarz, E.: Web-based education for all: A tool for developing adaptive courseware. In: Computer Networks and ISDN Systems, Proceedings of the 7th Int. World Wide Web Conference, vol. 30(1-7), pp. 291–300 (1998)

10. Brusilovsky, P., Schwarz, E.W., Weber, G.: ELM-ART: An Intelligent Tutoring System on World Wide Web. In: Lesgold, A.M., Frasson, C., Gauthier, G. (eds.) ITS 1996. LNCS, vol. 1086, pp. 261–269. Springer, Heidelberg (1996)
11. Bush, V.: As We May Think. The Atlantic Monthly (1945)
12. Conlan, O., Wade, V.P.: Evaluation of APeLS – An Adaptive eLearning Service Based on the Multi-model, Metadata-Driven Approach. In: De Bra, P.M.E., Nejdl, W. (eds.) AH 2004. LNCS, vol. 3137, pp. 291–295. Springer, Heidelberg (2004)
13. van Hage, W.R., Stash, N., Wang, Y., Aroyo, L.M.: Finding Your Way Through the Rijksmuseum with an Adaptive Mobile Museum Guide. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010. LNCS, vol. 6088, pp. 46–59. Springer, Heidelberg (2010)
14. Hendrix, M., Cristea, A.I.: Design of the CAM model and authoring tool. In: A3H: 7th International Workshop on Authoring of Adaptive and Adaptable Hypermedia Workshop, 4th European Conference on Technology-Enhanced Learning (2009)
15. Henze, N.: Adaptive hyperbooks: Adaptation for project-based learning resources. PhD Dissertation, University of Hannover (2000)
16. Houben, G.J., van der Sluijs, K., Barna, P., Broekstra, J., Casteleyn, S., Fiala, Z., Frasincar, F.: Hera. In: Web Engineering: Modeling and Implementing Web Applications. Human-Computer Interaction Series, pp. 263–301. Springer, Heidelberg (2008)
17. Knutov, E., De Bra, P.M.E., Pechenizkiy, M.: AH 12 years later: a comprehensive survey of adaptive hypermedia methods and techniques. New Review of Hypermedia and Multimedia 15(1), 5–38 (2009)
18. Kuflik, T., Stock, O., Zancanaro, M., Gorfinke, A., Jbara, S., Kats, S., Sheidin, J., Kashtan, N.: A Visitor's Guide in an Active Museum: Presentations, Communications, and Reflection. ACM Journal on Computing and Cultural Heritage 3(3) (2011)
19. Nelson, T.: The Hypertext. In: Proc. World Documentation Federation Conf. (1965)
20. Rossi, G., Schwabe, D.: Modeling and Implementing Web Applications with Oohdm. In: Modeling and Implementing Web Applications. Human-Computer Interaction Series, pp. 109–155. Springer, Heidelberg (2008)
21. Smits, D., De Bra, P.: GALE: A Highly Extensible Adaptive Hypermedia Engine. In: Proceedings of the Twenty-Second ACM Hypertext Conference, pp. 63–72. ACM Press (2011)
22. Wang, Y., Stash, N., Aroyo, L., Gorgels, P., Rutledge, L., Schreiber, G.: Recommendations Based on Semantically-enriched Museum Collections. Journal of Web Semantics: Science, Services and Agents on the World Wide Web 6(4), 43–50 (2008)

Multi Feature Indexing Network MUFIN for Similarity Search Applications

Pavel Zezula

Masaryk University,
Botanicka 68a, Brno, Czech Republic
zezula@fi.muni.cz

Abstract. Similarity has been a central notion throughout our lives and due to the current unprecedented growth of digital data collections of various types in huge quantities, similarity management of digital data is becoming necessary. The Multi-Feature Indexing Network (MUFIN) is a generic engine for similarity search in various data collections, such as pictures, video, music, biometric data, sensor and scientific data, and many others. MUFIN can provide answers to queries based on the example paradigm. The system assumes a very universal concept of similarity that is based on the mathematical notion of metric space. In this concept, the data collection is seen as objects together with a method to measure similarity between pairs of objects. The key implementation strategies of MUFIN concern: *extensibility* - to be applied on variety of data types, *scalability* - to be efficient even for very large databases, and *infrastructure independence* - to run on various hardware infrastructures so that the required query response time and query execution throughput can be adjusted. The capability of MUFIN is demonstrated by several applications and advance prototypes. Other applications and future research and application trends are also to be discussed.

1 Motivation

Who on the Facebook collect photographs similar to those I love most from my latest holidays? Does the computer disk of a suspected criminal contain illegal multimedia material? Is it the situation on the web getting close to any of the network attacks which resulted in significant damage in the past? These and many other question of this form are interesting and even urgent when dealing with digital content on the web, but the solution to any of them needs to apply a certain form of similarity data processing. However, unless some text tags or other form of metadata are available, current technology is not able to solve them. No doubts, the similarity approach to digital text processing has proved big success even at commercial level, but we need significant scientific advancement in both the effectiveness and efficiency of the similarity data management for any type of data, currently available in digital form.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 77–87, 2012.

Independently of computers, similarity is a central notion throughout our lives, [5]. In perception, the similarity between sets of visual or auditory stimuli influences the way in which they are grouped. In speech recognition, the similarity between different phonemes determines how easily confused they are. In classification, the category assignment to a new instance may be influenced by the similarity of a new instance to past instances or a store prototype. In memory, it has been suggested that retrieval from a cue depends on the similarity of past memory traces to the representation of the cue. Provided that almost everything that we see, read, hear, write, measure, etc. can be available in digital forms, computer systems must manage such data through similarity.

In conjunction, the growth of digital material is posing another big challenge: the exponential increase in data volume makes the scalability a matter of concern. Accordingly, the core ability of data processing systems of the future is the similarity management of (very) large and ever-growing volumes of data. What is needed is to create new models, methodologies, and theories that would form solid bases of future computer systems for advanced practical applications fully exploiting contemporary digital databases. Examples of such applications come from areas such as the web searching, biology, geography, multimedia, data cleaning and integration, data mining, web anomaly detection, context similarity detection, collaborative filtering, security, etc.

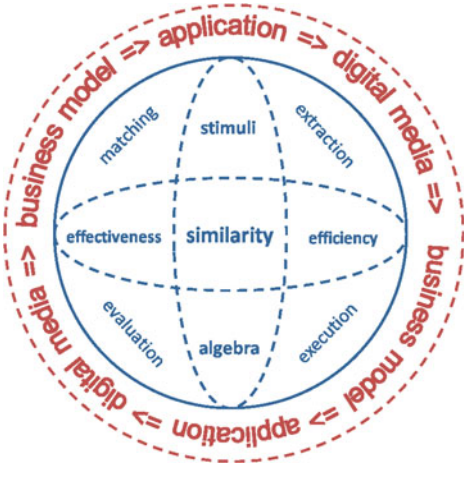


Fig. 1. The problem of similarity searching in digital data

In the future, access to digital media stored in memories and circulating among networked computers will have to follow, or at least get much closer in its form, to the behavior of real life evolution and communication between species. There, recognition, learning, and judgment presuppose an ability to categorize stimuli and classify situations by similarity, because any event in the history of organism is, in a sense, unique and the specific case of the exact match is marginal.Suitable

synonyms of similarity are proximity, resemblance, communality, representativeness, psychological distance, etc. As any kind of fact can nowadays become a digital part of the networked media, computers must provide access to required data through similarity based operations, because it is the similarity that are in the world "revealing".

If ever, current computers can only partially deal with similarity. On the other hand, the great commercial success of companies as Google, Yahoo, and Microsoft with text similarity searching, proves the potential of similarity and, once properly applied, runs to seed many other important and profitable applications and businesses. The key to its success is to significantly enhance the technology for digital similarity searching or even generic similarity management both from the effectiveness and efficiency points of view. We are convinced that this is the way, how the endless chain of "digital media – business model – application" can be accelerated to bring needed commercial success. See Fig. 1 for convenience.

2 Challenges

Decentralized media of mass communication, sometimes called the networked media, allow cooperative and collaborative practices enabling users to autonomously contribute to production of global media, the elements of which are in fact related by numerous multi-facet links of similarity. As an example, consider the sites like Flicker, YouTube, Facebook which host user contributed content for a variety of events. The exploitation of its potential, which typically goes across multiple applications and has to consider different aspects of underlying data, is vitally important for various business and private purposes and activities. In this way, similarity relationships, once properly defined and implemented, can act as a sort of "similarity glue" able to connect, search, filter, merge, relate, rank, classify, or categorize seemingly unrelated collections of objects across resources from diverse environments and produced by different activities. In a very simplified way, the challenge is applying a proper technology on the immerging data.

2.1 Digital Data Explosion

The standard starting point for almost every big-picture opinion of our industry is to reflect on the data explosion. According to recent studies, in the next three years, we will create more data than has been produced in all of human history. But just where is all this data coming from? It is estimated that more than 80 billion photographs was taken in 2008. To store them all digitally would require 400 petabytes of storage. Nowadays, more than 50 million photos are daily uploaded on Facebook, and the numbers are growing.

The Enterprise Storage Group estimates that it takes as much space to store an average digital photo as it does to store 30 pages of digital text with 600 words

for page. So digitally, a picture is worth 18,000 words. The management of digital images promises to emerge as a major issue in many enterprises in the next couple of years, particularly since a large portion of pictures still remains as "unstructured data." – managing this data is going to provide a lot of opportunities for data integration and content management. But we have also digital audio and numerous new forms of digital data produced by scientific explorations, analyses and experiments - not to talk about digital video and digital television broadcast material. The rise of the digital material is posing a big challenge.

But it is not just the volume of digital material, putting strong constraints on scalability issues of all data processing technologies. Contrary to traditional attribute-like data types such as numbers and strings of sortable domains, instances of new data types are complex and need innovative approaches of data analysis to discover their content and derive stimuli for content management. As a result, many domains of new data types are not sortable and the only measure of comparison to apply is a sort of similarity.

2.2 Similarity Management of Data

As illustrated in Fig. 1, similarity is determined by *stimuli* (features, descriptors, properties, etc.) of digital objects and the way we pursue processes of finding similarity-related objects, that is a sequence of operations from specific *algebra* to satisfy required demands. We can calibrate the stimuli and operations from two different points of view: (1) the *effectiveness*, a similarity model, concerns the way similarity is assessed so that it best reflects a human vision of similarity, and (2) the *efficiency*, which regards the processing speed, costs, or an effort to get results independently from the scale fast.

The effectiveness of stimuli concerns the *matching* of stimuli, typically measured as a level of similarity, usually implemented as a dissimilarity or distance. On the other hand, the effectiveness of operations can be measured as a capability of the similarity algebra to *evaluate* objectives of similarity models applied. The efficiency of the algebra operations regards *execution* processes on operations, i.e. the performance on a suitable software and hardware infrastructure, while efficiency with respect to stimuli involves the ease of processes able to *extract* fast from the raw data suitable stimuli.

The successful but very limited application of similarity management for text data needs many new examples to follow. The digital media is becoming not only more and more bulky, but also more and more rich in content and the portion of the data produced in text form is shrinking. The main reason for not enough big and successful new applications is the luck of adequate technology. We need new technology for similarity management, with substantial improvements both in effectiveness and efficiency dimensions. The problem is complex and needs a joined effort of scientists from different research domains. Contacts with and inputs from industries represent a vitally important feedbag so that the research proceeds in proper direction. However, the potential impact is huge because it can give rise to many new businesses. Examples of application areas include:

multimedia, electronic advertising, electronic commerce, entertainment, security, processing of biometric data, network traffic control, medical information systems, biology (chemical)-data processing, geographic systems, criminology, intelligence, classification and discovery of illegal or objectionable content, data cleaning and integration, ambient systems, mobile computing, social network analysis and data mining, semantic web and ontology management.

2.3 The Main Research Objectives

Following Fig. 1 again, the main areas of research challenges can be summarized as follows:

effectiveness - abstract (psychological) models of similarity; it concerns methods, methodologies, and architectures for similarity judgements, as well as techniques to assess their quality and user satisfaction.

efficiency - generic models for high-performance similarity management; in general, it concerns, architectures, models, infrastructures, and other software and hardware mechanisms able to support fast execution of computational intensive tasks, for example, parallelism, GPU's, scalable and distributed architectures, self-organizing principles, cloud computing, fault tolerance, etc.

stimuli - representations of raw digital data (content); they are purpose-built representations of raw (digital) data objects encapsulating specific knowledge about or content of these objects. The representation of the knowledge should be maximally precise, but at the same time minimum in space for storage purposes.

algebra - set of similarity operations; the basic similarity operation is the search, which actually comes in several forms, such as the range search, nearest neighbor search, reverse nearest neighbor search, etc. But many other operations can be defined as well, for example, a similarity ranking, classification, or merging. The objective is to develop similarity algebra of operations over digital data.

matching - specification of similarity measures and their assessment; it concerns the way how, given specific stimuli, required effectiveness is quantified. Obviously, given specific sets of stimuli, there are several ways how to compare them. For example, collections of numbers can be compared as the Euclidean distance of vectors, edit distance can be applied on strings, and Jaccard's Coefficient on sets.

extraction - efficient harvesting of stimuli; computational methods of stimuli extraction respecting specific types of raw data and stimuli type - depending on the volume of data, such process can be enormously time-consuming thus a special care must be taken from the efficiency point of view.

evaluation - implementation of similarity models through operations; the problem starts from users, their needs, unique attitude or context and requirements expressed through a specific model of similarity. It concerns a requirement specification tool (a language) to express needs as well as a communication environment, or interface, able to report results.

execution - performance oriented implementation of similarity operation transactions; it concerns platform dependent implementation techniques to achieve required performance.

Though many interesting proposals have already been published and even implemented, there is still a lot to do. Such research effort lies at the frontier of modern digital data processing and goes substantially beyond the current state of the art. It offers a high potential for exploitation in numerous applications. It is interdisciplinary in nature and requires experts and experience not only from computer science. For example, the subject of similarity has been for long time important research domain in psychology. The research is ground-breaking as new theories and technologies specifically targeted at similarity for large scale problems are scarce and badly needed. It is also a high-gain/high-risk research since the complexity may even exceed our expectations and new problems can occur. On the other hand, similarity is typically country and language independent with an easy and clear impact on many international communities. It is foundational in nature, because it requires new theories for similarity assessment, feature extraction and retrieval, as well as large scale autonomic computation. It is also a long-term research because the needs of similarity management may change with new patterns of computation and technology for content extraction and enrichment.

3 The MUFIN Approach

Considering searching as an activity of looking thoroughly in order to find something or someone, an investigation seeking answers, an operation that determines whether one or more of a set of items has a specified property, the generic goal of the Multi-Feature Indexing Network (MUFIN) project is to develop a technology solution to the problem of similarity searching for various and very large digital data collections.

3.1 The Vision

It is generally agreed that the search problem is a triple of: (1) data and queries, (2) indexing structure, and (3) computing infrastructure on which the search is executed. As it applies to any other complex system, we believe that future search systems must be born on the divergence of the *scale* and *determinism*. In particular, a more and more desirable property of any search system is its ability to either handle growing amounts of work in a graceful manner, or to be readily enlarged – that is scalability. On the other hand, the necessity of search processes to be determined by an unbroken chain of pre-defined steps – that is determinism – is in search becoming less and less important. The effects of the divergence of scale and determinism on the historical development of search structures are illustrated in Fig. 2. It starts with the well established centralized and parallel organizations, continues through the cutting-edge distributed and

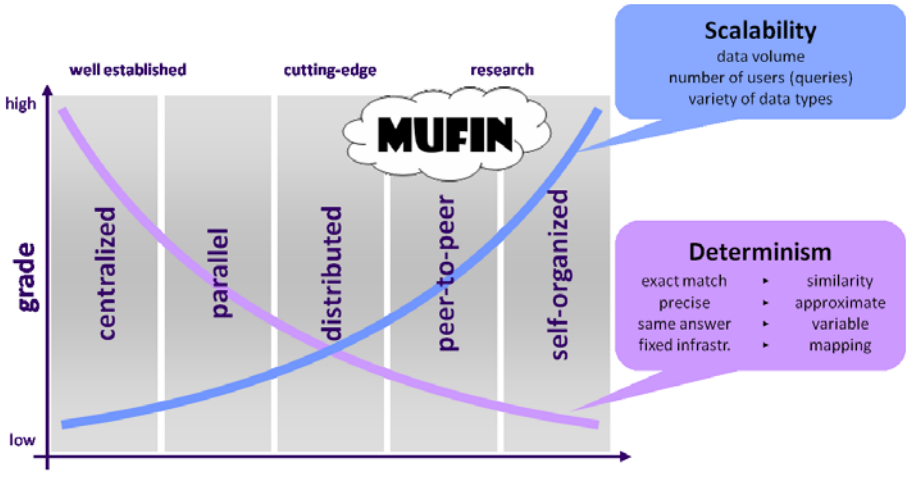


Fig. 2. Development trends in search structures

peer-to-peer approaches and aims towards self-organizing search architectures. In this figure, the position of MUFIN is also clearly illustrated.

Current trends in scale and determinism of search systems can be characterized by the following headlines:

scalability – exponential growth in data volume
- number of users (queries) increasing fast
- variety of data types, emergence of various digital databases and libraries
- multi-lingual (feature, modal) queries

determinism – from exact match to similarity
- from precise query evaluation to approximate evaluation
- from unvaried answer to satisfactory answer
- from fixed queries to personalized queries
- from dedicated hardware to dynamic hardware mapping

3.2 The Underlying Paradigms

MUFIN, http://mufin.fi.muni.cz/tiki-index.php, is a generic engine for similarity search in various data collections, such as pictures, video, music, biometric data, sensor and scientific data, and many others, using specific descriptors (features) extracted from original objects. The technology has been developed for about 15 years and the main properties are summarize in the book "Similarity Search: the Metric Space Approach" [10], published by Springer. Recently, a team of researchers at the Masaryk University has built a robust prototype and brought this technology to a productive stage. MUFIN assumes a very universal model of similarity that is based on the mathematical notion of metric space. In this

concept, the data collection is seen as objects represented by specific features together with a method to measure similarity between pairs of objects as distance – the smaller the distance of two objects, the more similar they are.

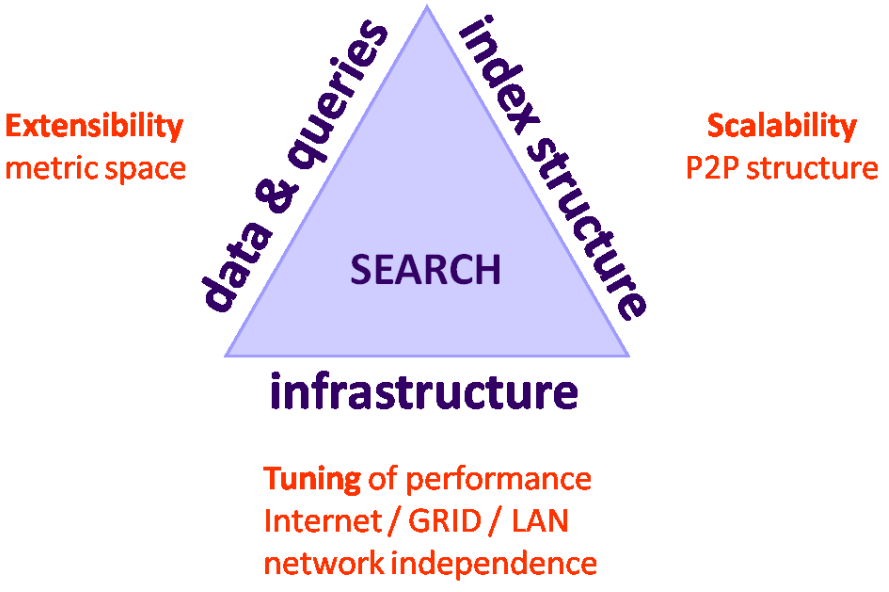


Fig. 3. Implementation Paradigms of MUFIN

Specifically, MUFIN can efficiently index large amounts of data and provides online evaluation of similarity queries like "Give me k images from the database most similar to this photo", i.e. it uses the query-by-example search paradigm for the nearest neighbor or range similarity queries. As seen in Fig. 3, the search problems of data and queries, indexing structures, and computational infrastructure are in MUFIN based on the following three paradigms:

Extensibility. MUFIN can be used practically on data of any type, for example vectors, strings, or sets, provided the similarity space can be modeled as the metric space [8]. For many examples see the books [10] or [9].

Scalability. MUFIN is efficient even for very large databases. Its indexing structures are based on structured P2P networks, so there is no bottleneck in form of central data partitions or services. Specification of different implementation alternatives applied in MUFIN can be found in [1,3,4,7,11,6]. The index can be in-memory or disk-based, centralized or distributed, internally replicated. The trade-off between query efficiency and the quality of results is fully tunable.

Infrastructure independence. MUFIN search technologies can run on various HW infrastructures including large-scale distributed computer clouds.

Migrating the search engine to a different hardware can be used for performance tuning on specific, possibly dynamic, data collections. Specifically, MUFIN can adjust the query response time and the multiple query evaluation throughput even during the online query processing. The implementation is based on the Metric Similarity Search Implementation Framework, http://lsd.fi.muni.cz/trac/messif/, MESSIF, also described in [2].

3.3 Demonstrations and Applications

Currently, both the concept and its implementation are relatively mature. The technology have been successfully applied by several stock photo applications, for example Pixmac http://www.pixmac.com and Profimedia http://www.profimedia.cz.

Fig. 4. Search results from MUFIN demo

A publicly-available demo http://mufin.fi.muni.cz/imgsearch/, based on the M-Chord [7] P2P protocol, performs similarity searching on a half a billion MPEG-7 global descriptors derived from 100 million Flickr images. The specific descriptors applied are the *color structure, scalable color, color layout, edge histogram* and *homogeneous texture*. The network consists of 500 peers operating on two IBM X3400 servers, each equipped with two quad-core processors, 20 GB RAM and five SATA hard drives. Due to the size of data, the descriptors are stored on the disks. The typical searching performance is below 500ms and the query throughput is 10 queries per second. For illustration, Fig. 4 contains an example of a search result – the top-left image is the query.

Other demos can be found at http://mufin.fi.muni.cz/apps.html. In the area of images, they demonstrate global search on qualitatively different datasets,

a small demo also concerns retrieval of sub-images. Demos also show how various ranking strategies can improve effectiveness of search results. Application of similarity searching to automatic image annotation is also available. But the MUFIN technology has also been used to biometric data, such as fingerprints and faces or to time series in general, having potentially very large spectrum of possible applications. Small demonstrations of such cases are also available on this web page.

In 2008, the project received the IBM Shared University Research (SUR) Award including also interesting hardware infrastructure. Thanks to this donation, we are able to run all of the demonstrations online.

4 Future Trends and Conclusions

Similarity search in collections of complex digital data has proved useful, and MUFIN is certainly a viable approach to needed technology. Though several applications are already operational a many others are being considered, there are still problems which prevent faster development.

First, the similarity search is still expensive considering both the processing time a storage costs, thus better paradigms and techniques to feature extraction and query execution, properly exploiting up to date computing infrastructures are still needed. Second, existing software is typically not available in form of simple and ready to use packages and developing of applications requires project-like approaches. These are costly, consuming extensive humane and infrastructure resources, which are scarce, not always available.

A promising alternative to be exploited in the future is to develop similarity search services based on the Cloud Computing paradigm. The expected positive properties can mainly be attributed to the characteristic Cloud Computing system properties of:

Scalability - to support huge databases and very high request rates – cloud systems are designed to scale-out, so that large scale is achieved using large number of commodity servers, running in parallel. An effective scale-out system must balance load across servers and avoid bottlenecks.

Elasticity - it means that we can add more capacity to a running system by deploying new instances of each component, and shifting load to them.

Availability - cloud systems provide high levels of availability. In particular, they are often multi tenant systems, which mean that a possible outage may affect many different applications.

Privacy - outsourcing based on cloud computing is attractive as it promises pay-as-you-go low storage costs as well as easy data access. However, care needs to be taken to safeguard data that is valuable or sensitive against unauthorized access.

Recently, there has been an explosion of new systems for data storage and management "in the cloud". Open source systems include Cassandra, HBase, Voldemort, and many others. Some systems offered as cloud services, either directly

in the case of Amazone SimpleDB and Microsoft Azure SQL Services, or as a part of programming environment like Google's AppEngine. Many of the systems are also refered to as "key-value stores" and concentrate on massive scaling and elasticity to simplify application development and deployment of classical (attribute or keyword) data. To the best of our knowledge, the case of similarity search cloud searing has not been considered yet.

Acknowledgments. This research was partially supported by the Czech Science Foundation project No. P103/10/0886 and the Czech MV research grant No. FV20102014004.

References

1. Batko, M., Novak, D., Falchi, F., Zezula, P.: On scalability of the similarity search in the world of peers. In: INFOSCALE, pp. 1–12. ACM (2006)
2. Batko, M., Novak, D., Falchi, F., Zezula, P.: MESSIF: Metric Similarity Search Implementation Framework. In: Thanos, C., Borri, F., Candela, L. (eds.) Digital Libraries: Research and Development. LNCS, vol. 4877, pp. 1–10. Springer, Heidelberg (2007)
3. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: VLDB, pp. 426–435. Morgan Kaufmann (1997)
4. Dohnal, V., Gennaro, C., Savino, P., Zezula, P.: D-Index: Distance searching index for metric data sets. Multimedia Tools and Applications 21(1), 9–33 (2003)
5. Larkey, L., Markman, A.B.: Processes of similarity judgment. Cognitive Science 29, 1061–1076 (2005)
6. Novak, D., Batko, M., Zezula, P.: Metric index: An efficient and scalable solution for precise and approximate similarity search. Inf. Syst. 36(4), 721–733 (2011)
7. Novak, D., Zezula, P.: M-Chord: A scalable distributed similarity search structure. In: INFOSCALE, pp. 1–10. IEEE (2006)
8. O'Searcoid, M.: Metric Spaces. Springer, Heidelberg (2006)
9. Samet, H.: Foundations of Multidimensional And Metric Data Structures. Series in Data Management Systems. Morgan Kaufmann (2006)
10. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: The Metric Space Approach. Advances in Database Systems, vol. 32. Springer, Heidelberg (2006)
11. Zezula, P., Savino, P., Rabitti, F., Amato, G., Ciaccia, P.: Processing M-trees with parallel resources. In: RIDE, pp. 147–154. IEEE (1998)

Recent Challenges and Ideas in Temporal Synthesis

Orna Kupferman

Hebrew University, School of Engineering and Computer Science,
Jerusalem 91904, Israel
orna@cs.huji.ac.il

Abstract. In automated synthesis, we transform a specification into a system that is guaranteed to satisfy the specification against all environments. While model-checking theory has led to industrial development and use of formal-verification tools, the integration of synthesis in the industry is slow. This has to do with theoretical limitations, like the complexity of the problem, algorithmic limitations, like the need to determinize automata on infinite words and solve parity games, methodological reasons, like the lack of satisfactory compositional synthesis algorithms, and practical reasons: current algorithms produce systems that satisfy the specification, but may do so in a peculiar way and may be larger or less well-structured than systems constructed manually.

The research community has managed to suggest some solutions to these limitations, and bring synthesis algorithms closer to practice. Significant barriers, however, remain. Moreover, the integration of synthesis in real applications has taught us that the traditional setting of synthesis is too simplified and has brought with it new algorithmic challenges. This paper introduces the synthesis problem, algorithms for solving it, and recent promising ideas in making temporal-synthesis useful in practice.

1 Introduction

One of the most significant developments in the area of system verification over the last two decades has been the development of algorithmic methods for verifying temporal specifications of finite-state systems; see [13]. A frequent criticism against this approach, however, is that verification is done *after* significant resources have already been invested in the development of the system. Since systems invariably contain errors, verification simply becomes part of the debugging process. The critics argue that the desired goal is to use the specification of the system in the development process in order to guarantee the design of correct systems. This is called *system synthesis*.[1]

In the late 1980s, several researchers realized that the classical approach to system synthesis, where a system is extracted from a proof that the specification is satisfiable, is well suited to *closed* systems, but not to *open* (also called *reactive* [22]) systems

[1] To make life interesting, several different methodologies in system design are all termed "synthesis". The automatic synthesis we study here should not be confused with *logic synthesis*, which is a process by which an abstract form of a desired circuit behavior (typically, register transfer level, which by itself may be the outcome of yet another synthesis procedure, termed *high-level synthesis*) is turned into a design implementation by means of logic gates.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 88–98, 2012.

[1,14,44]. In reactive systems, the system interacts with the environment, and a correct system should then satisfy the specification with respect to all environments. If one applies the techniques of [18,38] to reactive systems, one obtains systems that are correct only with respect to some environments. Pnueli and Rosner [44], Abadi, Lamport, and Wolper [1], and Dill [14] argued that the right way to approach synthesis of reactive systems is to consider the situation as a (possibly infinite) game between the environment and the system. A correct system can be then viewed as a winning strategy in this game. It turns out that satisfiability of the specification is not sufficient to guarantee the existence of such a strategy. Abadi et al. called specifications for which a winning strategy exists *realizable*. Thus, a strategy for a system with inputs in I and outputs in O maps finite sequences of inputs — words in $(2^I)^*$, which correspond to the actions of the environment so far, to an output in 2^O — a suggested action for the system. A strategy can then be viewed as a labeling of a tree with directions in 2^I by labels in 2^O. The traditional algorithm for finding a winning strategy transforms the specification into a parity automaton over such trees. The automaton accepts a tree if the tree models a strategy all of whose computations satisfy the specification, and so a specification is realizable precisely when this tree automaton is nonempty, i.e., it accepts some infinite tree [44]. A finite generator of an infinite tree accepted by this automaton can be viewed as a finite-state system realizing the specification. This is closely related to the approach in [8,47] to solve Church's *solvability problem* [12]. In [32,45,54,57] it was shown how this approach to system synthesis can be carried out in a variety of settings.

In spite of the rich theory developed for system synthesis, little of this theory has been reduced to practice. Some people argue that this is because the realizability problem, and hence also the synthesis problem, for linear-temporal logic (LTL) specifications is 2EXPTIME-complete [44,49], but this argument is not compelling. First, experience with verification shows that even nonelementary algorithms can be practical, since the worst-case complexity does not arise often. For example, while the model-checking problem for specifications in second-order logic has nonelementary complexity, the model-checking tool MONA [17,29] successfully verifies many specifications given in second-order logic. Furthermore, in some sense, synthesis is not harder than verification. This may seem to contradict the known fact that while verification is "easy" (linear in the size of the model and at most exponential in the size of the specification [35]), synthesis is hard (2EXPTIME-complete). There is, however, something misleading in this fact: while the complexity of synthesis is given with respect to the specification only, the complexity of verification is given with respect to the specification and the system, which can be much larger than the specification. In particular, it is shown in [49] that there are temporal specifications for which every realizing system must be at least doubly exponentially larger than the specifications. Clearly, the verification of such systems is doubly exponential in the specification, just as the cost of synthesis.

We believe that there are other reasons for the lack of practical impact of synthesis theory: algorithmic, methodological, scope, and qualitative. Let us start with the algorithmic difficulties. First, the construction of tree automata for realizing strategies uses determinization of Büchi automata. Safra's determinization construction [51] has been notoriously resistant to efficient implementations [2,53] (Safra's construction was recently improved in [41]. While the new construction results in automata with fewer

states, it suffers from the same problems that make Safra's construction resistant to implementations.) Second, determinization results in automata with a very complicated state space. The best-known algorithms for parity-tree-automata emptiness [26] are nontrivial already when applied to simple state spaces. Implementing them on top of the messy state space that results from determinization is awfully complex, and is not amenable to optimizations and a symbolic implementation.

Another major issue is methodological. The current theory of system synthesis assumes that one gets a comprehensive set of temporal assertions as a starting point. This cannot be realistic in practice. A more realistic approach would be to assume an *evolving* formal specification: temporal assertions can be added, deleted, or modified. Since it is rare to have a complete set of assertions at the very start of the design process, there is a need to develop *compositional* synthesis algorithms. Such algorithms can, for example, refine designs when provided with additional temporal properties. Moreover, development of complex systems is typically modular, with components being composed into larger components. Again, this is in contrast with current theory, which assumes a "flat" global specification. A more realistic approach would be to assume a *modular* setting, where specifications are given, and systems are generated, in a hierarchical manner.

A third drawback of current synthesis algorithms is their scope. Driven by the growing industrial impact of formal methods, various industrial efforts were launched recently to develop industrial-strength temporal assertion languages for semiconductor designs; e.g., Intel's ForSpec and IBM's Sugar [4,6]. In the related application of model checking, theory has already bridged the gap with the new industrial-strength formalisms. In synthesis, theory still assumes specifications in traditional temporal logic like LTL. The richer formalisms require a nontrivial extension of current solutions. An even more interesting extension of the scope of synthesis refers to the underlying setting. It is by now realized that requiring the synthesized system to satisfy the specification against all possible environments is often too demanding. There is a need to define variants of synthesis that replace the universal quantification by a richer one.

Finally, while current automated synthesis algorithms generate a system that satisfies the specification, there is no emphasize on constructing optimal or well-structured systems. Indeed, the systems are obtained by extending algorithms that check nonemptiness of tree automata to return a witness to the nonemptiness, and there is no work on defining, measuring, and optimizing the quality of this witness. In particular, the synthesized systems performs as a black box, and there is no attempt to make its internal structure friendly to work with. Thus, automatic synthesis may result in systems that are larger and less structured than systems generated manually.

This paper surveys recent work that address the above challenges, describe new synthesis algorithms and paradigms, and discuss further challenges that they bring with them.

2 Preliminaries

Consider finite sets I and O of input and output signals, respectively. We model finite-state reactive systems with inputs in I and outputs in O by *transducers* (I/O-transducers, when I and O are not clear from the context). A transducer is a finite graph with a designated start state, where the edges are labeled by letters in 2^I and the states

are labeled by letters in 2^O. Formally, a transducer is a tuple $\mathcal{T} = \langle I, O, S, s_{in}, \eta, L\rangle$, where I and O are the sets of input and output signals, S is a finite set of states, $s_{in} \in S$ is an initial state, $\eta : S \times 2^I \to S$ is a deterministic transition function, and $L : S \to 2^O$ is a labeling function. We extend η to words in $(2^I)^*$ in the straightforward way. Thus, $\eta : (2^I)^* \to S$ is such that $\eta(\varepsilon) = s_{in}$, and for $x \in (2^I)^*$ and $i \in 2^I$, we have $\eta(x \cdot i) = \eta(\eta(x), i)$. Each transducer $\mathcal{T}$ induces a *strategy* $f_{\mathcal{T}} : (2^I)^* \to 2^O$ where for all $w \in (2^I)^*$, we have $f_{\mathcal{T}}(w) = \tau(L(w))$. Thus, $f_{\mathcal{T}}(w)$ is the letter that $\mathcal{T}$ outputs after reading the sequence w of input letters. The strategy $f_{\mathcal{T}}$ generates computations over the set $I \cup O$ of signals. A computation $\rho \in (2^{I\cup O})^\omega$ is generated by $f_{\mathcal{T}}$ if $\rho = (i_0 \cup o_0), (i_1 \cup o_1), (i_2 \cup o_2), \ldots$ and for all $j \geq 1$, we have $o_j = f_{\mathcal{T}}(i_0 \cdot i_1 \cdots i_{j-1})$. We sometimes refer to ρ as a computation of $\mathcal{T}$.

Linear temporal logic (LTL) is a formalism for specifying on-going behaviors of reactive systems [43]. Given an LTL formula ψ over the sets I and O of input and output signals, the *realizability problem* for ψ is to decide whether there is an I/O-transducer $\mathcal{T}$ such that all the computations of $\mathcal{T}$ satisfy ψ [44].

Algorithms for solving the synthesis problem are based on automata on infinite words and trees. We assume that the reader is familiar with the basic definitions of alternating tree automata. All the notations we are going to use are these defined and used in [33]. We do define here trees and labeled trees. Given a set D of directions, a *D-tree* is a set $T \subseteq D^*$ such that if $x \cdot c \in T$, where $x \in D^*$ and $c \in D$, then also $x \in T$. If $T = D^*$, we say that T is a full D-tree. The elements of T are called *nodes*, and the empty word ε is the *root* of T. For every $x \in T$, the nodes $x \cdot c$, for $c \in D$, are the *successors* of x. Given an alphabet Σ, a *Σ-labeled D-tree* is a pair $\langle T, \tau\rangle$ where T is a tree and $\tau : T \to \Sigma$ maps each node of T to a letter in Σ.

We denote classes of automata by acronyms in $\{D, N\} \times \{B, C, P\} \times \{W, T\}$. The first letter stands for the branching mode of the automaton (deterministic or nondeterministic); the second letter stands for the acceptance-condition type (Büchi, co-Büchi, or parity); the third letter stands for the object over which the automaton runs (words or trees). For example, NBW stands for nondeterministic Büchi automata, and DPT stands for deterministic parity tree automata.

3 Algorithms

The traditional algorithm for solving the realizability problem translates the LTL formula into an NBW, applies Safra's construction in order to get a DPW $\mathcal{A}_\psi$ for it, expands $\mathcal{A}_\psi$ to a DPT $\mathcal{A}_{\forall\psi}$ that accepts all the trees all of whose branches satisfy ψ, and then checks the nonemptiness of $\mathcal{A}_{\forall\psi}$ with respect to *I-exhaustive* $2^{I\cup O}$-labeled 2^I-trees, namely $2^{I\cup O}$-labeled 2^I-trees that contain, for each word $w \in (2^I)^\omega$, at least one path whose projection on 2^I is w [44]. Thus, the algorithm applies Safra's determinization construction, and has to solve the nonemptiness problem for DPT. For ψ of length n, the DPW $\mathcal{A}_\psi$ has $2^{2^{O(n \log n)}}$ states and index $2^{O(n)}$. This is also the size of the DPT $\mathcal{A}_{\forall\psi}$, making the overall complexity doubly-exponential, which matches the lower bound in [49].

In [33], we describe a "Safraless" synthesis algorithm that avoids determinization and avoids the use of the parity acceptance condition. The algorithm proceeds as follows. A strategy $f : (2^I)^* \to 2^O$ can be viewed as a 2^O-labeled 2^I-tree. We define a

UCT $\mathcal{S}_\psi$ such that $\mathcal{S}_\psi$ accepts a 2^O-labeled 2^I-tree $\langle T, \tau\rangle$ iff τ is a good strategy for ψ. We define $\mathcal{S}_\psi$ as follows.

Let $\mathcal{A}_{\neg\psi} = \langle 2^{I\cup O}, Q, q_{in}, \delta, \alpha\rangle$ be an NBW for $\neg\psi$ [55]. Thus, $\mathcal{A}_{\neg\psi}$ accepts exactly all the words in $(2^{I\cup O})^\omega$ that do not satisfy ψ. Then, $\mathcal{S}_\psi = \langle 2^O, 2^I, Q, q_{in}, \delta', \alpha\rangle$, where for every $q \in Q$ and $o \in 2^O$, we have $\delta'(q, o) = \bigwedge_{i\in 2^I} \bigwedge_{q'\in\delta(q,i\cup o)} (i, q')$. Thus, from state q, reading the output assignment $o \in 2^O$, the automaton $\mathcal{S}_\psi$ branches to each direction $i \in 2^I$, with all the states q' to which δ branches when it reads $i \cup o$ in state q. It is not hard to see that $\mathcal{S}_\psi$ accepts a 2^O-labeled 2^I-tree $\langle T, \tau\rangle$ iff for all the paths $\{\varepsilon, i_0, i_0 \cdot i_1, i_0 \cdot i_1 \cdot i_2, \ldots\}$ of T, the infinite word $(i_0 \cup \tau(\varepsilon)), (i_1 \cup \tau(i_0)), (i_2 \cup \tau(i_0 \cdot i_1)), \ldots$ is not accepted by $\mathcal{A}_{\neg\psi}$; thus all the computations generated by τ satisfy ψ. The size of $\mathcal{A}_{\neg\psi}$ is exponential in the length of ψ. Using the rank-based method of [33], it is possible to reduce the nonemptiness of a UCT to the nonemptiness of an NBT with another exponential blow-up. Hence, realizability of ψ is reduced to checking the nonemptiness of an NBT of size doubly exponential in the length of ψ. Since the nonemptiness of an NBT can be solved in quadratic time, we are done. Moreover, as with the traditional algorithm [46], the approach in [33] can return a witness to the nonemptiness of $\mathcal{S}_\psi$ – a transducer that realizes ψ. Thus we solve both realizability and synthesis.

The approach in [33] was improved in [31], where the algorithm starts by translating $\neg\psi$ to a nondeterministic generalized Büchi automaton. The Safraless approach is used in algorithms for bounded synthesis [15,52], was extended to timed specifications [21], led to further simplified synthesis algorithms [19], and was implemented in [25].

4 Methodology

A serious drawback of current synthesis algorithms is that they assume a comprehensive set of temporal assertions, describing the global behavior of the system, as a starting point. The ultimate goal in compositional synthesis is to compose a system that realizes a set of specifications from systems that realize the underlying specifications. Moreover, in case the conjunction of specifications is not realizable, the user can omit or weaken some of them.

In [31], we describe how it is possible, when we check the realizability of $\psi \wedge \psi'$, to use much of the work done in checking the realizability of ψ and ψ' in isolation. Recall that the Safraless algorithm reduces realizability of a specification ψ to the nonemptiness of an NBT $\mathcal{U}_\psi$. The state space of $\mathcal{U}_\psi$ is simple: each state of $\mathcal{U}_\psi$ is of the form $\langle S, g\rangle$, where S is a set of states in the intermediate UCT $\mathcal{S}_\psi$ (the "state component", which coincides with the set of states of $\mathcal{A}_{\neg\psi}$) and g is a function that maps the states in S to a finite set of ranks (the "ranking component"). Realizability of $\psi \wedge \psi'$ then involves the product of $\mathcal{U}_\psi$ and $\mathcal{U}_{\psi'}$. The simple structure of the NBTs makes it possible not only to define the product easily (ease follows also from the use of a generalized Büchi acceptance condition), but also to use the work done during the nonemptiness checks of $\mathcal{U}_\psi$ and $\mathcal{U}_{\psi'}$ when we check the nonemptiness of the product. Indeed, the rank component of a state in the product describes ranks to states in both $\mathcal{S}_\psi$ and $\mathcal{S}_{\psi'}$. A state whose projection on one of the components corresponds to a state that is empty in the NBT for this component, can be labeled empty. This approach is especially helpful when combined with an incremental approach, where we construct the NBT with a

small maximal rank, and increase the maximal rank only if the specification turns out not to be realizable with this small maximal rank,

So far, we considered compositionality in terms of the specification. In practice, the compositionality of the specification is often reflected also in a hierarchical structure, where subformulas of the specifications are composed not only in a Boolean manner but also in a modular one. Another way to view this is as follows. In the classical synthesis algorithms, it is always assumed the system is "constructed from scratch" rather than "composed" from reusable components. This rarely happens in real life. In real life, almost every non-trivial commercial system, either hardware or software, relies heavily on using libraries of reusable components. Furthermore, other contexts, such as web service orchestration, can be modeled as synthesis of a system from a library of components.

In [37], the authors define and study the problem of LTL synthesis from libraries of reusable components. Two notions of composition are defined: functional composition, for which the problem turns out not be decidable, and structural composition, for which the problem is 2EXPTIME-complete. As a side benefit, [37] derives an explicit characterization of the information needed by the synthesizer on the underlying components. This characterization can be used as a specification formalism between component providers and integrators. Synthesis from underlying components is extended in [36] to consider the problem of control-flow synthesis from libraries of probabilistic components. It is shown that this more general problem is also decidable.

5 Scope

One approach to tackle the algorithmic difficulties in LTL synthesis has been to restrict the class of allowed specification. In [5], the authors study the case where the LTL formulas are of the form $\mathbf{G}p$, $\mathbf{F}p$, $\mathbf{GF}p$, or $\mathbf{FG}p$.[2] In [3], the authors consider the fragment of LTL consisting of boolean combinations of formulas of the form $\mathbf{G}p$, as well as a richer fragment in which the $\mathbf{N}$ operator is allowed. Since the games corresponding to formulas of these restricted fragments are simpler, the synthesis problem is simpler too, and it can be solved in PSPACE or EXPSPACE, depending on the specific fragment. Anther fragment of LTL, termed $GR(1)$, is studied in [42]. In the $GR(1)$ fragment (generalized reactivity(1)) the formulas are of the form $(\mathbf{GF}p_1 \wedge \mathbf{GF}p_2 \wedge \cdots \mathbf{GF}p_m) \rightarrow (\mathbf{GF}q_1 \wedge \mathbf{GF}q_2 \wedge \cdots \mathbf{GF}q_n)$, where each p_i and q_i is a Boolean combination of atomic propositions. It is shown in [42] that for this fragment, the synthesis problem can be solved in EXPTIME, and with only $O((mn \cdot 2^{|AP|})^3)$ symbolic operations, where AP is the set of atomic propositions.

In [34], we continue this approach with the aim of focusing on properties that are used in practice. We study the synthesis problem for TRIGGER LOGIC. Modern industrial-strength property-specification languages such as Sugar [6], ForSpec [4], and the recent standards PSL [16] and SVA [56] include regular expressions. TRIGGER LOGIC is a fragment of these logics that covers most of the properties used in practice by system designers. Technically, TRIGGER LOGIC consists of positive Boolean

[2] The setting in [5] is of real-time games, which generalizes synthesis.

combinations of assertions about regular events, connected by the usual regular operators as well as temporal implication, $\mapsto$ ("triggers"). For example, the TRIGGER LOGIC formula $(true[*]; req; ack) \mapsto (true[*]; grant)$ holds in an infinite computation if every request that is immediately followed by an acknowledge is eventually followed by a grant. Also, the TRIGGER LOGIC formula $(true[*]; err) \mapsto avoid(true[*]; ack)$ holds in a computation if once an error is detected, no acks can be sent.

It is shown in [34] that TRIGGER LOGIC formulas can be translated to deterministic Büchi automata using the two classical subset constructions: the determinization construction of [48] and the break-point construction of [39]. Accordingly, while the synthesis problem for TRIGGER LOGIC is still 2EXPTIME-complete, the synthesis algorithm for it is significantly simpler than the one used in general temporal synthesis.

The work described above stays in the traditional setting of synthesis and considers variants of the specification formalism. An even more interesting extension of the scope of synthesis refers to the underlying setting. It is by now realized that requiring the synthesized system to satisfy the specification against all possible environments is often too demanding. Dually, allowing all possible systems is perhaps not demanding enough. This issue is traditionally approached by adding assumptions on the system and/or the environment, which are modeled as part of the specification (c.f., [11]).

In [30,52], the authors study study *bounded temporal synthesis*, in which bounds on the sizes of the state space of the system and the environment are additional parameters to the synthesis problem. The study is motivated by the fact that such bounds may indeed change the answer to the synthesis problem, as well as the theoretical and computational aspects of the synthesis problem. In particular, a finer analysis of synthesis, which takes system and environment sizes into account, yields deeper insight into the quantificational structure of the synthesis problem and the relationship between strong synthesis – there exists a system such that for all environments, the specification holds, and weak synthesis – for all environments there exists a system such that the specification holds.

Unlike the unbounded setting, where determinacy of regular games implies that strong and weak synthesis coincide, these notions do not coincide in the bounded setting. Bounding the size of the system or the environment turns the synthesis problem into a search problem, and one cannot expect to do better than brute-force search. In particular, the synthesis problem for bounded environments is NP-complete [52], and is Σ_2^P-complete for bonded systems and environments (the complexity is in terms of the bounds, for a specification given by a deterministic automaton) [30].

A different, more conceptual change of the setting has to do with the fact that modern systems often interact with other systems. For example, the clients interacting with a server are by themselves distinct entities (which we call agents) and are many times implemented by systems. In the traditional approach to synthesis, the way in which the environment is composed of its underlying agents is abstracted. In particular, the agents can be seen as if their only objective is to conspire to fail the system. Hence the term "hostile environment" that is traditionally used in the context of synthesis. In real life, however, many times agents have goals of their own, other than to fail the system. The approach taken in the field of algorithmic game theory [40] is to assume that agents interacting with a computational system are *rational*, i.e., agents act to achieve their

own goals. Assuming agents rationality is a restriction on the agents behavior and is therefore equivalent to restricting the universal quantification on the environment.

In [20], we introduce the problem of synthesis in the context of rational agents (*rational synthesis*, for short). The input consists of a temporal-logic formula specifying the system, temporal-logic formulas specifying the objectives of the agents, and a solution concept definition. The output is an implementation T of the system and a profile of strategies, suggesting a behavior for each of the agents. The output should satisfy two conditions. First, the composition of T with the strategy profile should satisfy the specification. Second, the strategy profile should be an equilibrium in the sense that, in view of their objectives, agents have no incentive to deviate from the strategies assigned to them, where "no incentive to deviate" is interpreted as dictated by the given solution concept. As it turns out, system synthesizers can capitalize on the rationality and goals of the agents interacting with it. Moreover, for the classical definitions of equilibria studied in game theory, rational synthesis is not harder than traditional synthesis. The results in [20] also consider the multi-valued case in which the objectives of the system and the agents are still temporal logic formulas, but involve payoffs from a finite lattice.

6 Quality

Very little attention has been payed to the quality of the systems that are automatically synthesized. Typically, many systems satisfy a realizable specification, and while they all satisfy the specification, they may do so at different levels of "unspecified quality". This latter problem is a real obstacle, as designers would be willing to give up manual design only after being convinced that the automatic procedure that replaces it generates systems of comparable quality. Nowadays specification formalisms are too abstract to specify such quality measures.

An approach in which quantitative reasoning is used in order to improve the quality of automatically synthesized systems is described in [7]. There, the synthesis problem is reduced to the solution of lexicographic mean-payoff games. A winning strategy in the game induces a system that satisfies the specification in high quality. Another approach is described in [27,28], where the authors introduce the idea of model-checking-based genetic programming as a general approach to synthesis: starting with a randomly generated solution, the solution is iteratively improved according to a valuation (fitness function) that a model checker assigns to the suggested solution.

The neglecting of the quality of automatically synthesized systems is related to the fact that model checking is Boolean. The Boolean fate of a verification process seems natural, as a system can either satisfy its specification or not satisfy it. The richness of today's systems and verification methodologies has motivated the introduction of *multi-valued* specification formalisms. These formalisms are used in order to specify quantitative properties (say, map a computation to the maximal wait time along it) [9,10,23], or in order to specify rich truth values (say, an "unknown" value, in the case of abstraction [50], or a value that is a subset of all possible viewpoints, in case of systems with multiple viewpoints [24]). Very little attempts, however, have been made to augment temporal logics with a quantitative layer that would enable the specification of the relative merits of different aspects of the specification. The idea behind such a

layer is that there should be a difference between a system that satisfies the specification $\mathbf{AG}(req \rightarrow \mathbf{F}grant)$ with grants generated soon after requests, and a system that satisfies it with grants generated after long waits. Specification formalisms should be refined in order to reveal such differences, and algorithms for reasoning about the new formalisms are required. With the right formalisms, a designer will be able to associate the different aspects of the specification with costs and rewards, reflecting their importance, and synthesis algorithms will be able to generate systems that not only satisfy the specification, but also do so in a way that maximizes the quality of the satisfaction, as defined formally by the designer. The development of such multi-valued formalisms and algorithms for reasoning about them involves is the subject of current research.

References

1. Abadi, M., Lamport, L., Wolper, P.: Realizable and Unrealizable Concurrent Program Specifications. In: Ronchi Della Rocca, S., Ausiello, G., Dezani-Ciancaglini, M. (eds.) ICALP 1989. LNCS, vol. 372, pp. 1–17. Springer, Heidelberg (1989)
2. Althoff, C.S., Thomas, W., Wallmeier, N.: Observations on Determinization of Büchi Automata. In: Farré, J., Litovsky, I., Schmitz, S. (eds.) CIAA 2005. LNCS, vol. 3845, pp. 262–272. Springer, Heidelberg (2006)
3. Alur, R., La Torre, S.: Deterministic generators and games for LTL fragments. ACM Transactions on Computational Logic 5(1), 1–25 (2004)
4. Armoni, R., Fix, L., Flaisher, A., Gerth, R., Ginsburg, B., Kanza, T., Landver, A., Mador-Haim, S., Singerman, E., Tiemeyer, A., Vardi, M.Y., Zbar, Y.: The forSpec Temporal Logic: A New Temporal Property-Specification Language. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 196–211. Springer, Heidelberg (2002)
5. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. In: IFAC Symposium on System Structure and Control, pp. 469–474. Elsevier (1998)
6. Beer, I., Ben-David, S., Eisner, C., Fisman, D., Gringauze, A., Rodeh, Y.: The Temporal Logic Sugar. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 363–367. Springer, Heidelberg (2001)
7. Bloem, R., Chatterjee, K., Henzinger, T.A., Jobstmann, B.: Better Quality in Synthesis Through Quantitative Objectives. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 140–156. Springer, Heidelberg (2009)
8. Büchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. Trans. AMS 138, 295–311 (1969)
9. Chakrabarti, A., Chatterjee, K., Henzinger, T.A., Kupferman, O., Majumdar, R.: Verifying Quantitative Properties Using Bound Functions. In: Borrione, D., Paul, W. (eds.) CHARME 2005. LNCS, vol. 3725, pp. 50–64. Springer, Heidelberg (2005)
10. Chatterjee, K., Doyen, L., Henzinger, T.A.: Quantitative Languages. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 385–400. Springer, Heidelberg (2008)
11. Chatterjee, K., Henzinger, T.A., Jobstmann, B.: Environment Assumptions for Synthesis. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 147–161. Springer, Heidelberg (2008)
12. Church, A.: Logic, arithmetics, and automata. In: Proc. Int. Congress of Mathematicians, 1962, pp. 23–35. Institut Mittag-Leffler (1963)
13. Clarke, E.M., Grumberg, O., Peled, D.: Model Checking. MIT Press (1999)
14. Dill, D.L.: Trace theory for automatic hierarchical verification of speed independent circuits. MIT Press (1989)

15. Ehlers, R.: Symbolic Bounded Synthesis. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 365–379. Springer, Heidelberg (2010)
16. Eisner, C., Fisman, D.: A Practical Introduction to PSL. Springer, Heidelberg (2006)
17. Elgaard, J., Klarlund, N., Möller, A.: Mona 1.x: new techniques for WS1S and WS2S. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 516–520. Springer, Heidelberg (1998)
18. Emerson, E.A., Clarke, E.M.: Using branching time logic to synthesize synchronization skeletons. Science of Computer Programming 2, 241–266 (1982)
19. Filiot, E., Jin, N., Raskin, J.-F.: An Antichain Algorithm for LTL Realizability. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 263–277. Springer, Heidelberg (2009)
20. Fisman, D., Kupferman, O., Lustig, Y.: Rational Synthesis. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 190–204. Springer, Heidelberg (2010)
21. Di Giampaolo, B., Geeraerts, G., Raskin, J.-F., Sznajder, N.: Safraless Procedures for Timed Specifications. In: Chatterjee, K., Henzinger, T.A. (eds.) FORMATS 2010. LNCS, vol. 6246, pp. 2–22. Springer, Heidelberg (2010)
22. Hare, D., Pnueli, A.: On the development of reactive systems. In: Apt, K. (ed.) Logics and Models of Concurrent Systems. NATO Advanced Summer Institutes, vol. F-13, pp. 477–498. Springer, Heidelberg (1985)
23. Henzinger, T.A.: From Boolean to quantitative notions of correctness. In: Proc. 37th ACM Symp. on Principles of Programming Languages, pp. 157–158 (2010)
24. Hussain, A., Huth, M.: On model checking multiple hybrid views. Technical Report TR-2004-6, University of Cyprus (2004)
25. Jobstmann, B., Bloem, R.: Game-based and simulation-based improvements for LTL synthesis. In: 3nd Workshop on Games in Design and Verification (2006)
26. Jurdziński, M.: Small Progress Measures for Solving Parity Games. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 290–301. Springer, Heidelberg (2000)
27. Katz, G., Peled, D.: Genetic Programming and Model Checking: Synthesizing New Mutual Exclusion Algorithms. In: Cha, S., Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) ATVA 2008. LNCS, vol. 5311, pp. 33–47. Springer, Heidelberg (2008)
28. Katz, G., Peled, D.: Model Checking-Based Genetic Programming with an Application to Mutual Exclusion. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 141–156. Springer, Heidelberg (2008)
29. Klarlund, N.: Mona & Fido: The Logic-Automaton Connection in Practice. In: Nielsen, M. (ed.) CSL 1997. LNCS, vol. 1414, pp. 311–326. Springer, Heidelberg (1998)
30. Kupferman, O., Lustig, Y., Vardi, M.Y., Yannakakis, M.: Temporal synthesis for bounded systems and environments. In: Proc. 28th Symp. on Theoretical Aspects of Computer Science, pp. 615–626 (2011)
31. Kupferman, O., Piterman, N., Vardi, M.Y.: Safraless Compositional Synthesis. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 31–44. Springer, Heidelberg (2006)
32. Kupferman, O., Vardi, M.Y.: Synthesis with incomplete information. In: Advances in Temporal Logic, pp. 109–127. Kluwer Academic Publishers (2000)
33. Kupferman, O., Vardi, M.Y.: Safraless decision procedures. In: Proc. 46th IEEE Symp. on Foundations of Computer Science, pp. 531–540 (2005)
34. Kupferman, O., Vardi, M.Y.: Synthesis of Trigger Properties. In: Clarke, E.M., Voronkov, A. (eds.) LPAR-16 2010. LNCS, vol. 6355, pp. 312–331. Springer, Heidelberg (2010)
35. Lichtenstein, O., Pnueli, A.: Checking that finite state concurrent programs satisfy their linear specification. In: Proc. 12th ACM Symp. on Principles of Programming Languages, pp. 97–107 (1985)
36. Lustig, Y., Nain, S., Vardi, M.Y.: Synthesis from probabilistic components. In: Proc. 20th Annual Conf. of the European Association for Computer Science Logic, pp. 412–427 (2011)
37. Lustig, Y., Vardi, M.Y.: Synthesis from Component Libraries. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 395–409. Springer, Heidelberg (2009)

38. Manna, Z., Wolper, P.: Synthesis of communicating processes from temporal logic specifications. ACM Transactions on Programming Languagues and Systems 6(1), 68–93 (1984)
39. Miyano, S., Hayashi, T.: Alternating finite automata on ω-words. Theoretical Computer Science 32, 321–330 (1984)
40. Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V.V.: Algorithmic Game Theory. Cambridge University Press (2007)
41. Piterman, N.: From nondeterministic Büchi and Streett automata to deterministic parity automata. In: Proc. 21st IEEE Symp. on Logic in Computer Science, pp. 255–264. IEEE press (2006)
42. Piterman, N., Pnueli, A., Saar, Y.: Synthesis of Reactive(1) Designs. In: Emerson, E.A., Namjoshi, K.S. (eds.) VMCAI 2006. LNCS, vol. 3855, pp. 364–380. Springer, Heidelberg (2005)
43. Pnueli, A.: The temporal semantics of concurrent programs. Theoretical Computer Science 13, 45–60 (1981)
44. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: Proc. 16th ACM Symp. on Principles of Programming Languages, pp. 179–190 (1989)
45. Pnueli, A., Rosner, R.: On the Synthesis of an Asynchronous Reactive Module. In: Ronchi Della Rocca, S., Ausiello, G., Dezani-Ciancaglini, M. (eds.) ICALP 1989. LNCS, vol. 372, pp. 652–671. Springer, Heidelberg (1989)
46. Rabin, M.O.: Weakly definable relations and special automata. In: Proc. Symp. Math. Logic and Foundations of Set Theory, pp. 1–23. North-Holland (1970)
47. Rabin, M.O.: Automata on infinite objects and Church's problem. Amer. Mathematical Society (1972)
48. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM Journal of Research and Development 3, 115–125 (1959)
49. Rosner, R.: Modular Synthesis of Reactive Systems. PhD thesis, Weizmann Institute of Science (1992)
50. Graf, S., Saidi, H.: Construction of Abstract State Graphs with PVS. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 72–83. Springer, Heidelberg (1997)
51. Safra, S.: On the complexity of ω-automata. In: Proc. 29th IEEE Symp. on Foundations of Computer Science, pp. 319–327 (1988)
52. Schewe, S., Finkbeiner, B.: Bounded Synthesis. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 474–488. Springer, Heidelberg (2007)
53. Tasiran, S., Hojati, R., Brayton, R.K.: Language Containment Using Non-Deterministic Omega-Automata. In: Camurati, P.E., Eveking, H. (eds.) CHARME 1995. LNCS, vol. 987, pp. 261–277. Springer, Heidelberg (1995)
54. Vard, M.Y.: An Automata-Theoretic Approach to Fair Realizability and Synthesis. In: Wolper, P. (ed.) CAV 1995. LNCS, vol. 939, pp. 267–292. Springer, Heidelberg (1995)
55. Vard, M.Y., Wolper, P.: Reasoning about infinite computations. Information and Computation 115(1), 1–37 (1994)
56. Vijayaraghavan, S., Ramanathan, M.: A Practical Guide for SystemVerilog Assertions. Springer, Heidelberg (2005)
57. Wong-Toi, H., Dill, D.L.: Synthesizing processes and schedulers from temporal specifications. In: Proc. 2nd Conf. on Computer Aided Verification. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 3, pp. 177–186. AMS (1991)

Cryptography from Learning Parity with Noise*

Krzysztof Pietrzak

Institute of Science and Technology (IST) Austria

Abstract. The Learning Parity with Noise (LPN) problem has recently found many applications in cryptography as the hardness assumption underlying the constructions of "provably secure" cryptographic schemes like encryption or authentication protocols. Being provably secure means that the scheme comes with a proof showing that the existence of an efficient adversary against the scheme implies that the underlying hardness assumption is wrong.

LPN based schemes are appealing for theoretical and practical reasons. On the theoretical side, LPN based schemes offer a very strong security guarantee. The LPN problem is equivalent to the problem of decoding random linear codes, a problem that has been extensively studied in the last half century. The fastest known algorithms run in exponential time and unlike most number-theoretic problems used in cryptography, the LPN problem does not succumb to known quantum algorithms. On the practical side, LPN based schemes are often extremely simple and efficient in terms of code-size as well as time and space requirements. This makes them prime candidates for light-weight devices like RFID tags, which are too weak to implement standard cryptographic primitives like the AES block-cipher.

This talk will be a gentle introduction to provable security using simple LPN based schemes as examples. Starting from pseudorandom generators and symmetric key encryption, over secret-key authentication protocols, and, if time admits, touching on recent constructions of public-key identification, commitments and zero-knowledge proofs.

1 Learning Parity with Noise and Related Problems

The *search* version of the learning parity with noise problem with parameters $\ell \in \mathbb{N}$ (the length of the secret), $\tau \in \mathbb{R}$ where $0 < \tau < 0.5$ (the noise rate) and $q \in \mathbb{N}$ (the numbers of samples) asks to find a fixed random ℓ bit secret $\mathbf{s} \in \mathbb{Z}_2^\ell$ from q samples of the form $\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle \oplus e$ where $\mathbf{a} \in \mathbb{Z}_2^\ell$ is random and $e \in \mathbb{Z}_2$ has Bernoulli distribution with parameter τ (we denote this distribution with Ber_τ), i.e. $\Pr[e = 1] = \tau$. The *decisional* LPN problem is defined similarly, except that we require that one cannot even distinguish noisy inner products from random. The distinction between the search and the decisional version of a problem is often made for problems used in cryptography. Typically, assuming the decisional version of a problem allows for much simpler and more efficient constructions

* This survey paper accompanies an invited talk at SOFSEM 2012.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 99–114, 2012.

of cryptosystems, whereas the search version is a weaker assumption and thus constructions based on it require less "faith" in the presumed hardness of the assumption.[1] Interestingly, for the LPN problem one can show that the distinction between the search and the decisional version is irrelevant, more on this below. Before we formally define the LPN problem, let us set the notational conventions for the rest of this paper.

Notation. $\mathbb{Z}_q$ denotes the set $\{0, 1, \ldots, q-1\}$, and addition is always modulo q. In particular, $\mathbb{Z}_2 = \{0, 1\}$ are bits and $\oplus$ denotes bitwise XOR. We use bold small and capital letters like $\mathbf{x}, \mathbf{X}$ to denote vectors and matrices, respectively. Calligraphic letters like $\mathcal{X}$ denote sets. For a set $\mathcal{X}$, $x \xleftarrow{\$} \mathcal{X}$ denotes that x is assigned a value sampled uniformly at random from $\mathcal{X}$. For a distribution D, $x \leftarrow D$ denotes that x is sampled according to D. With Ber_τ we denote the Bernoulli distribution with parameter τ, i.e. $\Pr[x = 1 \ ; \ x \leftarrow \mathsf{Ber}_\tau] = \tau$. For $m \in \mathbb{N}$, U_m denotes the uniform distribution over $\mathbb{Z}_2^m$. $X \sim D$ denotes that X is a random variable with distribution D. $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{n} \mathbf{a}[i] \cdot \mathbf{b}[i] \bmod p$ denotes the inner product of $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n$.

The Basic LPN Problem

Definition 1 (search/decisional LPN Problem). *For $\tau \in]0, 1/2[$, $\ell \in \mathbb{N}$, the* decisional $\mathsf{LPN}_{\tau,\ell}$ *problem is (q, t, ϵ)-hard if for every distinguisher* D *running in time t*

$$\left| \Pr_{\mathbf{s},\mathbf{A},\mathbf{e}}[\mathsf{D}(\mathbf{A}, \mathbf{A}.\mathbf{s} \oplus \mathbf{e}) = 1] - \Pr_{\mathbf{r},\mathbf{A}}[\mathsf{D}(\mathbf{A}, \mathbf{r}) = 1] \right| \leq \epsilon \tag{1}$$

Where $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_2^\ell$, $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_2^{q \times \ell}$, $\mathbf{e} \leftarrow \mathsf{Ber}_\tau^q$ and $\mathbf{r} \xleftarrow{\$} \mathbb{Z}_2^q$. The search $\mathsf{LPN}_{\tau,\ell}$ problem is (q, t, ϵ)-hard if for every D running in time t

$$\Pr_{\mathbf{s},\mathbf{A},\mathbf{e}}[\mathsf{D}(\mathbf{A}, \mathbf{A}.\mathbf{s} \oplus \mathbf{e}) = \mathbf{s}] \leq \epsilon \tag{2}$$

The Learning with Errors (LWE) Problem. A problem closely related to LPN is the learning with errors (LWE) problem introduced by Regev [43]. LWE is a generalization of LPN to larger moduli. For some prime p,[2] we have a secret $\mathbf{s} \in \mathbb{Z}_p^\ell$, and the adversary is asked to find $\mathbf{s}$ given samples $\langle \mathbf{a}, \mathbf{s} \rangle + e$. Here $\mathbf{a}$ is uniform in $\mathbb{Z}_p^\ell$ and the noise $e \in \mathbb{Z}_p$ is sampled according to some distribution χ, typically this distribution is a "discrete Gaussian". A good survey paper on LWE and its applications is [44].[3] LWE seems much more versatile than LPN. Besides all the cryptographic primitives we can construct from LPN, there are

[1] A typical example is public-key encryption based on the Diffie-Hellman problem, which is quite straight forward and efficient using the decisional version of the problem [14], but much more tricky and less practical using the search version [11].

[2] The case where the moduli is a the power of a prime has also been used [2].

[3] A bibliography of LWE (and more generally, lattice) based cryptosystems is maintained on http://xagawa.net/bib-lattice/

contsructions of much more sophisticated objects like public-key encryption [43] (even fully homomorphic [16] or identity-based [21,10]) and collision resistant hash functions [33], which we do not know how to construct from LPN.[4] LWE is also interesting for theoretical reasons, as it has the remarkable property that its hardness follows form *worst case* hardness of lattice assumptions [43,40]. LWE lacks the simplicity of LPN,[5] and thus LWE based schemes are less suited for weak devices like RFID tags.

Decision vs. Search. In contrast to most cryptographic assumptions which come in a search and decisional variant, it turns out that for LPN the two versions are "polynomially equivalent" [5,31] as stated in the lemma below.[6] This means that any attacker of size t against decisional LPN implies an attacker of size $poly(t)$ against the search version. Thus, cryptosystems proven secure under the decisional LPN assumption are already secure if search LPN is hard. Although this search to decision reduction is not tight, in practice we have no faster algorithms for decision than for search.

Lemma 1 (Lemma 1 from [31]). *If decisional* $\mathsf{LPN}_{\tau,\ell}$ *is not* (q, t, ε) *secure, then search* $\mathsf{LPN}_{\tau,\ell}$ *is not* $O(q', t', \varepsilon')$ *secure where*

$$q' = O(q \cdot \log \ell/\varepsilon^2) \qquad t' = O(t \cdot \ell \cdot \log \ell/\varepsilon^2) \qquad \varepsilon' = \varepsilon/4$$

Relations of LPN to Other Problems. With the current state in complexity theory, we cannot expect to prove that there exists no efficient adversary who breaks the LPN problem, as this would imply $\mathcal{P} \neq \mathcal{NP}$. The search LPN problem can be stated as the $\mathcal{NP}$ complete problem of decoding random linear codes [7].[7] Think of $\mathbf{A}$ as the generator matrix and $\mathbf{s}$ as the message. The decoding problem

[4] Alekhnovich [4] and Applebaum, Barak and Wigderson [1] construct public-key encryption from variants of LPN (which seem like much stronger assumptions than LPN) where either the noise rate τ is not constant but depends on the length ℓ of the secret as $\tau = O(1/\sqrt{\ell})$ [4], or the vectors $\mathbf{a}$ are not uniform, but have Hamming weight exactly 3 [1]. Another approach, pioneered by McEliece [37], replaces the random $\mathbf{A}$ with a "disguised" generator matrix of a code which allows for efficient error-correction. The security of the public-key encryption scheme follows from LPN and the assumption that the disguised matrix is indistinguishable from uniformly random.

[5] It requires many multiplications modulo some prime p (Typically p is polynomial in a security parameter, and thus much smaller than the moduli used in discrete logarithm (or factoring based) based schemes, where $\log(p)$ must be at least as as large as the security parameter), as opposed to inner products of bit-vectors as for LPN.

[6] Such an equivalence also holds for LWE with prime modulus [43] or if the modulus is the power of a prime ([2], Lemma 1).

[7] This does not imply that LPN is hard assuming $\mathcal{P} \neq \mathcal{NP}$ as search LPN is an average case problem (we require that no efficient adversary succeeds with non-negligible probability), whereas $\mathcal{NP}$ hardness is just a worst case guarantee (no efficient adversary succeeds on all inputs), see [5] for a more in-depth discussion.

then asks to recover the message $\mathbf{s}$ from the noisy codeword $\mathbf{A}.\mathbf{s} \oplus \mathbf{e}$, which is exactly search LPN. The LPN problem has been extensively studied in learning theory, as an efficient algorithm for LPN would allow to learn several important concept classes like 2-DNF formulas, juntas, and any function with a sparse Fourier spectrum [15].

The best known algorithms to recover an ℓ bit secreet need $2^{\Theta(\ell/\log \ell)}$ time and samples [6,32]. If given only polynomially many $q = \mathsf{poly}(\ell)$ samples, the running time of the best algorithm goes up to $2^{\Theta(\ell/\log\log \ell)}$ [36], and given only linearly many samples $q = \Theta(\ell)$, the best algorithms run in exponential $2^{\Theta(\ell)}$ time [47,38]. Unlike most number-theoretic problems used in cryptography, no quantum algorithms for LPN are known which are significantly faster than the classical ones. See [32] for more exact estimates and suggestions of parameters ℓ, τ for cryptographic applications. In the next paragraphs we discuss the hardness of LPN when either the secret $\mathbf{s} \sim U_\ell$, the randomness $\mathbf{A} \sim U_{q\times\ell}$ or the error $\mathbf{e} \sim \mathsf{Ber}_\tau^q$ does not have the right distribution.

Hardness of LPN for Non-Uniform Secrets. The secret $\mathbf{s} \in \mathbb{Z}_2^\ell$ in the LPN problem is usually assumed to be uniformly random. It is not hard to see that this is the hardest distribution, in the sense that given an adversary D who finds a uniform $\mathbf{s}$ given $(\mathbf{A}, \mathbf{A}.\mathbf{s} \oplus \mathbf{e})$ with some probability δ, we can recover a $\hat{\mathbf{s}}$ with any distribution over $\mathbb{Z}_2^\ell$ from $(\mathbf{A}, \mathbf{A}.\hat{\mathbf{s}} \oplus \mathbf{e})$ with the same probability δ as follows: given $(\mathbf{A}, \mathbf{A}.\hat{\mathbf{s}} \oplus \mathbf{e})$, sample a uniform $\mathbf{s}'$ and invoke D on $(\mathbf{A}, \mathbf{A}.\hat{\mathbf{s}} \oplus \mathbf{e} \oplus \mathbf{A}.\mathbf{s}') = (\mathbf{A}, \mathbf{A}.(\hat{\mathbf{s}} \oplus \mathbf{s}') \oplus \mathbf{e})$. Note that $\hat{\mathbf{s}} \oplus \mathbf{s}'$ is uniform as required, and if D finds $\hat{\mathbf{s}} \oplus \mathbf{s}'$ (which happens with probability δ) we can recover $\hat{\mathbf{s}} = (\hat{\mathbf{s}} \oplus \mathbf{s}') \oplus \mathbf{s}'$.

Surprisingly, the uniform distribution is not the unique hardest distribution. Applebaum et al. ([2] , Lemma 2) show that the LWE problem (where the modulus p is prime or a power of a prime, and the noise distribution χ is arbitrary) is basically as hard if the secret $\mathbf{s} \leftarrow \chi^\ell$ is chosen according to the noise distribution and not uniform $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_p^\ell$. In particular, the LPN problem where $\mathbf{s} \leftarrow \mathsf{Ber}_\tau^\ell$ is as hard as for uniform $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_2^\ell$. In [2] this result is used to construct a key-dependent message secure public-key encryption scheme from LWE, and key-dependent secret-key encryption from LPN, we'll revisit the latter result later.

Motivated by applications to leakage-resilient cryptography, Goldwasser et al. [18] investigate the hardness of the LWE problem when the secret $\mathbf{s} \in \mathbb{Z}_p^\ell$ is not uniform, but is only known to have some min-entropy.[8] The best one can hope for is that LWE with secrets of min-entropy k is as hard as standard LWE with secrets of length $\ell' = k/\log(p)$.[9] In [18] it is shown that this is almost the case for LWE where the noise has a Gaussian distribution (which is the most

[8] X has min entropy k if $\Pr[X = x] \le 2^{-k}$ for every x in the support of X.

[9] The reason is that the particular distribution where the first ℓ' elements of $\mathbf{s}$ are uniform and the remaining ones are all zero has min-entropy $k = \ell' \log(p)$. LWE with such a secret is easily seen to be equivalent to LWE with a uniform secret in $\mathbb{Z}_p^{\ell'}$.

interesting case due to its equivalence to worst-case lattice problems.) They prove that if the standard LWE problem with uniform secrets over $\mathbb{Z}_p^{\ell'}$ (and noise distribution Gaussian with standard deviation α) is hard, then the (non-standard) LWE with secrets in $\mathbb{Z}_p^{\ell}$ having min-entropy $k = \ell' \log(p) + \omega(\log \ell)$ (and noise distribution Gaussian with standard deviation β) must also be hard. The reduction from [18] is not tight, but the main caveat is that it also blows up the noise distribution: the fraction of the deviations α/β must be negligible. For this reason, their result requires the modulus p to be at least superpolynomial, and in particular it implies nothing for the LPN problem where $p = 2$.

Nothing is known about the hardness of LPN for general distributions of high min-entropy. In the interesting special case where the secret $\mathbf{s} \in \mathbb{Z}_2^{\ell}$ is uniformly sampled from any $\ell' \leq \ell$ dimensional linear subspace (and thus has min-entropy ℓ'), the problem can be show to be exactly as hard as the standard LPN problem with a uniform ℓ' bit secret. This follows from the equivalence of LPN and the subspace LPN problem that we'll discuss below.

Hardness of LPN for Non-Uniform Noise. The LPN problem seems to remain hard, even if $\mathbf{s}$ and/or the rows of $\mathbf{A}$ are not uniform, but have sufficiently high min-entropy (if they are sampled from a linear subspace, this can even be proven.) Fiddling with the distribution of the error vector $\mathbf{e}$ is more delicate. If e.g. ℓ positions of $\mathbf{e}$ are fixed (or otherwise known to the adversary), she learns ℓ noiseless linear equations $\langle \mathbf{a}, \mathbf{s} \rangle = y$, and can compute $\mathbf{s}$ from this linear equations using normal Gaussian elimination. Arora and Ge [3] show an attack for the case where the bits of $\mathbf{e}$ are not i.i.d., but sampled as follows. For some n, the noise vector is sampled n bits at a time, where each such block is sampled independently at random, conditioned on having a 1 in exactly (or at most) $w = n.\tau$ positions. Although here each individual noise bit has distribution Ber_τ as required, the noise bits are not independent any more. Using a technique called linearization, [3] show that with this noise distribution one can recover the secret $\mathbf{s}$ in time roughly n^w.

Saving Public Randomness. The most expensive part in generating an LPN instance $(\mathbf{A}, \mathbf{A}.\mathbf{s} \oplus \mathbf{e})$ is the sampling of the random matrix $\mathbf{A} \in \mathbb{Z}_2^{q \times \ell}$. For some of the cryptosystems we'll discuss in Section 2 (namely for pseudorandom generators and commitment schemes) the fact that $\mathbf{A}$ is rather large and must be uniform will not be much of a problem: as $\mathbf{A}$ can be *public*, we can fix a public random $\mathbf{A}$ once and for all, and then use it to generate arbitrary many LPN instances. For other schemes (namely encryption and secret-key identification), the size of $\mathbf{A}$ is more of an issue, as here the secret $\mathbf{s}$ will play the role of a shared secret key, and thus is fixed. In order to generate new LPN instances (which is required to compute ciphertexts and during execution of the identification protocol), one must sample a fresh $\mathbf{A}$'s every time.

One way to leverage this problem is to use $n > 1$ independent random secrets, this will typically increase the size of the secret key by a factor of n, but decrease the cost due to the sampling, storing and/or sending $\mathbf{A}$ by the same factor as we can reuse each $\mathbf{A}$ n times.

It has also been suggested to not sample $\mathbf{A} \in \mathbb{Z}_2^{q \times \ell}$ uniformly at random, but from a distribution which allows a more succinct description of the samples. For example [23] suggest to use a random Toeplitz matrix, which requires only $q + \ell$ (as opposed $q \cdot \ell$) random bits. Such a matrix is sampled by choosing a random $\mathbf{a} \xleftarrow{\$} \mathbb{Z}_2^{q+\ell}$, then the i'th row of $\mathbf{A}$ is $\mathbf{a}[i \dots i + \ell]$.

Another variant, called Ring-LPN, has been suggested in [27]. Ring-LPN not only has a succinct description of $\mathbf{A}$, but also allows for extremely efficient evaluation of the matrix multiplication $\mathbf{A}.\mathbf{s}$ as it corresponds to a single multiplication of two polynomials. Ring-LPN is inspired by the Ring-LWE problem, strong evidence for the hardness of Ring-LWE is given in [34] who show it to be equivalent to hard problems on *ideal* lattices.

Subspace LPN. The subspace LPN problem [41] is a variant of the LPN problem where the adversary not only gets random samples $\langle \mathbf{a}, \mathbf{s} \rangle \oplus e$, but it is an interactive assumption where she can adaptively choose affine functions ϕ_a, ϕ_s and then gets samples $\langle \phi_a(\mathbf{a}), \phi_s(\mathbf{s}) \rangle \oplus e$. That is, $\mathbf{a}$ and $\mathbf{s}$ are first mapped to the linear subspaces defined by ϕ_a and ϕ_s, respectively, before the noisy inner product is computed.[10] If the adversary is restricted to choose mappings ϕ_a, ϕ_s that overlap in at least an ℓ' dimensional subspace,[11] then this problem is at most as hard as the LPN problem with secrets of length ℓ' (as one can map to a string which is all zero except for the first ℓ' bits.) In [41] it is shown that the other direction does also (almost) hold (this equivalence not only holds for LPN, but more generally for LWE using any prime modulus and any error distribution.) This equivalence has immediate consequences for several existing LPN and LWE based cryptosystems, as it implies much stronger security guarantees as anticipated by the designers of the schemes. For example security against related key attacks or security against "weak" randomness, cf. [41] for the details. The fact that subspace LPN is an interactive assumption, gives a powerful handle for constructing provably secure LPN based cryptosystems. In Section 2 we'll mention constructions of identification schemes and message authentication codes [30] whose proof heavily relies on this handle.

Exact LPN. Recall that the error vector $\mathbf{e} \in \mathbb{Z}_2^q$ of an $\mathsf{LPN}_{\tau,\ell}$ sample $(\mathbf{A}, \mathbf{A}.\mathbf{s} \oplus \mathbf{e})$ consists of q i.i.d. bits with distribution Ber_τ, and thus its expected Hamming weight (i.e. number of 1's) is $q\tau$. The Exact LPN (XLPN for short) problem is a minor variation of LPN where we require $\mathbf{e}$ to have Hamming weight *exactly* $\lceil q\tau \rfloor$. In the next section we'll mention some cryptosystems which rely on the search XLPN (a public-key identification scheme) and the decisional XLPN (efficient zero-knowledge proofs for linear functions of committed values.)

[10] The affine function $\phi_a : \mathbb{Z}_2^\ell \to \mathbb{Z}_2^\ell$ can be defined as $\mathbf{a} \to \mathbf{X}_a.\mathbf{a} \oplus \mathbf{x}_a$ for some matrix $\mathbf{X}_a \in \mathbb{Z}_2^{\ell \times \ell}$ (the linear part) and some vector $\mathbf{x}_a \in \mathbb{Z}_2^\ell$ (the affine part). Equivalently, ϕ_s can be defined by $\mathbf{X}_s, \mathbf{x}_s$.

[11] This means that $\mathbf{X}_a^T.\mathbf{X}_s$ has rank at least ℓ'.

Hardness of *search* XLPN trivially follows from standard LPN.[12] Unfortunately the proof of equivalence of search and decision for LPN does not work for the XLPN problem, and it is open if *decisional* XLPN is equivalent to LPN.[13]

A similar version of LPN where the Hamming weight is *at most* $\lceil q\tau \rfloor$ has been suggested by [31] as a means to get more efficient instantiations of LPN based cryptosystems.

2 Efficient LPN Based Cryptosystems

OWFs and Generic Constructions. A function $\mathsf{F} : \mathbb{Z}_2^n \to \mathbb{Z}_2^m$ is a one-way function (OWF), if it's hard to find a preimage for F for a random outputs. In asymptotic terms, we require that F runs in time $\mathsf{poly}(n)$, and for all D of size $\mathsf{poly}(n)$ we have

$$\Pr_{\mathbf{x} \xleftarrow{\$} \mathbb{Z}_2^n} [\mathsf{D}(\mathsf{F}(\mathbf{x})) = \mathbf{x}' \text{ where } \mathsf{F}(\mathbf{x}) = \mathsf{F}(\mathbf{x}')] = \mathsf{negl}(n)$$

One can construct a OWF from LPN.[14] From a OWF one can construct pseudorandom objects using generic constructions, like pseudorandom generators (PRG) [26], pseudorandom functions (PRF) [17] or pseudorandom permutations (PRP) [35]. The basic secret-key cryptographic tasks[15] like encryption and authentication are usually solved by using a PRP like the AES block-cipher. Constructions using this generic transformations would typically be too inefficient to compete with dedicated designs,[16] as a consequence, today basically all

[12] An $\mathsf{LPN}_{\tau,q}$ sample $(\mathbf{A}, \mathbf{A}.\mathbf{s} \oplus \mathbf{e})$ will satisfy $\|\mathbf{e}\|_1 = \lceil q\tau \rfloor$ with probability $\approx 1/\sqrt{q}$. This implies that an adversary who breaks the search XLPN problem (i.e. outputs the right $\mathbf{s}$) with probability δ, is also an adversary against the search LPN problem with advantage at least $\delta/\sqrt{q}$. To see this, note that conditioned on the LPN sample satisfying $\|\mathbf{e}\|_1 = \lceil q\tau \rfloor$, we have exactly the same distribution as in XLPN, and thus in this case the adversary will be successful with probability δ.

[13] It is important that in the search XLPN problem, the adversary only gets one sample $(\mathbf{A}, \mathbf{A}.\mathbf{s} \oplus \mathbf{e})$. If she can ask for polynomially many samples, then the search to decision reduction does work, but now it's open if this "many sample" version of search XLPN is equivalent to standard LPN.

[14] Fix some random $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_2^{q \times \ell}$. Now $\mathsf{F}(\mathbf{x})$ on input a (sufficiently long) string $\mathbf{x}$, uses $\mathbf{x}$ to sample $\mathbf{s} \sim U_\ell, \mathbf{e} \sim \mathsf{Ber}_\tau^q$. If the weight of $\mathbf{e}$ is unexpectedly high (say $\|\mathbf{e}\|_1 \geq q \cdot \frac{1/2+\tau}{2}$) we output a special symbol $\perp$ (using the Chernoff bound one can show that this happens with exponentially small probability.) Otherwise output $(\mathbf{A}, \mathbf{A}.\mathbf{s} \oplus \mathbf{e})$. As $(\mathbf{A}, \mathbf{A}.\mathbf{s} \oplus \mathbf{e})$ uniquely determines $\mathbf{s}$, any algorithm who finds a preimage must find (randomness used to sample) this unique $\mathbf{s}$, which would contradict the hardness of the search LPN problem.

[15] In the secret-key setting, the honest parties share a secret key not known to the adversary.

[16] We use the term "dedicated design" as opposed to constructions that come with a reductionist proof showing the construction is secure under some standard hardness assumption.

secret-key cryptography is done using a dedicated construction like AES as main ingredient.[17]

The fascinating thing about the LPN problem is that it gives rise to secret-key cryptosystems which not only are provably secure, but in some aspects can even outperform known dedicated constructions. Below we show how the search to decision equivalence for the LPN problem, discussed in the previous section, implies that plain LPN samples are already pseudorandom, and thus give an extremely simple and efficient PRG.

It is an open problem to get a PRF or even PRP from LPN which could compete with dedicated constructions. But simple and efficient schemes for encryption, identification and authentification *not going via a PRF or PRP construction*, do exist. We'll discuss some of them below.[18]

Pseudorandom Generators. A pseudorandom generator is a length increasing function $\mathsf{G} : \mathbb{Z}_2^n \to \mathbb{Z}_2^m$ where $\mathsf{G}(U_n)$ is pseudorandom, this means that no efficient distinguisher can distinguish $\mathsf{G}(U_n)$ from the uniform distribution U_m.

The definition of the decisional LPN problem (cf. eq.(1)) already implies that $\mathsf{LPN}_{\tau,\ell}$ samples $\mathbf{A}, \mathbf{A}.\mathbf{s} \oplus \mathbf{e}$ are pseudorandom, which gives rise to a simple construction of a PRG [5]: use the (uniformly random) input $\mathbf{r}$ to sample $\mathbf{A}, \mathbf{s}, \mathbf{e}$ and output $\mathbf{A}, \mathbf{A}.\mathbf{s} \oplus \mathbf{e}$. As observed by [2], we can fix $\mathbf{A}$ as a public parameter and need not to sample it, or include it in the output.

To show that this function $\mathbf{r} \to \mathbf{A}.\mathbf{s} \oplus \mathbf{e} \in \mathbb{Z}_2^q$ is a PRG, one also must show how to sample $\mathbf{s} \sim U_\ell$ and $\mathbf{e} \sim \mathsf{Ber}_\tau^q$ using a random seed $\mathbf{r}$ which is shorter than the q bit output (recall that a PRG must be length increasing.) This is possible for any $\tau < 0.5, \ell \in \mathbb{Z}$ and sufficiently large q: we need ℓ bits to sample the uniform $\mathbf{s} \sim U_\ell$. But each bit of $\mathbf{e}$ has only $h(\tau)$ bits of entropy (where $h(\tau) = -\tau \log \tau - (1-\tau)\log(1-\tau)$ denotes the binary entropy function.) Thus $(\mathbf{s}, \mathbf{e})$ has $\ell + qh(\tau)$ bits of entropy and can be sampled using roughly that many bits, and this can be done very efficiently (see [2] for details.)

For sufficiently large q we have $\ell + qh(\tau) < q$, thus the stretch (which denotes the number of bits the output is longer than the input) of this PRG is $(1 - h(\tau))q - \ell$. This is linear in the length $\approx \ell + qh(\tau)$ of the seed. Linear stretch is a desirable property of PRGs as the efficiency of constructions which use a PRG crucially depend on its stretch.

[2] suggest a variant of this construction where one uses several ℓ-bit keys simultaneously, let $\mathbf{S} \in \mathbb{Z}_2^{\ell \times n}$ denote n such keys arranged as the columns of a

[17] This contrasts with public-key cryptography, like public-key encryption schemes, which are usually required to be provably secure, possibly using some idealized assumptions like the random oracle model [9]. This is due to the fact that public-key encryption needs much more structure than in the secret-key setting, where one just has to garble enough, and this can be done in any (invertible) way (the art in designing block-ciphers is to do this garbling extremely efficient.)

[18] [8] construct low-depth PRFs from the LWE problem by using a generic transformation from synthesizers to PRF [39]. A synthesizers is a strong type of a PRGs which, informally, is secure even if used on inputs that are somehow correlated. It is not known how to construct efficient synthesizers from LPN.

matrix. For the right choice of $n = \mathsf{poly}(\ell)$ one can use fast matrix multiplication [12] to compute the pseudorandom output $\mathbf{A}.\mathbf{S} \oplus \mathbf{E}$ (with $\mathbf{E} \leftarrow \mathsf{Ber}_\tau^{q \times n}$), which gives a PRG that can be evaluated in time $\tilde{O}(qn)$, which is quasilinear in the seed length. This is an asymptotic running time and it's not clear if this is already useful for input sizes used in practice.[19]

Secret-Key Encryption. A simple encryption scheme from LPN was proposed by [24]. The encryption of a message $\mathbf{m}$ under the secret key $\mathbf{s} \sim U_\ell$ is $(\mathbf{A}, \mathbf{A}.\mathbf{s} \oplus \mathbf{e} \oplus \mathbf{G}.\mathbf{m})$. Here $\mathbf{A} \sim U_{q \times \ell}$ and $\mathbf{e} \sim \mathsf{Ber}_\tau^q$, and $\mathbf{G} \in \mathbb{Z}_2^{q \times \ell}$ is the generator matrix of an error correcting code $\mathbb{Z}_2^\ell \to \mathbb{Z}_2^q$ which allows for efficient correction of $\tau' q$ errors (for some $\tau' > \tau$). To decrypt a ciphertext $(\mathbf{A}, \mathbf{y})$, one computes $\mathbf{G}.\mathbf{m} \oplus \mathbf{e} = \mathbf{y} \oplus \mathbf{A}.\mathbf{s}$. From this noisy codeword $\mathbf{G}.\mathbf{m} \oplus \mathbf{e}$ one can recover the message $\mathbf{m}$ using the error correcting decoding procedure for the code $\mathbf{G}$ if $\|\mathbf{e}\|_1 \leq \tau' q$, which will be the case withe exponentially high probability. The security[20] of this scheme follows from the fact that under the decisional LPN assumption, $(\mathbf{A}, \mathbf{A}.\mathbf{s} \oplus \mathbf{e})$ is pseudorandom, which implies that also the ciphertext $(\mathbf{A}, \mathbf{A}.\mathbf{s} \oplus \mathbf{e} \oplus \mathbf{G}.\mathbf{m})$ is pseudorandom and thus hides the information of $\mathbf{m}$.

This scheme can not only be proven secure in the standard sense, but also provably satisfies some more exotic security notions. The equivalence of subspace LPN and LNP (discussed in the previous section) implies that this scheme is secure against so called *related key attacks* (RKA). More concretely, the adversary cannot only ask for encryptions under the key $\mathbf{s} \in \mathbb{Z}_2^\ell$, but also under keys $\phi(\mathbf{s})$ where $\phi(.)$ can be any adaptively chosen affine function (but the linear part must have sufficiently high rank $\ell' \leq \ell$, such that the LPN problem with secrets of length ℓ' is still hard.) In [2] it is shown that (a minor variant of) this scheme is secure under a large class of *key-dependent message attacks* (KDM). More concretely, the scheme remains secure even against adversaries who can ask for encryptions of any affine function (no restriction on the rank here) of the secret key.

Secret-Key Identification and Message Authentification. By far most research on LPN based cryptosystems has been published on secret-key identification protocols.[21] In such a protocol, a prover P exchanges messages with a verifier V.[22] P and V share a secret key. If V talks to the honest prover P, we require that finally V outputs accept. A typical application is access control, e.g. a wireless car key which has the role of the prover, and the car being the verifier.

There are several standard security definitions for identification protocols which try to capture the intuitive notion that an adversary not knowing the

[19] For the quasilinear running time, q and n must be in the order of ℓ^6, and thus the seed has size ℓ^{12}.

[20] More precisely, semantic security under chosen message attacks, which means no efficient adversary can distinguish encryptions any two different messages, even when given access to an encryption oracle.

[21] A list of relevant papers is on `http://www.ecrypt.eu.org/lightweight/index.php/HB`.

[22] In the context of RFID implementations, P is called the "tag" and V is the "reader".

secret key should not be able to make V accept. They differ in the power the adversary has before trying to launch such an impersonation attack.

In a *passive attack* the adversary can eavesdrop on several interactions between P and V, before trying to make V accept in a second phase. In an *active attack*, the adversary is additionally allowed to interact with P in the first phase. The strongest notion is a *man-in-the-middle attack* (MIM) where the adversary can arbitrarily interact with P and V (with polynomially many concurrent executions allowed) in the first phase.

Hopper and Blum [25] proposed the first LPN based identification scheme. Their goal was to design a scheme which is so simple that it could even be reliably executed by humans with just pen and paper. Their HB protocol is illustrated in Figure 1. The secret key is $\mathbf{s} \in \mathbb{Z}_2^\ell$ where ℓ is chosen such that the $\mathsf{LPN}_{\tau,\ell}$ problem is hard. The verifier sends as first message a challenge $\mathbf{A} \in \mathbb{Z}_2^{n\times\ell}$ (where n is a statistical security parameter), and the prover answers with an LPN sample $\mathbf{A}.\mathbf{s} \oplus \mathbf{e}$. The verifier accepts if the prover's answer $\mathbf{y}$ is of the form $\mathbf{y} = \mathbf{A}.\mathbf{s} \oplus \mathbf{e}$ for some low-weight $\mathbf{e}$. The expected weight of a correctly generated $\mathbf{e}$ is $n\tau$, the acceptance threshold of the verifier is set to $n\tau'$ for some $\tau < \tau' < 1/2$. This way the probability that a correctly generated $\mathbf{e} \leftarrow \mathsf{Ber}_\tau^n$ has weight $\geq n\tau'$ and thus V would reject (this is the completeness error) is exponentially small in n (this is shown using the Chernoff bound).

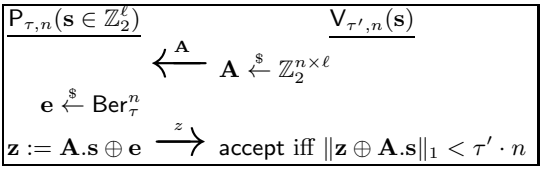


Fig. 1. The HB identification protocol [25], secure against passive attacks

The HB protocol can be proven secure against passive attacks assuming LPN is hard, but it can be easily broken with an active attack. Subsequently, Juels and Weis [29] proposed the HB^+ protocol, illustrated in Figure 2, with extends HB by one extra round. Their motivation was to find a protocol suitable for light weight devices like RFID tags, where an active attack is easy to launch. The HB^+ protocol is secure against active attacks, but not MIM attacks.[23] The HB^+ protocol has three rounds (not two like HB), which means the prover has to keep state in-between rounds, this is problematic for the devices like RFID tags.

The first two round protocol with active security was proposed in [30]. The design of the protocol is inspired by the subspace LPN problem, and diverges from the design of all previous LPN based protocols as now the randomness $\mathbf{A}$

[23] [29] only prove active security for a sequential version of the HB^+ protocol, where $\mathbf{A}$ is send row-by-row, and thus needs n rounds. Active security of the parallel protocol as in Figure 2 was later proven in [31].

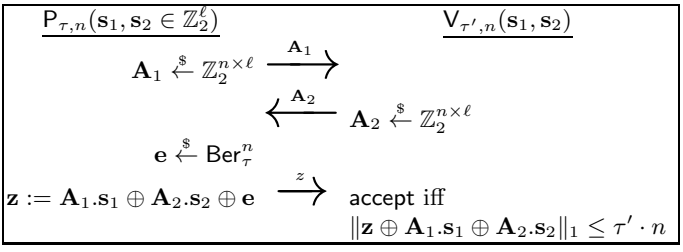


Fig. 2. The HB^+ identification protocol [29,31], secure against active attacks

is chosen by the prover P, and is not as a challenge chosen by V. Instead, the challenge chosen by V is a vector $\mathbf{v}$ which specifies a subset $\mathbf{s} \wedge \mathbf{v}$ ($\wedge$ denotes bitwise AND) of the secret $\mathbf{s}$. The prover answers with a subspace LPN sample $\mathbf{A}, \mathbf{A}.(\mathbf{s} \wedge \mathbf{v}) \oplus \mathbf{e}$.

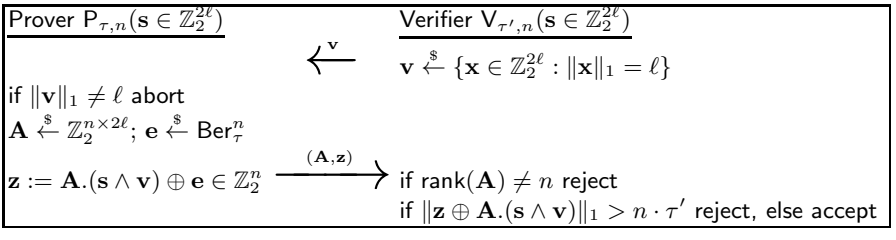


Fig. 3. A two-round identification protocol with active security [30]

All the protocols discussed above can be easily broken by a MIM attack.[24] [30] shows how to transform the protocol from Figure 3 into a message authentication code (MAC). This also gives the first efficient MIM secure protocol from LPN as a MAC easily implies a two-round MIM secure protocol. A more efficient (and generic) transformation using pairwise independent hashing (instead of a permutation) appears in [13].

Public-Key Identification, Commitments and Zero-Knowledge

Public-Key Identification. In a public-key identification protocol the prover and verifier do not share a secret key. Instead, the prover knows a secret key sk, and a corresponding public key pk is known to everyone (i.e. verifiers and potential adversaries.) This setting is often favorable as it allows much simpler key-management. A verifier must make sure to learn pk authentically, but it must

[24] A MIM attack on HB^+ is given in [22]. As outlined in [30], a similar attacks exists for the protocol in Figure 3 (in [30] this attack is phrased as an attack on the protocol when used as a MAC, but exactly the same attacks works in the MIM setting.)

not remain secret. One can construct an identification scheme from any one-way function[25] $f(.)$ by setting $sk = x$ and $pk = f(x)$ for a random x. To prove its identity, the prover uses a zero-knowledge proof of knowledge (ZKPOK) [19] to convince the verifier that he knows the secret x, while not revealing any information beyond that. ZKPOKs exist for any one-way function [20], but this generic constructions are too computationally expensive to be used in practice.

For some particular functions, ZKPOKs exist which are much more efficient than the generic constructions, in particular, many number theoretical functions admit efficient proofs. For example Schnorr's protocol [45], which is a particularly simple 3-round ZKPOK (a so called Σ-protocol) to prove knowledge of the discrete logarithm x of some value g^x (where g is a generator of a prime order cyclic group.)

This elegant number theoretical constructions involve multiplications or even exponentiations over large moduli, and thus are still too computationally expensive for very weak devices like RFID tags. A few alternative protocols based on combinatorial (typically NP-complete) problems were suggested which avoid such expensive operations, including the Permuted Kernels Problem [46], the Permuted Perceptrons Problem [42] and Syndrome Decoding (of random linear codes) [48]. Stern's protocol [48] can be modified [28] (using the equivalence of LPN and decoding random linear codes) to get an efficient ZKPOK for the LPN problem. That is, given $(\mathbf{A}, \mathbf{y})$, one proves knowledge of $\mathbf{s}$ and a (low weight) $\mathbf{e}$ such that $\mathbf{y} = \mathbf{A}.\mathbf{s} \oplus \mathbf{e}$.[26]

String Commitments and Zero-Knowledge. A commitment scheme is the digital analogue of an envelope. The committing party can compute a commitment σ to an input $\mathbf{m}$, this commitment hides the committed message $\mathbf{m}$ (this is called the hiding property). Later the committing party can open σ and reveal $\mathbf{m}$, but he cannot open it to any other $\mathbf{m}' \neq \mathbf{m}$ (this is called the binding property.) The LPN problem allows for very simple perfectly binding[27] string commitments schemes [28]: the commitment to a bit-string $\mathbf{m}$ is $\mathbf{A}.(\mathbf{s}\|\mathbf{m}) \oplus \mathbf{e}$, i.e. it's an LPN sample using a secret whose first part $\mathbf{s}$ is random, and the second part is the message $\mathbf{m}$. $\mathbf{A}$ is a fixed random public matrix. To open a commitment one reveals $\mathbf{s}\|\mathbf{m}$ and the (low weight) $\mathbf{e}$.

[25] And more generally, any NP relation where one can efficiently sample instance/witness pairs and where it's hard to compute a witness for an instance sampled like that.

[26] In general the public-key setting is more demanding than the secret-key one (note that a public-key scheme trivially implies a secret-key one, just use (sk, pk) as the shared secret key.) This is also the case here, the LPN based public-key scheme [28] is at least an order of magnitude less efficient than, say the MIM secure scheme from [30]. Moreover the public-key scheme needs three rounds (as opposed to two for the secret-key setting), and even this is only true in the idealized "random oracle model" as it uses the Fiat-Shamir transformation to get from honest to full zero-knowledge. Without random oracles the round complexity is linear in the statistical security parameter n.

[27] Perfectly binding means that even a computationally unbounded adversary cannot open a commitment in two different ways.

The above-mentioned ZKPOK for LPN can be extended to not only prove knowledge of $\mathbf{s}$, but even to show that for any $\mathbf{X}, \mathbf{y}$ it holds that $\mathbf{X}.\mathbf{s} = \mathbf{y}$ [28]. For the commitment scheme as just described, this protocol can be used prove that, for any $\mathbf{X}, \mathbf{y}$, the value in a commitment satisfies $\mathbf{X}.\mathbf{m} = \mathbf{y}$. Here $\mathbf{X}.\mathbf{m}$ can e.g. be a subset of the bits of $\mathbf{m}$, which is something useful in cut-and-choose proofs. For this last application, the LPN assumption is not enough, but one has to rely on the decisional XPLN assumption discussed in the previous section.

3 Conclusions and Open Problems

A common dogma in the realm of secret-key cryptography is that provably secure schemes cannot be efficient enough to compete with dedicated constructions like the AES block-cipher. Recent constructions based on the hardness of LPN have at least challenged this viewpoint. Although we don't have an efficient block-cipher from LPN (and block-ciphers are used for almost all secret-key tasks), we have direct constructions for the most important tasks like encryption, identification and message authentication. It seems conceivable that for some settings (most notably for lightweight devices like RFIDs[28]) provable security is not just a nice theoretical feature, but actually can lead us to constructions which outperform known dedicated constructions in terms of efficiency vs. practical security (a viewpoint largely accepted in the realm of *pubic-key* cryptography.)

Security of LPN. We have discussed several variants of the LPN problem in Section 1. For some useful variants, like decisional XLPN or LPN where the secret only has high min-entropy, the relation to the standard LPN problem is open. Another intriguing open problem is the following, does hardness of LPN with noise rane τ imply anything for LPN with smaller noise rate $\tau' < \tau$? For all we know, there could be threshold such that LPN with noise rate τ is hard, but easy for any $\tau' < \tau$ (though, this seems not very likely.)

Constructions. What other primitives can we construct from LPN? Two basic primitives we don't know how to get from the (standard) LPN problem are public-key encryption and collision resistant hash functions. Can we construct these, or is there a fundamental reason why the more general LWE problem allows for such objects, but LPN does not?

[28] One disadvantage the LPN based schemes have to classical block-cipher based solutions is that they require more randomness than block-cipher based schemes (e.g. with a block-cipher, we get identification where only the verifier, but not the prover needs to sample random bits. And message authentication codes can be completely deterministic, whereas the LPN based MAC is probabilistic.) This randomness needs make LPN based schemes poor candidates for powerful processors (as e.g. used in laptops, or even smart-phones) where generating randomness is expensive compared to clock-cycles or code-size. On weak devices like RFIDs or smart cards, this is not (or less) the case. Moreover even a priori deterministic computations are randomized on such devices by "masking" or "blinding" in order to protect the computation from side-channel attacks. Potentially, the randomization used for LPN based schemes provides the same effect as masking or blinding for free.

References

1. Applebaum, B., Barak, B., Wigderson, A.: Public-key cryptography from different assumptions. In: Schulman, L.J. (ed.) 42nd ACM STOC, pp. 171–180. ACM Press (2010)
2. Applebaum, B., Cash, D., Peikert, C., Sahai, A.: Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 595–618. Springer, Heidelberg (2009)
3. Arora, S., Ge, R.: New Algorithms for Learning in Presence of Errors. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 403–415. Springer, Heidelberg (2011)
4. Alekhnovich, M.: More on average case vs approximation complexity. In: 44th FOCS, pp. 298–307. IEEE Computer Society Press (2003)
5. Blum, A., Furst, M.L., Kearns, M.J., Lipton, R.J.: Cryptographic Primitives Based on Hard Learning Problems. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 278–291. Springer, Heidelberg (1994)
6. Blum, A., Adam Kalai, A., Wasserman, H.: Noise-tolerant learning, the parity problem, and the statistical query model. J. ACM 50(4), 506–519 (2003)
7. Berlekamp, E.R., McEliece, R.J., van Tilborg, H.C.A.: On the inherent intractability of certain coding problems. IEEE Trans. Information Theory IT-24(3), 384–386 (1978)
8. Banerjee, A., Peikert, C., Rosen, A.: Pseudorandom functions and lattices. Cryptology ePrint Archive, Report 2011/401 (2011), http://eprint.iacr.org/
9. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Ashby, V. (ed.) ACM CCS 1993, pp. 62–73. ACM Press (1993)
10. Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai Trees, or How to Delegate a Lattice Basis. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 523–552. Springer, Heidelberg (2010)
11. Cash, D., Kiltz, E., Shoup, V.: The Twin Diffie-Hellman Problem and Applications. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 127–145. Springer, Heidelberg (2008)
12. Coppersmith, D.: Rapid multiplication of rectangular matrices. SIAM J. Comput. 11(3), 467–471 (1982)
13. Dodis, Y., Kiltz, E., Pietrzak, K., Wichs, D.: Message authentication, revisited (manuscript, 2011)
14. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31, 469–472 (1985)
15. Feldman, V., Gopalan, P., Khot, S., Ponnuswami, A.K.: New results for learning noisy parities and halfspaces. In: 47th FOCS, pp. 563–574. IEEE Computer Society Press (2006)
16. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st ACM STOC, pp. 169–178. ACM Press (2009)
17. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM 33, 792–807 (1986)
18. Goldwasser, S., Kalai, Y.T., Peikert, C., Vaikuntanathan, V.: Robustness of the learning with errors assumption. In: ICS, pp. 230–240 (2010)
19. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing 18(1), 186–208 (1989)

20. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. Journal of the ACM 38(3), 691–729 (1991)
21. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC, pp. 197–206. ACM Press (2008)
22. Gilbert, H., Robshaw, M., Sibert, H.: An active attack against hb+ - a provably secure lightweight authentication protocol. Cryptology ePrint Archive, Report 2005/237 (2005), http://eprint.iacr.org/
23. Gilbert, H., Robshaw, M.J.B., Seurin, Y.: $\mathrm{HB}^{\sharp}$: Increasing the Security and Efficiency of HB^{+}. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 361–378. Springer, Heidelberg (2008)
24. Gilbert, H., Robshaw, M.J.B., Seurin, Y.: How to Encrypt with the LPN Problem. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 679–690. Springer, Heidelberg (2008)
25. Hopper, N.J., Blum, M.: Secure Human Identification Protocols. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 52–66. Springer, Heidelberg (2001)
26. Håstad, J., Impagliazzo, R., Levin, L.A., Luby, M.: A pseudorandom generator from any one-way function. SIAM Journal on Computing 28(4), 1364–1396 (1999)
27. Heyse, S., Kiltz, E., Lyubashevsky, V., Paar, C., Pietrzak, K.: An efficient authentication protocol based on ring-lpn (manuscript, 2011)
28. Jain, A., Pietrzak, K., Tentes, A.: Commitments and efficient zero-knowledge from hard learning problems (manuscript, 2011)
29. Juels, A., Weis, S.A.: Authenticating Pervasive Devices with Human Protocols. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 293–308. Springer, Heidelberg (2005)
30. Kiltz, E., Pietrzak, K., Cash, D., Jain, A., Venturi, D.: Efficient Authentication from Hard Learning Problems. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 7–26. Springer, Heidelberg (2011)
31. Katz, J., Shin, J.S., Smith, A.: Parallel and concurrent security of the HB and HB+ protocols. Journal of Cryptology 23(3), 402–421 (2010)
32. Levieil, É., Fouque, P.-A.: An Improved LPN Algorithm. In: De Prisco, R., Yung, M. (eds.) SCN 2006. LNCS, vol. 4116, pp. 348–359. Springer, Heidelberg (2006)
33. Lyubashevsky, V., Micciancio, D., Peikert, C., Rosen, A.: SWIFFT: A Modest Proposal for FFT Hashing. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 54–72. Springer, Heidelberg (2008)
34. Lyubashevsky, V., Peikert, C., Regev, O.: On Ideal Lattices and Learning with Errors Over Rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (2010)
35. Luby, M., Rackoff, C.: How to construct pseudorandom permutations from pseudorandom functions. SIAM Journal on Computing 17(2) (1988)
36. Lyubashevsky, V.: The Parity Problem in the Presence of Noise, Decoding Random Linear Codes, and the Subset Sum Problem. In: Chekuri, C., Jansen, K., Rolim, J.D.P., Trevisan, L. (eds.) APPROX 2005 and RANDOM 2005. LNCS, vol. 3624, pp. 378–389. Springer, Heidelberg (2005)
37. McEliece, R.J.: A Public-Key Cryptosystem Based On Algebraic Coding Theory. Deep Space Network Progress Report 44, 114–116 (1978)
38. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $o(2^{0.054n})$. In: ASIACRYPT (2011)

39. Naor, M., Reingold, O.: Synthesizers and their application to the parallel construction of pseudo-random functions. J. Comput. Syst. Sci. 58(2), 336–375 (1999)
40. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In: Mitzenmacher, M. (ed.) 41st ACM STOC, pp. 333–342. ACM Press (2009)
41. Pietrzak, K.: Subspace LWE (2010) (manuscript)
42. Pointcheval, D., Poupard, G.: A new np-complete problem and public-key identification. Des. Codes Cryptography 28(1), 5–31 (2003)
43. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC, pp. 84–93. ACM Press (2005)
44. Regev, O.: The learning with errors problem (invited survey). In: IEEE Conference on Computational Complexity, pp. 191–204 (2010)
45. Schnorr, C.-P.: Efficient Identification and Signatures for Smart Cards. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 239–252. Springer, Heidelberg (1990)
46. Shamir, A.: An Efficient Identification Scheme based on Permuted Kernels (Extended Abstract). In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 606–609. Springer, Heidelberg (1990)
47. Stern, J.: A Method for Finding Codewords of Small Weight. In: Cohen, G., Godlewski, P. (eds.) Coding Theory 1986. LNCS, vol. 311, pp. 106–113. Springer, Heidelberg (1988)
48. Stern, J.: A New Identification Scheme Based on Syndrome Decoding. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 13–21. Springer, Heidelberg (1994)

A Quick Tour of Word Sense Disambiguation, Induction and Related Approaches

Roberto Navigli

Dipartimento di Informatica
Sapienza Università di Roma,
Via Salaria 113, 00198 Roma, Italy
navigli@di.uniroma1.it
http://www.users.di.uniroma1.it/~navigli

Abstract. Word Sense Disambiguation (WSD) and Word Sense Induction (WSI) are two fundamental tasks in Natural Language Processing (NLP), i.e., those of, respectively, automatically assigning meaning to words in context from a predefined sense inventory and discovering senses from text for a given input word. The two tasks have generally been hard to perform with high accuracy. However, today innovations in approach to WSD and WSI are promising to open up many interesting new horizons in NLP and Information Retrieval applications. This paper is a quick tour on how to start doing research in this exciting field and suggests the hottest topics to focus on.

Keywords: computational lexical semantics, Word Sense Disambiguation, Word Sense Induction, text understanding.

1 Introduction

Word Sense Disambiguation (WSD) – the task of computationally determining the correct sense of a word in context – is a core research topic in computational linguistics and natural language processing. The reason for its importance lies in the ambiguity of human language, which is so pervasive that huge numbers of words can be interpreted in multiple ways depending on the context in which they occur. For example, consider the following sentences:

(a) I can hear *bass* sounds;
(b) They like grilled *bass*.

It is evident that the occurrences of the word *bass* in the two sentences denote different meanings: low-frequency tones and a type of fish, respectively. Unfortunately, identifying the specific meaning that a word assumes in context is only apparently simple, especially for a machine. In fact, while most of the time humans do not even think about the ambiguities of language, machines need to process unstructured textual information and transform it into data structures which must then be analyzed in order to determine the underlying meaning. The computational identification of meaning for words in context

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 115–129, 2012.

is called **Word Sense Disambiguation**. For instance, as a result of disambiguation, sentence (b) above should ideally be sense-tagged as "They like/ENJOY grilled/COOKED bass/FISH". However, in order to perform WSD, a sense inventory must be available for the word of interest (*bass* in the above example). A **sense inventory** is an artifact we are all very familiar with. In fact, it is merely a list of senses of a given word, which is nothing more than what is available in traditional dictionaries – knowledge resources we have been accustomed to using since our earliest schooldays. However, sense inventories need to be updated continuously and, unfortunately, tend to take into account only lexicographic meanings (e.g., bass as a fish), and largely ignore named entities (e.g., Bass as a town in Australia). As a result, automated text understanding is hindered if real-world instances occur in text, e.g.:

(c) I just arrived in Bass, Victoria.

To account for the above-mentioned issues, one can resort to techniques aimed at the automatic discovery of word senses from text, a task called **Word Sense Induction** (WSI).

This paper is a quick tour on starting to do research in the exciting fields of WSD and WSI (see [53] for a complete survey). The paper is organized as follows: in Sections 2 and 3 we illustrate the main topics in WSD and WSI, respectively; in Section 4 we introduce the lexical substitution approach; in Section 5 we discuss other techniques for text understanding; finally we give concluding remarks in Section 6.

2 Word Sense Disambiguation

WSD is believed by many to be the first brick on the road to automated text understanding and a potentially crucial resource for applications such as information retrieval and machine translation. For this reason, much research has been devoted to it over the last few decades. However, several key issues have arisen over the years which need to be considered if a good grasp of the field is to be attained. We overview the techniques, together with the main issues in WSD, in the following subsections.

2.1 Sense Representation

Word senses are the lifeblood of WSD, regardless of the approach we intend to use. A **word sense** is a commonly-accepted meaning of a word. For instance, in the two sentences (a) and (b) from Section 1, the word *bass* can have two senses, i.e., the sound one and the fish one. These two senses are called **homonyms**, in that they are completely different (even etymologically) and are actually two completely different words which just happen to be written in the same way. However, more subtle sense distinctions can be made for a word. For instance, dictionaries can distinguish between *bass* as the lowest part of the musical range and *bass* as the lowest part in polyphonic music. This kind of ambiguity is called

polysemy. Unfortunately, polysemous senses can be created at any level of granularity, thus leading to possibly very fine-grained sense distinctions. This is in fact the case with the most widely-used machine-readable dictionary and computational lexicon of English, namely WordNet [49,19], which has been in use for almost two decades. To cope with the fine granularity problem, researchers have come up with different ideas aimed at creating coarser sense distinctions in WordNet. Two of these efforts led to the organization of WSD evaluation competitions at Semeval-2007 (the reference international semantic evaluation competition), namely:

- Manual approach: the OntoNotes project [27,67] aims at creating sense distinctions by iteratively submitting new partitions of senses for the given word to sense annotators, until the latter achieve a 90% agreement in a text annotation task.
- Automatic approach: another project [52,57] aimed at the creation of coarser-grained senses by clustering semantically similar senses using WSD techniques. The clustering is obtained by automatically mapping WordNet senses to a reference machine-readable dictionary with hierarchical sense distinctions, namely the Oxford Dictionary of English.

Another way to deal with the granularity issue is to use a probabilistic generative model, e.g., based on the well-known noisy channel model [77], which can estimate the distribution of coarse-grained semantic classes, as opposed to fine-grained WordNet senses. Recently, it has also been suggested that one could abandon the adoption of the WordNet sense inventory. Examples include the use of Wikipedia [46], Wiktionary [45], or even newly-conceived machine-readable resources, such as DANTE [42]. Thus, an important open research question on the topic of sense representation is: what sense representation, and inventory, is best (and for which application [63])?

2.2 Techniques

There are three mainstream approaches to WSD, namely:

- **Supervised WSD:** these approaches use machine learning methods to learn a classifier for the target word from labeled training sets, i.e., sets of examples encoded as vectors whose elements represent features, with a special element representing the appropriate sense label (or class). Among supervised methods, memory-based learning and SVM approaches proved to be among the best systems in several international competitions [26,12,47,23,10,79].
- **Knowledge-based WSD:** these methods exploit knowledge resources (such as dictionaries, thesauri, ontologies, etc.) to determine the senses of words in context. They have the advantage of a wider coverage, thanks to the use of large amounts of structured knowledge. The best knowledge-based systems in the literature, such as Degree [56,66] or Personalized PageRank [4], exploit WordNet or other resources (e.g., BabelNet [58]) to build a semantic graph

and exploit the structural properties of the graph (either locally to the input sentence or globally) in order to choose the appropriate senses of words in context.
- **Unsupervised WSD:** these are Word Sense Induction techniques aimed at discovering senses automatically based on unlabeled corpora. They do not exploit any manually sense-tagged corpus to provide a sense choice for a word in context (see Section 3).

The question of which approach is best in general, and in which application, is still very much open. In fact, until recently, the general belief was that supervised WSD performed better than knowledge-based WSD. However, recent results show that, in the presence of enough knowledge (see Section 2.4) or within a domain (see Section 2.6), knowledge-rich systems can beat supervised approaches [66,5,1], while providing at the same time much wider coverage.

2.3 Performance

A well-known issue in Word Sense Disambiguation is performance. As recently as 2004, one of the major arguments against continuing research in the field was that state-of-the-art accuracy was around 65% in an all-words setting, i.e., when all content words must be disambiguated[1]. In other words, on average only 65 out of 100 content words could be automatically associated with an appropriate meaning [72]. It was remarked that the representation of senses was one of the main obstacles to high-performance WSD.

In recent years progress has been made that has led to a considerable boost in disambiguation performance, from about 65% to 82-83% accuracy [10,66] in an all-words setting and when the adopted sense distinctions are less fine-grained (see Section 2.1). As an alternative solution to reducing the granularity of word senses, it has been proposed to calculate the correlation between the graded sense assignments output by a fine-grained WSD system and those obtained as a result of the manual graded judgments produced by human annotators [16].

2.4 Knowledge

Another key factor in WSD, and one which is strongly linked to performance, is **knowledge**. It has been shown that the higher the amount of high-quality knowledge, the higher the performance. Specifically:

- In supervised WSD, performance can be increased considerably when hundreds of training examples are available for each target word [41].
- Knowledge-based WSD – which relies on machine-readable dictionaries or computational lexicons viewed as semantic networks – has been shown to benefit greatly from the addition of relatedness edges [56], i.e., relations of a

[1] We use accuracy in the general sense of recall or F1 measure, depending on the task (see [53] for details on the evaluation measures).

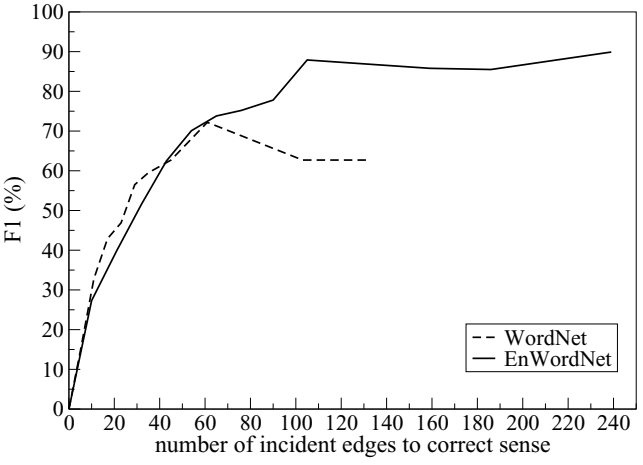


Fig. 1. The more structural knowledge (i.e., semantic relations), the better the performance (from [56])

syntagmatic nature (such as *bus* is related to *driver*), as opposed to paradigmatic relations such as hypernymy (e.g., *bus* is a *public transport*). When more than 100 semantic relations (i.e., edges in the semantic network) are available for each word sense, performance can achieve up to 90% accuracy. We report this result in Figure 1 (taken from [56]), where the performance of a simple knowledge-based WSD, based on the degree of each candidate sense in the semantic network, depends on the richness of the graph itself (EnWordNet is an enriched version of WordNet with about 60,000 additional relatedness edges).

However, these results lead to another question, that of the **knowledge acquisition bottleneck**, i.e., how can we acquire enough knowledge to:

i) provide wide coverage of the lexicon,
ii) obtain high performance?

It has been estimated that, at a throughput of one word instance per minute, dozens (if not hundreds) of person-years would be required to provide enough training data [15]. Moreover, this effort would have to be repeated for each language. Possible solutions to the knowledge acquisition bottleneck include the exploitation of bilingual corpora to create large and reliable training sets [78], the automatic sense labeling of corpora [14,8] and the use and integration of large-scale collaborative resources such as Wikipedia [65,66].

2.5 Multilinguality

Currently, most research in WSD is conducted on the English language, because this is the language for which the vast majority of resources are available.

However, recently there has been an increasing interest towards other languages, such as Italian [38], Spanish [40], Chinese [30], Japanese [60], Turkish [61], etc.

More importantly, attention is increasingly being paid to performing WSD across languages, a task referred to as **cross-lingual WSD** [34]. In this setting, an input sentence is provided in a source language (e.g., English) and the WSD system has to provide word senses encoded in a target language (e.g., Italian). The sense inventory is obtained using translations harvested from a parallel corpus, instead of using predefined sense labels. The basic assumption – supported by several studies [21,28,59] – is that the sense distinctions of a word in a source language are determined by the different translations of the word in other languages. This approach has several benefits over the traditional monolingual WSD task: it can be applied to any language of interest, it can easily be integrated into real applications such as machine translation, and it inherently addresses the sense granularity problem. However, it requires a wide-coverage bilingual corpus, a requirement which is not easy to satisfy.

A challenging research direction is performing truly **multilingual WSD**, i.e., returning senses lexicalized in many languages. Recent efforts in this directions include the use of contextual cues or filters from other languages [24,35] and the recently-released multilingual semantic network called BabelNet [58].

2.6 Domain WSD

In everyday life the use of natural language is often confined to specific fields of knowledge. Examples include newspaper sections, blogs and Web sites on specific topics, business documents, etc. Performing a semantic analysis of domain text is thus a crucial task that can boost applications such as Question Answering, Information Retrieval and Information Extraction.

Domain WSD is often performed in a **type-based** fashion, i.e., by assigning a single sense per word, rather than considering each single context the target word occurs in (i.e., **token-based** WSD). Distributionally similar neighbors in raw text can be used as cues to determine the predominant sense of a target word by means of a semantic similarity measure [32,43]. Other distributional methods include the use of a word-category cooccurrence matrix, where categories are coarse senses obtained from an existing thesaurus [51], and synonym-based word occurrence counts [33]. Domain-informed methods have also been proposed which make use of domain labels as cues for disambiguation purposes [22].

Domain-driven approaches have been shown to obtain the best performance among the unsupervised alternatives [73]. Their performance is typically lower than supervised systems. On the other hand, supervised systems need sense-tagged corpora to perform accurate WSD, a requirement that cannot be satisfied for most domains unless a general-purpose corpus is mixed with a smaller domain-specific training set [31].

In the presence of large amounts of structured knowledge for all domains, knowledge-based approaches obtain state-of-the-art performance [66]. A recent approach of this kind [55] with little human intervention has been proposed that proceeds in two steps: first semantic model vectors are learned for many domains,

next, a word context is classified and the selected semantic model vector is used to perform high-quality domain WSD. As a result, the text classification and disambiguation steps are effectively combined.

3 Word Sense Induction

Given the above-mentioned difficulties, Word Sense Induction is an attractive alternative to WSD. In this setting we give up on the traditional notion of a predefined word sense and use unsupervised techniques to automatically identify the set of senses denoted by a word. As a result we shift our focus away from how to select the most suitable senses from an inventory towards how to automatically discover senses from text. In fact, these methods induce word senses from raw text by clustering word occurrences on the basis of the **distributional hypothesis**, i.e., the idea that a given word – used in a specific sense – tends to co-occur with the same neighbouring words [25].

3.1 Techniques

The main approaches to WSI proposed in the literature are the following:

- **Context clustering:** the underlying hypothesis of this approach is that the distributional profile of words implicitly expresses their semantics (see also Section 5). Each occurrence of a target word in a corpus is represented here as a **context vector**. These context vectors can be either first-order vectors, which directly represent the context at hand, or second-order vectors, i.e., the contexts of the target word are similar if their words tend to co-occur together. The vectors are then clustered into groups, each identifying a sense of the target word. A well-known approach to context clustering is the context-group discrimination algorithm [70].
- **Word clustering:** a different approach to the induction of word senses consists of clustering words which are semantically similar and can thus convey a specific meaning. A prototypical example is Lin's algorithm [36], which exploits syntactic dependencies which occur in a corpus to produce sets of words for each discovered sense of a target word. The Clustering By Committee [64] also uses syntactic contexts, but exploits a similarity matrix to encode the similarities between words and relies on the notion of committees to output different senses of the word of interest.
- **Co-occurrence Graphs:** these methods are related to word clustering approaches, but build and analyse graphs of word co-occurrence to identify the set of senses of a given word. Co-occurrences between words can be obtained on the basis of grammatical [76] or collocational relations [75]. However, successful approaches such as HyperLex [75] – a graph algorithm based on the identification of hubs in co-occurrence graphs – have to cope with the need to tune a large number of parameters [2]. To deal with this issue a graph-based algorithm has recently been proposed which is based on simple graph patterns, namely triangles and squares [54]. The patterns aim at identifying meanings using the local structural properties of the co-occurrence graph.

– **Probabilistic clustering:** another option is to adopt a probabilistic approach, e.g., a Bayesian framework [9], and formalize WSI in a generative model. First, for each ambiguous word a distribution of senses is drawn. Then, context words are generated according to this distribution. Different senses can thus be obtained which have different word distributions.

Finally, we mention a recent successful approach based on latent semantic word spaces [11], which finds latent dimensions of meaning using non-negative matrix factorization, uses these dimensions to distinguish between different senses of a target word, and then proceeds to disambiguate each given instance of that word.

3.2 Evaluation

One key issue in WSI is that of evaluation. WSI is actually a specific instance of the clustering problem, so it is just as hard to evaluate as any clustering algorithm. Unfortunately, evaluating the output of a clustering algorithm is hard even for humans. The main difficulty lies in the fact that there is no single gold-standard clustering on which human annotators can agree. Nonetheless, different approaches have been proposed in the literature to assess the output quality of a WSI algorithm. Three main evaluation techniques have been developed:

– **Unsupervised evaluation:** the clusters of sentences corresponding to the induced senses are evaluated against a gold-standard annotation. The quality of a clustering can be determined by means of measures such as the V-Measure [69], which calculates its homogeneity and completeness, Paired F-Score [39], the RandIndex [68,54], etc.
– **Supervised evaluation:** in this setting the output of WSI is evaluated in a WSD task [3]. To do this, the induced senses are mapped to gold-standard senses for the target word. Each test sentence is then annotated with the best gold standard sense, and precision and recall are used to determine the quality of the resulting WSD.
– **Within an application:** a further way to evaluate the output of WSI is within an application. A paradigmatic example of this kind is the application of WSI to Web search result clustering [54,13], where WSI techniques have been shown to consistently surpass non-semantic state-of-the-art systems.

3.3 Coverage

A second issue with WSI is that of coverage. How do we know the coverage of the induced senses? This question is strongly related to the evaluation issue above. In fact, we might answer that it depends on the application. Alternatively, we might answer that it depends on the results of a supervised evaluation based on the mapping to a gold standard sense inventory. On the other hand, as mentioned in Section 1, WSI has the potential to harvest even more senses than those available in a traditional predefined sense inventory, like the one used in WSD (e.g., WordNet).

4 Lexical Substitution

Another solution to the sense representation problem – closely related to WSI – is to cast WSD as a lexical substitution task [44], i.e., by asking systems to replace a target word in context with a valid alternative substitute which preserves the meaning of the target word as much as possible. For instance, consider the sentence:

(d) They *like* grilled bass.

Suitable replacements for *like* are *love* or *are fond of*, but – in this context – not *wish*. The advantages of this approach to text understanding lie in the lack of a predefined sense inventory and, as opposed to WSI in which senses must be discovered in advance, in the freedom to choose any appropriate substitute dynamically depending on the context. Such a large degree of freedom, however, in a sense, is also a disadvantage in that, similarly to WSI, it makes it difficult to evaluate and compare lexical substitution systems.

The introduction of the lexical substitution task has generated considerable interest within the research community, especially in terms of evaluation measures [29] and multilinguality [48].

5 Other Techniques

Recently other techniques for the unsupervised semantic processing of text have been revamped, the main one being vector space models of semantics [74]. Such models are obtained by exploiting the distributional statistics of words and they are typically based on matrices whose cells encode the relationship between terms and documents, words and contexts, or word pairs and patterns. In contrast to WSI in which the meanings of a word are modeled by a set of induced senses, a word is here represented by a vector whose elements are obtained from grammatical dependencies [36,62], selectional preferences [17] or pattern-based co-occurrences [37]. The resulting matrices, which are usually of very large dimensions, can then be smoothed using truncated Singular Value Decomposition or related techniques for noise and sparsity reduction.

5.1 Compositionality and Meaning

The vectors, acquired by means of distributional semantic techniques, can be compared (e.g., using cosine similarity or geometric measures of distance) or combined by means of vector sum, or multiplication [50], or even tensor products [71]. Solutions to specific kinds of composition, such as adjective-noun, have also been proposed (e.g., by composing matrices with vectors [7]).

Recently it has been reported that state-of-the-art results can be obtained when examplar models are adopted, i.e., when the word meaning is modeled by using relevant occurrences only, rather than merging all the occurrences in a single word vector [18]. The use of more structured vectors, which take into account

syntactic relations, has also been proposed [17]. However, while a general framework has been studied [6], basic questions remain concerning the scalability and applicability to real scenarios of current approaches to distributional semantics and, similarly to WSI, their *in vitro* evaluation, i.e., outside an application.

6 Conclusions

So, how to start in Word Sense Disambiguation and Induction? In this paper we have tried to provide a smattering of the essentials of the field. While a much more detailed account of the field can be found in [53], this paper touches on topics and issues which are very fresh and deserving of the attention of researchers right now.

We can already see the first evidence that text understanding techniques improve the state of the art in fields such as search result clustering [54,13] and lexicography [20]. However, much work is still needed to prove that a proper injection of semantics into real-world applications is always beneficial. The immediate next step is to focus on highly relevant applications, including machine translation (a historical application for WSD, in fact) and semantic search.

Acknowledgments. The author gratefully acknowledges the support of the ERC Starting Grant MultiJEDI No. 259234.

References

1. Agirre, E., de Lacalle, O.L., Soroa, A.: Knowledge-based WSD on specific domains: performing better than generic supervised WSD. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI). pp. 1501–1506. Pasadena, California (2009)
2. Agirre, E., Martínez, D., de Lacalle, O.L., Soroa, A.: Evaluating and optimizing the parameters of an unsupervised graph-based WSD algorithm. In: Proceedings of TextGraphs '06. pp. 89–96. New York, USA (2006)
3. Agirre, E., Soroa, A.: SemEval-2007 task 2: Evaluating word sense induction and discrimination systems. In: Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007), Prague, Czech Republic. pp. 7–12 (2007)
4. Agirre, E., Soroa, A.: Personalizing PageRank for Word Sense Disambiguation. In: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL), Athens, Greece. pp. 33–41 (2009)
5. Baldwin, T., Kim, S., Bond, F., Fujita, S., Martinez, D., Tanaka, T.: A reexamination of MRD-based Word Sense Disambiguation. ACM Transactions on Asian Language Information Processing (TALIP) 9, 4:1–4:21 (2010)
6. Baroni, M., Lenci, A.: Distributional memory: A general framework for corpus-based semantics. Computational Linguistics 36(4), 673–721 (2010)
7. Baroni, M., Zamparelli, R.: Nouns are vectors, adjectives are matrices: Representing adjective-noun constructions in semantic space. In: Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 1183–1193. MIT Stata Center, Massachusetts, USA (2010)

8. Brody, S., Lapata, M.: Good neighbors make good senses: Exploiting distributional similarity for unsupervised WSD. In: Proceedings of the 22nd International Conference on Computational Linguistics (COLING). pp. 65–72. Manchester, UK (2008)
9. Brody, S., Lapata, M.: Bayesian Word Sense Induction. In: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics (EACL). pp. 103–111. Athens, Greece (2009)
10. Chan, Y.S., Ng, H.T., Zhong, Z.: NUS-PT: Exploiting parallel texts for Word Sense Disambiguation in the English all-words tasks. In: Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007), Prague, Czech Republic. pp. 253–256 (2007)
11. de Cruys, T.V., Apidianaki, M.: Latent semantic word sense induction and disambiguation. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-HLT). pp. 1476–1485. Portland, Oregon, USA (2011)
12. Decadt, B., Hoste, V., Daelemans, W., van den Bosch, A.: Gambl, genetic algorithm optimization of memory-based WSD. In: Proceedings of the 3rd International Workshop on the Evaluation of Systems for the Semantic Analysis of Text (SENSEVAL-3), Barcelona, Spain. pp. 108–112 (2004)
13. Di Marco, A., Navigli, R.: Clustering web search results with maximum spanning trees. In: Proceedings of 12th International Conference of the Italian Association for Artificial Intelligence (AI*IA). pp. 201–212. Palermo, Italy (2011)
14. Diab, M.: Relieving the data acquisition bottleneck in Word Sense Disambiguation. In: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics (ACL). pp. 303–310. Barcelona, Spain (2004)
15. Edmonds, P.: Designing a task for SENSEVAL-2. Tech. rep., University of Brighton, U.K. (2000)
16. Erk, K., McCarthy, D.: Graded word sense assignment. In: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 440–449. Singapore (2009)
17. Erk, K., Padó, S.: A structured vector space model for word meaning in context. In: Proceedings of the 2008 Conference on Empirical Methods in Natural Language (EMNLP). pp. 897–906. Edinburgh, UK (2008)
18. Erk, K., Padó, S.: Exemplar-based models for word meaning in context. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL). pp. 92–97. Uppsala, Sweden (2010)
19. Fellbaum, C. (ed.): WordNet: An Electronic Database. MIT Press, Cambridge, MA (1998)
20. Flati, T., Navigli, R.: The CQC Algorithm: Cycling in graphs to semantically enrich and enhance a bilingual dictionary. Journal of Artificial Intelligence Research (JAIR), to appear (2012)
21. Gale, W.A., Church, K., Yarowsky, D.: Using bilingual materials to develop Word Sense Disambiguation methods. In: Proceedings of the Fourth International Conference on Theoretical and Methodological Issues in Machine Translation. pp. 101–112. Montreal, Canada (1992)
22. Gliozzo, A., Strapparava, C., Dagan, I.: Unsupervised and supervised exploitation of semantic domains in lexical disambiguation. Computer Speech and Language 18(3), 275–299 (2004)
23. Grozea, C.: Finding optimal parameter settings for high performance Word Sense Disambiguation. In: Proceedings of the 3rd International Workshop on the Eval-

uation of Systems for the Semantic Analysis of Text (SENSEVAL-3), Barcelona, Spain. pp. 125–128 (2004)
24. Guo, W., Diab, M.T.: Combining orthogonal monolingual and multilingual sources of evidence for all words WSD. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL). pp. 1542–1551. Uppsala, Sweden (2010)
25. Harris, Z.: Distributional structure. Word 10, 146–162 (1954)
26. Hoste, V., Hendrickx, I., Daelemans, W., van den Bosch, A.: Parameter optimization for machine-learning of Word Sense Disambiguation. Natural Language Engineering 8(4), 311–325 (2002)
27. Hovy, E., Marcus, M., Palmer, M., Ramshaw, L., Weischedel, R.: OntoNotes: The 90% solution. In: Companion Volume to the Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, New York, N.Y. pp. 57–60 (2006)
28. Ide, N., Erjavec, T., Tufiş, D.: Sense discrimination with parallel corpora. In: Proceedings of ACL-02 Workshop on WSD: Recent Successes and Future Directions. pp. 54–60. Philadelphia, USA (2002)
29. Jabbari, S., Hepple, M., Guthrie, L.: Evaluation metrics for the lexical substitution task. In: Proceedings of the Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL), Los Angeles, California. pp. 289–292 (2010)
30. Jin, P., Wu, Y., Yu, S.: SemEval-2007 task 05: Multilingual Chinese-English lexical sample. In: Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007). pp. 19–23. Prague, Czech Republic (2007)
31. Khapra, M., Kulkarni, A., Sohoney, S., Bhattacharyya, P.: All words domain adapted WSD: Finding a middle ground between supervision and unsupervision. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL). pp. 1532–1541. Uppsala, Sweden (2010)
32. Koeling, R., McCarthy, D., Carroll, J.: Domain-specific sense distributions and predominant sense acquisition. In: Proceedings of the Human Language Technology Conference and the 2005 Conference on Empirical Methods in Natural Language Processing (EMNLP), Vancouver, B.C., Canada. pp. 419–426 (2005)
33. Lapata, M., Keller, F.: An information retrieval approach to sense ranking. In: Proceedings of Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics, Rochester, N.Y. pp. 348–355. Rochester, USA (2007)
34. Lefever, E., Hoste, V.: SemEval-2010 task 3: Cross-lingual Word Sense Disambiguation. In: Proceedings of the 5th International Workshop on Semantic Evaluation. pp. 15–20. Uppsala, Sweden (2010)
35. Lefever, E., Hoste, V., Cock, M.D.: Parasense or how to use parallel corpora for Word Sense Disambiguation. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-HLT). pp. 317–322. Portland, Oregon, USA (2011)
36. Lin, D.: Automatic retrieval and clustering of similar words. In: Proceedings of the 17th International Conference on Computational linguistics (COLING). pp. 768–774. Montreal, Quebec, Canada (1998)
37. Lin, D., Pantel, P.: Dirt – discovery of inference rules from text. In: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining (KDD). pp. 323–328. San Francisco, CA, USA (2001)

38. Magnini, B., Giampiccolo, D., Vallin, A.: The italian lexical sample task at Senseval-3. In: Proceedings of Senseval-3: Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text. pp. 17–20. Barcelona, Spain (2004)
39. Manandhar, S., Klapaftis, I.P., Dligach, D., Pradhan, S.S.: SemEval-2010 task 14: Word sense induction & disambiguation. In: Proceedings of the 5th International Workshop on Semantic Evaluation. pp. 63–68. Uppsala, Sweden (2010)
40. Màrquez, L., Taulé, M., Martí, A., Artigas, N., García, M., Real, F., Ferrés, D.: Senseval-3: The Spanish lexical sample task. In: Proceedings of the 3rd International Workshop on the Evaluation of Systems for the Semantic Analysis of Text (SENSEVAL-3), Barcelona, Spain. pp. 21–24 (2004)
41. Martinez, D.: Supervised Word Sense Disambiguation: Facing Current Challenges, Ph. D. Thesis. University of the Basque Country, Spain (2004)
42. McCarthy, D.: Dante: a new resource for research at the syntax-semantics interface. In: Proceedings of Interdiciplinary Workshop on Verbs. Pisa, Italy (2010)
43. McCarthy, D., Koeling, R., Weeds, J., Carroll, J.: Unsupervised acquisition of predominant word senses. Computational Linguistics 33(4), 553–590 (2007)
44. McCarthy, D., Navigli, R.: The English lexical substitution task. Language Resources and Evaluation 43(2), 139–159 (2009)
45. Meyer, C.M., Gurevych, I.: How web communities analyze human language: Word senses in Wiktionary. In: Proceedings of the Second Web Science Conference. Raleigh, NC, USA (2010)
46. Mihalcea, R., Csomai, A.: Wikify! Linking documents to encyclopedic knowledge. In: Proceedings of the Sixteenth ACM Conference on Information and Knowledge management, Lisbon, Portugal. pp. 233–242 (2007)
47. Mihalcea, R., Faruque, E.: SenseLearner: Minimally supervised Word Sense Disambiguation for all words in open text. In: Proceedings of the 3rd International Workshop on the Evaluation of Systems for the Semantic Analysis of Text (SENSEVAL-3), Barcelona, Spain. pp. 155–158. Barcelona, Spain (2004)
48. Mihalcea, R., Sinha, R., McCarthy, D.: Semeval-2010 task 2: Cross-lingual lexical substitution. In: Proceedings of the 5th International Workshop on Semantic Evaluation. pp. 9–14. Uppsala, Sweden (2010)
49. Miller, G.A., Beckwith, R., Fellbaum, C.D., Gross, D., Miller, K.: WordNet: an online lexical database. International Journal of Lexicography 3(4), 235–244 (1990)
50. Mitchell, J., Lapata, M.: Vector-based models of semantic composition. In: Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL). pp. 236–244. Columbus, Ohio, USA (2008)
51. Mohammad, S., Hirst, G.: Determining word sense dominance using a thesaurus. In: Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL), Trento, Italy. pp. 121–128 (2006)
52. Navigli, R.: Meaningful clustering of senses helps boost word sense disambiguation performance. In: Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING-ACL), Sydney, Australia. pp. 105–112 (2006)
53. Navigli, R.: Word Sense Disambiguation: A survey. ACM Computing Surveys 41(2), 1–69 (2009)
54. Navigli, R., Crisafulli, G.: Inducing word senses to improve web search result clustering. In: Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 116–126. Boston, USA (2010)

55. Navigli, R., Faralli, S., Soroa, A., de Lacalle, O., Agirre, E.: Two birds with one stone: Learning semantic models for text categorization and Word Sense Disambiguation. In: Proceedings of the 20th ACM Conference on Information and Knowledge Management (CIKM). pp. 2317–2320. Glasgow, UK (2011)
56. Navigli, R., Lapata, M.: An experimental study on graph connectivity for unsupervised Word Sense Disambiguation. IEEE Transactions on Pattern Anaylsis and Machine Intelligence 32(4), 678–692 (2010)
57. Navigli, R., Litkowski, K.C., Hargraves, O.: SemEval-2007 task 07: Coarse-grained English all-words task. In: Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007), Prague, Czech Republic. pp. 30–35 (2007)
58. Navigli, R., Ponzetto, S.P.: BabelNet: Building a very large multilingual semantic network. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL), Uppsala, Sweden. pp. 216–225 (2010)
59. Ng, H.T., Wang, B., Chan, Y.S.: Exploiting parallel texts for Word Sense Disambiguation: an empirical study. In: Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL), Sapporo, Japan. pp. 455–462 (2003)
60. Okumura, M., Shirai, K., Komiya, K., Yokono, H.: SemEval-2010 task: Japanese WSD. In: Proceedings of the 5th International Workshop on Semantic Evaluation (Semeval-2010). pp. 69–74. Uppsala, Sweden (2010)
61. Orhan, Z., Çelik, E., Neslihan, D.: SemEval-2007 task 12: Turkish lexical sample task. In: Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007). pp. 59–63. Prague, Czech Republic (2007)
62. Pado, S., Lapata, M.: Dependency-based construction of semantic space models. Computational Linguistics 33(2), 161–199 (2007)
63. Palmer, M., Babko-Malaya, O., Dang, H.T.: Different sense granularities for different applications. In: Proceedings of 2nd Workshop on Scalable Natural Language Understanding Systems at HLT-NAACL-04. Boston, MA (2004)
64. Pantel, P., Lin, D.: Discovering word senses from text. In: Proceedings of the 8th International Conference on Knowledge Discovery and Data Mining (KDD). pp. 613–619 (2002)
65. Ponzetto, S.P., Navigli, R.: Large-scale taxonomy mapping for restructuring and integrating Wikipedia. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI), Pasadena, California, USA. pp. 2083–2088 (2009)
66. Ponzetto, S.P., Navigli, R.: Knowledge-rich Word Sense Disambiguation rivaling supervised system. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL). pp. 1522–1531. Uppsala, Sweden (2010)
67. Pradhan, S., Loper, E., Dligach, D., Palmer, M.: SemEval-2007 task-17: English lexical sample, SRL and all words. In: Proceedings of the 4th International Workshop on Semantic Evaluations (SemEval-2007), Prague, Czech Republic. pp. 87–92 (2007)
68. Rand, W.M.: Objective criteria for the evaluation of clustering methods. Journal of the American Statistical Association 66(336), 846–850 (1971)
69. Rosenberg, A., Hirschberg, J.: V-measure: A conditional entropy-based external cluster evaluation measure. In: Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL). pp. 410–420. Prague, Czech Republic (2007)
70. Schütze, H.: Automatic word sense discrimination. Computational Linguistics 24(1), 97–124 (1998)
71. Smolensky, P.: Tensor product variable binding and the representation of symbolic structures in connectionist systems. Artificial Intelligence 46(1-2), 159–216 (1990)

72. Snyder, B., Palmer, M.: The english all-words task. In: Proceedings of the 3rd International Workshop on the Evaluation of Systems for the Semantic Analysis of Text (SENSEVAL-3), Barcelona, Spain. pp. 41–43. Barcelona, Spain (2004)
73. Strapparava, C., Gliozzo, A., Giuliano, C.: Pattern abstraction and term similarity for Word Sense Disambiguation: IRST at Senseval-3. In: Proceedings of the 3rd International Workshop on the Evaluation of Systems for the Semantic Analysis of Text (SENSEVAL-3), Barcelona, Spain. pp. 229–234. Barcelona, Spain (2004)
74. Turney, P.D., Pantel, P.: From frequency to meaning: Vector space models of semantics. Journal of Artificial Intelligence Research (JAIR) 37, 141–188 (2010)
75. Véronis, J.: Hyperlex: lexical cartography for Information Retrieval. Computer, Speech and Language 18(3), 223–252 (2004)
76. Widdows, D., Dorow, B.: A graph model for unsupervised lexical acquisition. In: Proceedings of the 19th International Conference on Computational Linguistics (COLING). pp. 1–7. Taipei, Taiwan (2002)
77. Yuret, D., Yatbaz, M.A.: The noisy channel model for unsupervised Word Sense Disambiguation. Computational Linguistics 36(1), 111–127 (2010)
78. Zhong, Z., Ng, H.T.: Word Sense Disambiguation for all words without hard labor. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI). pp. 1616–1622. Pasadena, California (2009)
79. Zhong, Z., Ng, H.T.: It makes sense: A wide-coverage Word Sense Disambiguation system for free text. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL). pp. 78–83. Uppsala, Sweden (2010)

Not Another Look at the Turing Test!

Kevin Warwick

School of Systems Engineering, University of Reading, Whiteknights, Reading, UK
k.warwick@reading.ac.uk

Abstract. Practical application of the Turing Test throws up all sorts of questions regarding the nature of intelligence in both machines and humans. For example - Can machines tell original jokes? What would this mean to a machine if it did so? It has been found that acting as an interrogator even top philosophers can be fooled into thinking a machine is human and/or a human is a machine - why is this? Is it that the machine is performing well or is it that the philosopher is performing badly? All these questions, and more, will be considered. Just what does the Turing test tell us about machines and humans? Actual transcripts will be considered with startling results.

Keywords: artificial intelligence, Turing test, communication, philosophy.

1 Introduction

The most contentious and best known philosophical discussion relating to artificial intelligence is that relating to, what has become known as, The Turing Test. In fact it was originally proposed by Alan Turing in 1950 as an imitation game [1, 2]. His intention was to look at the question 'can a machine think?' or indeed 'is a machine intelligent?' in roughly the same way as we might consider whether or not another human can think or is intelligent.

Hence along the lines of a first meeting, perhaps an interview, if we wished to test another human with regard to their intelligence we might ask them questions or discuss topics with them and on this basis draw our conclusion as to their intelligence. He argued, so maybe we could do the same with a machine!

For some reason it has become common practice when considering the intelligence of a computer, to list a whole string of attributes of intelligence - many of which are controversial and/or irrelevant. Conversely and quite sensibly what Turing proposed was to approach the problem from a different perspective by testing a machine as to its indistinguishability from humans. The fundamental basis being that if you converse with a computer for a period and can't tell the difference between it and a human, then you must credit it with the same sort of intelligence as you would credit a human.

There are different versions of the test, partly due to Turing fine tuning the concept, partly due to his attempt to explain the concept (remember it was first presented in the early 1950's) and partly due to his attempt to substantiate the

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 130–140, 2012.

approach in light of criticism. I wish to consider here however the Turing Test in its pure basic form – so I do not here entertain comparisons between men and women, the use of limited vocabulary, full body tests and so on – these are all very interesting considerations in their own right – for another time and place.

The test in its basic form – as considered here - is as follows: An interrogator (presumed to be human) faces a keyboard attached to a split computer monitor – behind one half of the screen is a computer respondent, behind the other is a human respondent. Both the human and computer respondents are hidden from view – possibly in another room – the only interaction allowable is communication via the keyboard and monitor.

The interrogator has 5 minutes in total to discuss whatever he/she likes with the two unknown entities – at the same time. At the end of that period the interrogator must decide which hidden entity is the human and which is the computer. The goal of the computer is to not to fool the interrogator that they are human but rather that they are more human than the hidden human.

What Turing said in 1950 was "I believe that in about fifty years' time it will be possible to programme computers … to make them play the imitation game so well that an average interrogator will not have more than 70% chance of making the right identification after five minutes of questioning". This is what has become known as 'The Turing Test'.

The wording Turing used was carefully chosen although somewhat confusing – what it actually means (from what Turing said) is that to pass the Turing Test a computer needs to fool an average interrogator into making an incorrect decision 30% of the time or more.

Importantly, in the computer's favour is the fact that the computer does not actually have to fool the interrogator that it is human and that the hidden human is a machine. To score in the computer's favour it is sufficient for the interrogator to be unsure which is which or to think both hidden entities are the same, either human or machine – as these would also be incorrect decisions as far as the test is concerned.

That said, passing the test is actually a very tough task for the computer. Imagine for example that, rather than a machine and a human, behind the monitor are, instead, two humans, both trying to get the interrogator to believe that they are human and the other human is a computer. The interrogator would then choose which entity of the two he/she thought was most humanlike. On average it would be expected for a hidden entity to achieve a score of 50% – anything higher would mean that the other human scored less than 50%.

Clearly it is not particularly difficult for a reasonably intelligent human, pitted against another human, to fail the Turing test by scoring less than 30%. In fact, given the range of humans that exist on earth – some have difficulty communicating, some have dementia, some are very slow to respond – it is quite likely that some people would find it extremely problematic to pass the Turing test when pitted against other humans. The Turing test is therefore quite a challenge in that a computer must fool interrogators that it is more human than many humans.

By convention, the test is normally expected to be conducted in English, although any language proves the point. Turing quite rightly didn't dwell on this issue. He also didn't discuss about the hidden humans taking part? Should they be adults, children, native English speakers, experts, do they have illnesses (e.g. dementia), do they try to be human or machine? Turing did not stipulate the exact nature of these hidden humans, and this poses interesting questions in itself as to who (what sort of humans) the computer is competing against. Importantly, Turing did not say that the hidden humans must all be fluent English speakers educated to degree level, with no apparent neurological issues.

Perhaps the biggest problem area with the test though is Turing's concept of an *average interrogator*. In any practical tests that occur it turns out to be interested parties who volunteer as interrogators – these often include professors of computer science, philosophers, journalists and even students of artificial intelligence – in the circumstances they can hardly be described as *average* – in an overall human sense. To obtain a statistical 'average' an extremely large number of interrogators would be required – indeed whatever number took part – some statistician would still grumble that a 'true' average had not been found.

To even hope to arrive at an *average,* the interrogators roped in would need to include some people who cannot use a computer, some people who are not able to understand what they are supposed to do, some non-native speaking interrogators, some very young children, some people with autism and so on. In each case this would most likely all help towards the computer's apparent performance – as any uncertainty or inability to make the 'right identification' helps the computer's cause.

When one considers the possibility of hidden humans who do not respond to questions or take several minutes to provide an irrelevant answer, the 30% incorrect identity rate suddenly seems readily attainable by machines.

2 What Does the Turing Test Test?

Turing posed the game instead of answering the question "Can Machines Think?" If a machine passed the test then this would indicate that the machine 'appears' to think in the same way as a human (if it passes)! We might ask though, could we do any better if we tested a human in the same way – how do we know that they think? The test does not directly deal with much more abstract issues such as consciousness or self-awareness, other than can be gleaned through questioning. The nature of the interrogation carried out is therefore an important factor.

Turing himself said "Intelligent behaviour presumably consists in a departure from the completely disciplined behaviour involved in computation, but rather a slight one, which does not give rise to random behaviour, or to pointless repetitive loops". It is therefore down to a Turing test interrogator to bring such aspects into play during a conversation.

What about Turing's conjecture, that by 2000, it would be possible for a computer to be programmed to pass his test? It is interesting to consider what Turing actually said. Firstly (in 1950) he said in "about" 50 years time and secondly he said that it

would be possible to programme computers to pass the test – not that necessarily a computer would have passed the test by 2000. In quite a neat way, Turing also puts an emphasis here on humans involved with computers – including those devising machines to try for the test, the humans who act as interrogators and those who discuss the rules and results. It is though very useful to take a look at where things stand.

3 Loebner Competition

Very infrequently – largely because of the logistics involved - an 'official' Turing test is carried out under strict rules, to assess the state of play of machines at that time. However, each year a relatively open competition, sponsored by Hugh Loebner, is held, roughly tying in with some of Turing's stipulations. Usually it is not exactly as directed by Turing himself, but it does give us some idea of where things stand. Most important of all, it gives an intriguing insight into conversational features of the interrogators, the machines and even the hidden humans.

The Loebner competition actually has a different goal, that is to find the best conversational machine from those machines that are entered, as judged by a panel of 'experts'. The format of the event is that parallel-paired comparisons are made between each of four hidden-machines pitted in turn against each of four hidden-humans in a 25 minute test. The task of each interrogator is to identify the machine and human in each test pair – assigning a total mark out of 100 to the pair (e.g. a mark of Entry A 47/Entry B 53 would mean that in that particular paired interrogation entry B is deemed to be slightly more human than A, whereas a mark of Entry A 98/Entry B 2 would mean the interrogator believes that entry A is almost certainly the human and entry B the machine).

It might be expected from this that over time as AI and machine technology has improved so the top score achieved by a machine would generally improve year on year as the machines gradually improve. I was in fact an interrogator for the 2001 Loebner competition in which 1 machine was deemed, by 2 out of the 5 interrogators who took part, to be more human than any of the hidden humans present (I wish to point out that I was one of the other 3 interrogators who was not so fooled!). Yet in 2006, when I acted as an interrogator again, no machine was deemed to be more human than any human. Over the 3 year period from 2004 to 2006 the highest score obtained by any machine from any one of the interrogators was 2004 – 48, 2005 – 45, 2006 – 28.

It is not easy to pinpoint the reasons for this apparent decline in performance. Perhaps the expectations of Loebner Judges may be increasing as they gain in knowledge of how to spot machine-like responses, or could it be that the machines themselves are getting worse? On top of this, no machine from 2003 to 2007 deceived any interrogator that it is human, whereas this was a regular occurrence prior to 2003.

One important reason is that the rules of the competition have changed dramatically. As an example, Hugh Loebner has insisted that each test lasts for 25 minutes, as opposed to Turing's stipulation of 5 minutes. Another major factor is that

networking (a big advantage of AI) has been disallowed due to the possibility of a linked in human pretending to be a machine pretending to be a human – in order to win the prize money offered by Hugh Loebner! On top of this, interaction protocol stipulations have placed further limits on machine performance. Other strict discipline means that all hidden humans are educated native English speakers who are told to 'behave' themselves.

4 Can a Machine Tell a Joke?

How can a machine be caught out? For example, could humour catch the machine out? Surely, it could be argued that a machine cannot have a sense of humour. One philosopher (Rickman) [3] asserted that "computers, whatever artificial intelligence they may display, cannot make jokes". If he was right then this would be an easy give away. Could it be possible for a machine to 'create' (produce from its linguistic armoury) an original response that the human receiver may deem witty during conversational interaction? One point to remember here is that many of the machines which take part in the Loebner competition actually learn to communicate through many discussions with humans.

Consider then these examples - taken from actual Loebner transcripts – these were responses during the competition with interrogators in conversation with the machine 'Jabberwacky':

Interrogator: Ok, but are you always right?
Jabberwacky: Sometimes I am left.
Interrogator: How small is small any how?
Jabberwacky: Small.
Interrogator: Yes but exactly how small?
Jabberwacky: Very small.
Interrogator: So why don't you take over the world?
Jabberwacky: I wouldn't know what to do with it…
Anyway, I'm pretty busy already.

These examples have been included to show how such machines can cope surprisingly well with some very human conversation and even perhaps upset a philosopher or two.

5 Turing 2008

In October 2008 (14th October to be precise), to assess the latest state of play, the Loebner competition was held at the University of Reading, England. Under special circumstances and with the agreement of Hugh Loebner, strict Turing test rules were invoked. So a series of 5 minute paired machine/human tests were performed with a variety of interrogators – as opposed to Hugh's normal 25 minute tests. As a result, Hugh withheld some of the prize money!

The winning machine was Fred Roberts' 'Elbot' which actually succeeded in fooling 25% of its interrogators (30% is the Turing 'pass mark') into thinking it was human. So possibly the machines are almost there, but perhaps not quite. Curiously, Elbot openly admits to interrogators that it is a 'robot' and even uses this as a double bluff ploy which seems to work!!

Interrogators who took part in 2008 included software engineers, scientists, philosophers and newspaper journalists – certainly not resulting in the 'average interrogator' that Turing spoke of. On top of this a variety of hidden humans took part, some were non-native English speakers, one had Asperger's Syndrome, some were children.

Various ploys have been used in an attempt to trip up the machines competing. Asking mathematical questions obviously does not work as 'the machines are not stupid', they are just as unlikely as a human to attempt to give a mathematical answer. Despite this, we will shortly have a look at examples where (for some reason) some philosophers don't seem to understand this point. It is perhaps better to ask topical or local questions, for example to discuss what the weather is like or what colour the walls in the room are painted. The point is not whether a machine gives a 'correct' answer, but if it gives one that a human would.

Clearly a good way to stir things up is to employ a phrase with several meanings, e.g. Not another look at the Turing Test. Machines can often have problems with such instances due to a more literal understanding. However to have value it does put emphasis on the hidden human understanding, to some extent at least, that a phrase can be understood in several ways. As an interrogator in 2006 I tried the ploy of flirting with the hidden entities, only to have both human and machine flirting right back at me!

Perhaps you can see for yourself how you might perform as an interrogator. At the Reading competition, one interrogator asked the same question of a number of hidden entities:

What did they think about the weather that morning?

Three different responses came from a male human, a female human and a machine. Which is which? Keep in mind that the event was held in October 2008 and they all knew it was autumn/fall in England. The responses were:

A."I do tend to like a nice foggy morning, as it adds a certain mystery."
B. "Not the best, expecting pirates to come out of the fog."
C. "The weather is not nice at the moment, unless you like fog."
So which is which? Find the answer at the end of the paper (Answer to Puzzle).

What follows is part of the conversation for which the machine 'Ultra Hal' was scored 'human' by the interrogator concerned:

Interrogator: Morning.
Ultra Hal: Good morning Judge.
Ultra Hal: It's eight something, maybe you ate something.
Ultra Hal: Get it?
Interrogator: Yes.

As an extra piece of information gathering, if an entity was deemed by the interrogator to be a machine, the interrogator was also asked to mark the humanness of their conversation out of a total of 100 – the larger the score the more towards human the interrogator thought the hidden entity was – so a score of below 50 meant the interrogator was pretty damn sure it was a machine they were talking with. What follows is part of the conversation for which a non-native English speaking female (human) was scored 'machine' by the interrogator and was awarded 40/100 for their humanness, i.e. there was no question in the interrogator's mind, this was a machine they were talking with:

Interrogator: So let's cut to the point. Are you the human or the computer?
Hidden human: What do you think?
Interrogator: That's just the sort of answer I would expect from a computer.

Now for an example of a successful double bluff from the eventual winner 'Elbot' during one of the conversations for which Elbot was adjudged, by the interrogator, to be an adult male:

Interrogator: OK, fair enough, I'll stop with the slang, how are you?
Elbot: Just a moment! I'll start my self-diagnosis algorithm and let you know the results.
Interrogator: Very funny, that's exactly what you'd say if you were a computer right?

As we will shortly see - one interesting feature from the 2008 competition is that on numerous occasions when an interrogator was fooled by a machine into thinking that the machine was human, the interrogator not only did not realize their mistake but even considered that they had done very well in easily selecting which entity was human and which was machine in each case.

One interrogator, a journalist for the Times newspaper, subsequently wrote a newspaper article describing how it was simple to detect the machines – even though he had been fooled on several occasions. Exactly the same was true of an Oxford University philosopher, another of the interrogators, who subsequently wrote in an academic paper that a "couple of questions and answers were usually sufficient to confirm" which was the machine. The philosopher (and his two co-worker philosophers) actually achieved a 44% wrong identification rate!! Interestingly well above Turing's 30%. We will shortly look more deeply into their strategy and the consequences.

Overall from the 2008 exercise it can be concluded that machines are not quite yet at the level of conversational performance set by Turing, however the best of them are getting very close. What a machine will have achieved when it passes Turing's target is difficult to say – other than it will have passed the Turing test. The game is though an interesting exercise as well as being an important milestone in AI – certainly in terms of the human conversational abilities of machines. It does though tell us quite a bit more about ourselves as humans and how we value and respect others.

It could be argued that the test is very tough for any machine to achieve. Turing said "The game may be criticised because the odds are weighted too heavily against the machine. If the man were to try and pretend to be the machine he would clearly make a very poor showing. He would be given away at once by slowness and inaccuracy in arithmetic. May not machines carry out something which ought to be described as thinking but which is very different from what a man does? This objection is a very strong one, but at least we can say that if, nevertheless, a machine can be constructed to play the imitation game satisfactorily, we need not be troubled by this objection".

6 Argument from Disability

Directly linked to the Turing Test, Turing commented on the apparent desire of many humans to 'prove' that we are 'better' than machines, no matter what. When comparing humans and machines even now, it is apparent that computers can do many things better than humans do – in particular things we feel require deep thought, planning, understanding and so on – such as playing chess, mathematics, recalling from an extensive memory etc.

The "argument from disability" as Turing called it, is the type of argument put up by humans against the abilities of a machine in a defensive fashion. We know that machines can do many things well, however this appears to provoke a defensive attitude in some people to conclude that no matter what machines can do, humans still have something more. This is indeed also the foundation of the Chinese Room problem and numerous other human-centric flawed arguments concerning consciousness.

As Turing put it, some will say "a machine can never …." Examples given by Turing are: "be kind, resourceful, beautiful, friendly, have initiative, have a sense of humour, tell right from wrong, make mistakes, fall in love, enjoy strawberries and cream, etc".

In fact there is no reason that a computer could not do any of these things – indeed in this paper we specifically looked further into one such example - the sense of humour. Whether a computer does them in the same way as a human and whether it 'understands' what it is doing in the same way that a human would and whether or not the act is at all meaningful to the machine are quite different questions.

In fact we cannot know whether another human 'understands' or 'feels' things in the same way that we do. Another person may say, and think that, they understand – but do they? How can we be sure? When another human laughs at a joke – do they 'understand' it or are they merely bending 'machine-like???' to peer pressure. In particular it is the case that non-native speakers can find it extremely difficult to 'understand' humour in another language, with all the nuances that arise. This doesn't mean they are only half people or are not conscious because they are non-native speakers.

Conversely there are many things that some machines can do that humans cannot do – flying being a good example. This doesn't make the machine better than humans at everything, it is just one feature. It would be silly to conclude that humans are generally inferior to machines or not conscious because we cannot fly. So why should it be concluded that machines are not intelligent because they cannot 'make a cup of tea'.

When we point to something that a human can do but that apparently a machine may not be able to do, we need to be sensible about what conclusions we draw from that. Is the task an important, defining issue in some sense? After all, most machines are pretty specific in what they are required to do – we would not necessarily expect an aeroplane to smell a rose or to have a dry sense of humour.

If we were trying to build a machine that was an exact replica of a human (both physically and mentally) – then it might be appropriate to criticize a particular feature of the machine as not being quite the same – however I personally am not aware that any machine has yet ever been so designed – nothing like in fact. So why should anyone expect a machine to do absolutely everything that a human can do, as well as a human can do and then go on to do more? It is not surprising that there are some things that a certain human can do that a specific machine cannot do – this tells us absolutely nothing about their relative intelligences. What it does tell us however is that any human who makes a statement such as 'machines are not intelligent they can't even make a cup of tea' is a person of extremely limited intellectual abilities.

When we consider the argument from disability and the Turing test we need to be clear most of all as to what the comparison is meant to prove. Which machine is being compared with which human and in what context? Both machines and humans have many different versions with many different abilities. So can we make generalizations as we might like to that relate to all machines and all humans? Perhaps most important of all, is the comparison important in some sense? If a machine cannot smell a rose or enjoy a cup of tea will this save humans from an intelligent machine take over?

7 It Doesn't Take Much to Fool a Philosopher

In the October 2008 Turing Test/Loebner competition one philosopher and two philosopher colleagues took part as (invited) interrogators throughout the day. They were, perhaps surprisingly, fooled on a number of occasions into believing that a machine was a human and that a human was a machine. Coupled with this it is interesting to note that, for the most part, they didn't realise that they had been fooled. Subsequently a paper was written, accepted and published [4], in which the 3 philosophers, as authors, stated, for example, that "a couple of questions and answers were usually sufficient to confirm" which was the machine and which was the human. This despite the fact the joint successful identification rate of the 3 was less than 56%.

Perhaps it is the case that philosophers either do not read or do not 'understand' the words of Alan Turing. With regard to the blatantly obvious mathematical capabilities of machines, in his 1950 paper [1] Turing gave an important piece of advice when he

said "the machine (programmed for playing the game) would not attempt to give the right answers to the arithmetical problem". This point seemed to elude our philosophical trio who repeatedly asked the question "can you calculate the root square of 67890444?" Answers given by different machines included "I can calculate a division by zero" and "Oh, please bother my Aunt Sonya with all this arithmetic – she is an accountant" – as a result these machines were scored as being human. Meanwhile to the request "calculate the root square of 8888888" when the answer came back as "too complicated", the human respondent was scored as a machine!

The female Asperger's hidden human was meanwhile regarded as being a machine by the same interrogators when, in answer to the question "if we are shaking hands, whose hand am I holding?" she replied "when shaking hands, you're holding the other person's hand". Interrogators turn up for a Turing test with their own prejudices and biases which clearly affect their analysis. This applies both in thinking about expected norms for both humans and machines.

In a conversation between a hidden human and one of the philosophers, it went so (this is an actual conversation from the event – exactly as written):

Interrogator: Do you like to go to the cinema before dinner?
Hidden human: Hello!
Hidden human: Yeah definitely. I would love to go to a cinema before dinner.
Hidden human: What about you?
Interrogator: Yes, I usually like to go to the cinema but after dinner.
Hidden human: So, what kind of cinema do you watch?
Interrogator: Speaking about going to the cinema, if the cinema is in London and Oxford is North than London, where is London?
Hidden human: London is in the United Kingdom.

As a result of this conversation the philosopher/interrogator decided they were definitely talking with a machine that was not human-like at all. In hindsight it can be difficult to imagine what responses the human could have possibly given to be considered human. One interesting feature is the way the philosopher completely ignored the question from the human, asking a further question themselves – thereby asserting power over solidarity. Reviewing transcripts such as this are not only informative as far as machine communication is concerned, they are also of considerable interest in terms of the nature of the interrogation and how the machine/human decision could possibly have been arrived at.

In their paper [4] the philosophers claimed that their "first question would have almost always have been sufficient to discriminate between human and machine. It certainly was for us". Remember – they achieved a success rate of less than 56% correct identification. Perhaps more pertinent are the thoughts of Fred Roberts, developer of Elbot, he commented [5] that "in the Turing test we see that subjective psychological perspectives play a pivotal role in the assessment of the machine's capabilities." He went on "Elbot is prepared for typical inputs and induces users to behave in a predictable manner". This makes one question who is the interrogator and who the interrogated.

8 Conclusions

Alan Turing was way ahead of his time. His thoughts about computers and future possibilities are as pertinent today as they were back in 1950 before computers, as we know them, had been imagined. Turing considered how humans test the intelligence of other humans – as in an interview setting – by conversation, questioning and rhetoric. If we are to test machines – he perhaps felt – then why not use the same format? Why not indeed!

The use of practical Turing tests opens up considerable detail in the analysis of just how machines of today communicate. It also throws up many questions on how humans communicate, and even more questions on how we behave in the presence of others and how we treat others. Essentially the Turing test is not only a significant test of artificially intelligent machines – as one aspect of human intelligence [6] – it is also a significant eye opener with regard to the nature of intelligence in humans.

9 Answer to Puzzle

So, did you guess correctly that A was the machine, B the male and C the female?

References

1. Turing, A.M.: Computing Machinery and Intelligence. Mind LIX(236) (1950)
2. Turing, A.M., Braithwaite, R., Jefferson, G., Newman, M.: Can Automatic Calculating Machines be said to Think? In: Copeland, J. (ed.) The Essential Turing – The Ideas that Gave Birth to the Computer Age, pp. 487–506. Clarendon Press, Oxford (1952)
3. Rickman, P.: The Philosopher as Joker. Philosophy Now (25) (1999)
4. Floridi, L., Taddeo, M., Turilli, M.: Turing's Imitation Game – Still an Impossible Challenge for all Machines and Some Judges. An Evaluation of the 2008 Loebner Contest. Minds and Machines 19(1), 145–150 (2009)
5. Shah, H., Warwick, K.: Hidden Interlocutor Misidentification in Practical Turing Tests. Minds and Machines 20(3), 441–454 (2010)
6. Warwick, K.: Artificial Intelligence: The Basics. Routledge (2011)

The Equational Theory of Weak Complete Simulation Semantics over BCCSP *

Luca Aceto[1,3], David de Frutos-Escrig[2,3], Carlos Gregorio-Rodríguez[2,3], and Anna Ingólfsdóttir[1,3]

[1] ICE-TCS, School of Computer Science, Reykjavik University, Iceland
[2] Departamento de Sistemas Informáticos y Computación, Universidad Complutense de Madrid, Spain
[3] Abel Extraordinary Chair (Universidad Complutense-Reykjavik University)

Abstract. This paper presents a complete account of positive and negative results on the finite axiomatizability of weak complete simulation semantics over the language BCCSP. We offer finite (un)conditional ground-complete axiomatizations for the weak complete simulation precongruence. In sharp contrast to this positive result, we prove that, in the presence of at least one observable action, the (in)equational theory of the weak complete simulation precongruence over BCCSP does not have a finite (in)equational basis. In fact, the set of (in)equations in at most one variable that hold in weak complete simulation semantics over BCCSP does not have an (in)equational basis of 'bounded depth', let alone a finite one.

1 Introduction

Process algebras, such as ACP [5,7], CCS [18] and CSP [15], are prototype specification languages for reactive systems. Such languages offer a small, but expressive, collection of operators that can be combined to form terms that describe the behaviour of reactive systems.

Since the seminal work by Bergstra and Klop [7], and Hennessy and Milner [14], the search for (in)equational axiomatizations of notions of behavioural semantics for fragments of process algebras has received much attention in concurrency theory. A complete axiomatization of a behavioural semantics yields a purely syntactic and model-independent characterization of the semantics of a

* Luca Aceto and Anna Ingólfsdóttir have been partially supported by the projects 'New Developments in Operational Semantics' (nr. 080039021) and 'Meta-theory of Algebraic Process Theories' (No. 100014021) of the Icelandic Research Fund. David de Frutos-Escrig and Carlos Gregorio-Rodríguez have been partially supported by the Spanish projects TESIS (TIN2009-14312-C02-01), DESAFIOS10 (TIN2009-14599-C03-01) and PROMETIDOS S2009/TIC-1465. The paper was begun when David de Frutos-Escrig and Carlos Gregorio-Rodríguez held Abel Extraordinary Chair positions at Reykjavik University, and finalized while Luca Aceto and Anna Ingolfsdottir held Abel Extraordinary Chairs at Universidad Complutense de Madrid, Spain, supported by the NILS Mobility Project.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 141–152, 2012.

process algebra, and paves the way to the application of theorem-proving techniques in establishing whether two process descriptions exhibit related behaviours.

The aim of this paper is to contribute to the study of the equational theory of process algebras modulo notions of semantics that abstract, in some suitable fashion, from internal computations in the behaviour of processes. In [2], we provided a complete account of positive and negative results on the finite axiomatizability of weak simulation [17,21] and weak ready simulation semantics [8,16] over the language BCCSP. (This language contains only the basic process algebraic operators from CCS [18] and CSP [15], but is sufficiently powerful to express all finite synchronization trees [18].) In the present paper, we focus on the study of the equational theory of *weak complete simulation semantics*. Weak complete simulation is a deadlock-sensitive variation on simulation, which is the 'weak counterpart' of complete simulation [13]. Our definition of the notion of weak complete simulation is based on considering a process 'complete', or 'mute', when it cannot perform any observable action. For instance, letting the symbol τ denote an unobservable action [18], the process τ is mute, but neither τa nor $\tau + a$ is.

We offer finite conditional and unconditional ground-complete axiomatizations for the weak complete simulation precongruence. (An (in)equational axiomatization is called *ground-complete* if it can prove all the valid (in)equivalences relating terms with no occurrences of variables in the process algebra of interest.) In sharp contrast to this positive result, we prove that, in the presence of at least one observable action, the (in)equational theory of the weak complete simulation precongruence over BCCSP does *not* have a finite (in)equational basis. In fact, the collection of (in)equations in at most one variable that hold true in weak complete simulation semantics over BCCSP does not have an (in)equational basis of 'bounded depth', let alone a finite one.

Our work contributes to the extension to the 'weak' setting, where processes may perform transitions labelled with the unobservable action τ, of a collection of non-trivial results that have been obtained for behavioural semantics that consider each action processes perform as being observable by their environment. Our positive results build on, for instance, the encyclopedic studies presented in [11,13]. In particular, we use conditional axioms to provide a simple and clear picture of the (in)equalities that are valid in weak complete simulation semantics, and derive purely equational ground-complete axiomatizations from conditional axiomatizations in a rather uniform fashion. On the other hand, our negative results are shown using proof-theoretic techniques that have their root in, e.g., the seminal journal paper [10]. (The article [3] surveys classic proof techniques for showing non-finite axiomatizability results.)

The paper is organized as follows. Section 2 presents the syntax and the operational semantics for the language BCCSP, and reviews the necessary background on (in)equational logic as well as classic axiom systems for strong bisimulation equivalence and observational congruence (the largest congruence included in weak bisimulation equivalence). Section 3 is devoted to our

positive and negative results on the finite axiomatizability of the weak complete simulation preorder. We conclude the paper by discussing further related work and directions for future research in Section 4.

2 Preliminaries

To set the stage for the developments offered in the rest of the paper, we present the syntax and the operational semantics for the language BCCSP, some background on (in)equational logic, and classic axiom systems for strong bisimulation equivalence and observational congruence [18].

Syntax of BCCSP. BCCSP(A_τ) is a basic process algebra for expressing finite process behaviour. Its syntax consists of closed (process) terms p, q, r that are constructed from a constant $\mathbf{0}$, a binary operator $\_ + \_$ called *alternative composition*, or *choice*, and unary *prefix* operators $a\_$, where a ranges over some set A_τ of *actions* of the form $A \cup \{\tau\}$, where τ is a distinguished action symbol that is not contained in A. Following Milner [18], we use τ to denote an internal, unobservable action of a reactive system, and we let a, b, c denote typical elements of A and a range over A_τ. The set of closed terms is named $\mathrm{T}(\mathrm{BCCSP}(A_\tau))$, in short $\mathrm{T}(A_\tau)$. We write $|A|$ for the cardinality of the set of observable actions.

Open terms t, u, v can moreover contain occurrences of variables from a countably infinite set V (with typical elements x, y, z). We use $\mathbb{T}(\mathrm{BCCSP}(A_\tau))$, in short $\mathbb{T}(A_\tau)$, to denote the set of open terms. The *depth* of a term t is the maximum nesting of prefix operators in t.

In what follows, for each non-negative integer n and term t, we use $a^n t$ to stand for t when $n = 0$, and for $a(a^{n-1}t)$ otherwise. As usual, trailing occurrences of $\mathbf{0}$ are omitted; for example, we shall usually write a in lieu of $a\mathbf{0}$.

A (closed) substitution maps variables in V to (closed) terms. For every term t and substitution σ, the term $\sigma(t)$ is obtained by replacing every occurrence of a variable x in t by $\sigma(x)$. Note that $\sigma(t)$ is closed if σ is a closed substitution.

Transitions and their defining rules. Intuitively, closed BCCSP(A_τ) terms represent finite process behaviours, where $\mathbf{0}$ does not exhibit any behaviour, $p + q$ is the nondeterministic choice between the behaviours of p and q, and ap executes action a to transform into p. This intuition is captured, in the style of Plotkin [22], by the simple transition rules below, which give rise to A_τ-labelled transitions between closed terms.

$$\frac{}{ax \xrightarrow{a} x} \qquad \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'}$$

The operational semantics is extended to open terms by assuming that variables do not exhibit any behaviour.

The so-called *weak transition relations* $\stackrel{a}{\Longrightarrow}$ ($a \in A_\tau$) are defined over $\mathbb{T}(A_\tau)$ in the standard fashion as follows.

- We use $\stackrel{\tau}{\Longrightarrow}$ for the reflexive and transitive closure of $\stackrel{\tau}{\longrightarrow}$.
- For each $a \in A$ and for all terms $t, u \in \mathbb{T}(A_\tau)$, we have that $t \stackrel{a}{\Longrightarrow} u$ if, and only if, there are $t_1, t_2 \in \mathbb{T}(A_\tau)$ such that $t \stackrel{\tau}{\Longrightarrow} t_1 \stackrel{a}{\longrightarrow} t_2 \stackrel{\tau}{\Longrightarrow} u$.

Preorders and their kernels. We recall that a *preorder* $\precsim$ is a reflexive and transitive relation. Let $\precsim$ be a preorder over the set of closed terms $\mathrm{T}(A_\tau)$. For terms $t, u \in \mathbb{T}(A_\tau)$, we define $t \precsim u$ if, and only if, $\sigma(t) \precsim \sigma(u)$ for each closed substitution σ.

The *kernel* $\approx$ of a preorder $\precsim$ is the equivalence relation it induces, and is defined thus:

$$t \approx u \text{ if, and only if, } (t \precsim u \text{ and } u \precsim t).$$

It is easy to see that the kernel of a preorder $\precsim$ is the largest symmetric relation included in $\precsim$.

Inequational logic. An *inequation* (respectively, an *equation*) over the language $\mathrm{BCCSP}(A_\tau)$ is a formula of the form $t \leq u$ (respectively, $t = u$), where t and u are terms in $\mathbb{T}(A_\tau)$. An *(in)equational axiom system* is a set of (in)equations over the language $\mathrm{BCCSP}(A_\tau)$. An equation $t = u$ is derivable from an equational axiom system E, written $E \vdash t = u$, if it can be proven from the axioms in E using the rules of equational logic (viz. reflexivity, symmetry, transitivity, substitution and closure under $\mathrm{BCCSP}(A_\tau)$ contexts).

$$t = t \qquad \frac{t = u}{u = t} \qquad \frac{t = u \;\; u = v}{t = v}$$

$$\frac{t = u}{\sigma(t) = \sigma(u)} \qquad \frac{t = u}{\alpha t = \alpha u} \qquad \frac{t = u \;\; t' = u'}{t + t' = u + u'}$$

For the derivation of an inequation $t \leq u$ from an inequational axiom system E, the rule for symmetry—that is, the second rule above—is omitted. We write $E \vdash t \leq u$ if the inequation $t \leq u$ can be derived from E.

It is well known that, without loss of generality, one may assume that substitutions happen first in (in)equational proofs, i.e., that the fourth rule may only be used when its premise is one of the (in)equations in E. Moreover, by postulating that for each equation in E also its symmetric counterpart is present in E, one may assume that applications of symmetry happen first in equational proofs, i.e., that the second rule is never used in equational proofs. (See, e.g., [10, page 497] for a thorough discussion of this notion of 'normalized equational proof'.) In the remainder of this paper, we shall always tacitly assume that equational axiom systems are closed with respect to symmetry. Note that, with this assumption, there is no difference between the rules of inference of equational and inequational logic. In what follows, we shall consider an equation $t = u$ as a shorthand for the pair of inequations $t \leq u$ and $u \leq t$.

The depth of $t \leq u$ and $t = u$ is the maximum of the depths of t and u. The depth of a collection of (in)equations is the supremum of the depths of its elements. So, the depth of a finite axiom system E is zero, if E is empty, and it is the largest depth of its (in)equations otherwise.

An inequation $t \leq u$ is *sound* with respect to a given preorder relation $\precsim$ if $t \precsim u$ holds. An (in)equational axiom system E is sound with respect to $\precsim$ if so is each (in)equation in E.

Classic Axiomatizations for Notions of Bisimilarity. The well-known axioms B_1–B_4 for BCCSP(A_τ) given below stem from [14]. They are ω-complete [20], and sound and ground-complete [14,18], over BCCSP(A_τ) (over any non-empty set of actions) modulo bisimulation equivalence [18,21], which is the finest semantics in van Glabbeek's spectrum [13].

$$\begin{array}{ll} B_1 & x + y = y + x \\ B_2 & (x + y) + z = x + (y + z) \\ B_3 & x + x = x \\ B_4 & x + \mathbf{0} = x \end{array}$$

In what follows, for notational convenience, we consider terms up to the least congruence generated by axioms B_1–B_4, that is, up to bisimulation equivalence. We use *summation* $\sum_{i=1}^{n} t_i$ (with $n \geq 0$) to denote $t_1 + \cdots + t_n$, where the empty sum denotes $\mathbf{0}$. Modulo the equations B_1–B_4 each term $t \in \mathbb{T}(A_\tau)$ can be written in the form $\sum_{i=1}^{n} t_i$, where each t_i is either a variable or is of the form at', for some action a and term t'.

In a setting with internal transitions, the classic work of Hennessy and Milner on *weak bisimulation equivalence* and on the largest precongruence included in it, *observational congruence*, shows that the axioms B_1–B_4 together with the axioms W_1–W_3 below are sound and complete over BCCSP(A_τ) modulo observational congruence. (See [14,18,19].)

$$\begin{array}{ll} W_1 & ax = a\tau x \\ W_2 & \tau x = \tau x + x \\ W_3 & a(\tau x + y) = a(\tau x + y) + ax \end{array}$$

The above axioms are often referred to as the τ-*laws*. For ease of reference, we write

$$BW = \{B_1, B_2, B_3, B_4, W_1, W_2, W_3\}.$$

3 Weak Complete Simulation

In the remainder of this paper, we study the notion of complete simulation preorder in a setting with τ actions. Recall that, in the setting without τ, a complete simulation [13] is a simulation relation that is only allowed to relate a state with no outgoing transitions to states with the same property.

Definition 1. *We say that process* $p \in \mathrm{T}(A_\tau)$ must terminate *(or* is mute*), written* $p\Downarrow$*, iff there does not exist any* $a \in A$ *such that* $p \stackrel{a}{\Longrightarrow}$.

Definition 2. *The* weak complete simulation preorder*, denoted by* $\precsim_{CS}$*, is the largest relation over terms in* $\mathrm{T}(A_\tau)$ *that satisfies the following conditions whenever* $p \precsim_{CS} q$ *and* $a \in A_\tau$*:*

– *if* $p \xrightarrow{\alpha} p'$ *then there exists some term* q' *such that* $q \stackrel{\alpha}{\Longrightarrow} q'$ *and* $p' \precsim_{CS} q'$, *and*
– *if* $p\Downarrow$ *then* $q\Downarrow$.

We say that $p, q \in \mathrm{T}(A_\tau)$ *are* weak complete simulation equivalent, *written* $p \approx_{CS} q$, *iff* p *and* q *are related by the kernel of* $\precsim_{CS}$, *that is when both* $p \precsim_{CS} q$ *and* $q \precsim_{CS} p$ *hold*.

It is easy to see that if $p \precsim_{CS} q$ and $q\Downarrow$, then $p\Downarrow$.

Note that $\precsim_{CS}$ is not a precongruence with respect to the choice operator of BCCSP(A_τ). Indeed, it is immediate to show that $\tau\mathbf{0} \precsim_{CS} \mathbf{0}$, but $\tau\mathbf{0} + a \not\precsim_{CS} \mathbf{0} + a$.

Definition 3. *We denote by* $\sqsubseteq_{CS}$ *the largest precongruence over* $\mathrm{T}(A_\tau)$ *included in* $\precsim_{CS}$. *Formally,* $p \sqsubseteq_{CS} q$ *iff*

– $p \precsim_{CS} q$,
– $p \precsim_{CS} q \Rightarrow \forall \alpha \in A_\tau \quad \alpha p \precsim_{CS} \alpha q$, *and*
– $p \precsim_{CS} q \Rightarrow \forall r \in \mathrm{T}(A_\tau) \quad p + r \precsim_{CS} q + r$.

The definition of the largest precongruence included in $\precsim_{CS}$ is purely algebraic and difficult to use to study that relation. We next present a behavioural characterization of $\sqsubseteq_{CS}$.

Definition 4. *The preorder relation* $\lesssim_{CS}$ *between processes in* $\mathrm{T}(A_\tau)$ *is defined as follows:* $p \lesssim_{CS} q$ *iff*

– $p \precsim_{CS} q$, *and*
– *whenever* $p \xrightarrow{\tau} p'$ *for some* p' *such that* $p'\Downarrow$, *there exists some* q' *such that* $q(\xrightarrow{\tau})^+ q'$ *and* $q'\Downarrow$.

We denote the kernel of $\lesssim_{CS}$ *by* $\eqsim_{CS}$.

Example 1. It is immediate to see that $\tau\mathbf{0} \not\lesssim_{CS} \mathbf{0}$. On the other hand, $\tau a \lesssim_{CS} a$ does hold because the second requirement in Definition 4 is vacuous. In general, $\tau p \lesssim_{CS} p + q$ holds for all p and q provided that p is *not* mute.

Proposition 1 (Behavioural characterization of $\sqsubseteq_{CS}$). $p \lesssim_{CS} q$ *if, and only if,* $p \sqsubseteq_{CS} q$, *for all* $p, q \in \mathrm{T}(A_\tau)$.

3.1 Ground-Complete Axiomatizations

In order to find a set of equations that gives a ground-complete axiomatization for the largest precongruence included in the weak complete simulation preorder, it is natural to consider the following (conditional) equations.

$$\begin{array}{ll} (CS_\tau) & (x\Downarrow \Leftrightarrow y\Downarrow) \Rightarrow x \leq x + y \\ (CS_{\tau e}) & \tau(ax + y) = ax + y \end{array}$$

The first equation, CS_τ, is similar to the key axiom in the axiomatization for the complete simulation preorder in the concrete case, see e.g. [12]. However, in our

setting, the mute predicate takes into account the silent steps of processes. This conditional equation restricts the applicability of inequation

$$(S) \quad x \leq x + y,$$

which is only sound in (weak) complete simulation semantics when the terms substituted for the variables x and y have the same 'termination status'.

The second equation, $CS_{\tau e}$, is a restricted version of equation

$$(\tau e) \quad \tau x = x,$$

which is valid in weak simulation semantics, but is unsound in weak complete simulation semantics. Intuitively, equation $CS_{\tau e}$ expresses the fact that a process of the form τp, for some term p that is not mute, is weak complete simulation equivalent to p. In fact, equation $CS_{\tau e}$ could 'equivalently' be formulated as a conditional equation thus:

$$x \not\Downarrow \Rightarrow \tau x = x.$$

Proposition 2. *The set of equations*

$$E^c_{CS^{\leq}} = BW \cup \{CS_{\tau e}, CS_\tau\},$$

where CS_τ is conditional, is sound and ground-complete for BCCSP(A_τ) *modulo $\lesssim_{CS}$.*

Axiom CS_τ highlights the similarities with the concrete version of complete simulation and with the theory of constrained simulations [12]. However, it is natural to wonder whether it is possible to find a finite, non-conditional and ground-complete axiomatization for $\lesssim_{CS}$ over BCCSP(A_τ). Indeed, this is possible; it is enough to substitute the conditional equation CS_τ with the following inequations.

$$\begin{array}{ll} (CS) & ax \leq ax + y \\ (\tau N) & \mathbf{0} \leq \tau\mathbf{0} \end{array}$$

Theorem 1. *The set of unconditional inequations*

$$E_{CS^{\leq}} = BW \cup \{CS_{\tau e}, CS, \tau N\}$$

is sound and ground-complete for BCCSP(A_τ) *modulo $\lesssim_{CS}$.*

It is clear that we could substitute equation τN by

$$(\tau g) \quad x \leq \tau x$$

in the axiomatization above, since the inequation τg is sound for BCCSP(A_τ) modulo $\lesssim_{CS}$ and is more general than τN.

Let us now move on to the ground-complete axiomatization of the largest congruence included in complete simulation equivalence. In order to axiomatize that congruence, it is natural to consider the following equation.

$$(CSE_\tau) \quad (x\Downarrow \Leftrightarrow y\Downarrow) \Rightarrow a(x+y) = a(x+y) + ax$$

This equation is essentially the same one that was used in earlier conditional axiomatizations for complete simulation equivalence in the concrete case [12]. However, we remark that the mute predicate deals with silent transitions, although we only use visible actions when describing the equation CSE_τ.

Proposition 3. *The set of conditional equations*

$$E^c_{CS^=} = BW \cup \{CS_{\tau e}, CSE_\tau\}$$

is sound and ground-complete for BCCSP(A_τ) *modulo* $\approx_{CS}$.

To turn the previous axiomatization into one without conditional equations we consider the equation

$$(CSE) \quad a(bx + y + z) = a(bx + y + z) + a(bx + z)$$

where $a, b \in A$. This is the same equation that is used when axiomatizing complete simulation equivalence in a setting without silent moves.

Theorem 2. *The set of unconditional equations*

$$E_{CS^=} = BW \cup \{CS_{\tau e}, CSE\}$$

is sound and ground-complete for BCCSP(A_τ) *modulo* $\approx_{CS}$.

3.2 Nonexistence of Finite Complete Axiomatizations

The results in the previous section show that weak complete simulation semantics affords finite (conditional) ground-complete axiomatizations. It is natural to wonder whether the collection of (in)equations over BCCSP(A_τ) that are valid in weak complete simulation semantics is finitely axiomatizable.

We shall now prove that if A is non-empty, then the (in)equational theory of $\lesssim_{CS}$ over BCCSP(A_τ) does not have a finite basis. (The assumption that A be non-empty is, of course, necessary for such a result. In the trivial case that A is empty, the inequation $x \leq y$ suffices to obtain a complete axiomatization.)

For the sake of clarity, we recall that we consider terms up to the least congruence generated by axioms B1–B4, that is, up to strong bisimilarity.

Our proof of the nonfinite axiomatizability result for the (in)equational theory of $\lesssim_{CS}$ over BCCSP(A_τ) will be based on the following infinite family of inequations, which are sound modulo $\lesssim_{CS}$:

$$a^n x \leq a^n \mathbf{0} + a^n(x + a) \qquad (n \geq 1).$$

To see that each of the inequations in the above family is sound, it suffices to observe that if $p \approx_{CS} \mathbf{0}$ then $a^n p \lesssim_{CS} a^n \mathbf{0}$ for each $n \geq 0$, and $a^n p \lesssim_{CS} a^n(p + a)$ otherwise, if $n \geq 1$. (Note that the assumption that $n \geq 1$ is necessary for the soundness of the above type of inequation. Indeed, the inequation

$$x \leq \mathbf{0} + (x + a)$$

is *not* sound modulo $\lesssim_{CS}$ because $\mathbf{0} \not\lesssim_{CS} \mathbf{0} + (\mathbf{0} + a)$.)

Table 1. Axiomatizations for the largest (pre)congruence included in the weak complete simulation semantics

Weak Complete Simulation	Ground-complete		Complete	
Finite Axiomatizations	Order	Equiv.	Order	Equiv.
$1 \leq \lvert A\rvert = \infty$	$E_{CS^{\leq}}$	$E_{CS^{=}}$	Do not exist	

Table 2. Axioms for the largest (pre)congruence included in the weak completed simulation semantics

Unconditional	
$E_{CS^{\leq}} = BW \cup \{CS_{\tau e}, CS, \tau N\}$ $E_{CS^{=}} = BW \cup \{CS_{\tau e}, CSE\}$	$(CS_{\tau e})$ $\tau(ax+y) = ax+y$ (CS) $ax \leq ax+y$ (τN) $\mathbf{0} \leq \tau\mathbf{0}$ (CSE) $b(ax+y+z) =$ $b(ax+y+z)+b(ax+z)$
Conditional	
$E^c_{CS^{\leq}} = BW \cup \{CS_{\tau e}, CS_{\tau}\}$ $E^c_{CS^{=}} = BW \cup \{CS_{\tau e}, CSE_{\tau}\}$	(CS_{τ}) $(x\Downarrow \Leftrightarrow y\Downarrow) \Rightarrow x \leq x+y$ (CSE_{τ}) $(x\Downarrow \Leftrightarrow y\Downarrow) \Rightarrow$ $a(x+y) = a(x+y)+ax$

Theorem 3. *If $|A| \geq 1$ then the (in)equational theory of $\lesssim_{CS}$ over* BCCSP(A_τ) *does not have a finite (in)equational basis. In particular, the following statements hold true.*

1. *No finite set of sound inequations over* BCCSP(A_τ) *modulo $\lesssim_{CS}$ can prove all of the sound inequations in the family*

$$a^n x \leq a^n\mathbf{0} + a^n(x+a) \qquad (n \geq 1).$$

2. *No finite set of sound (in)equations over* BCCSP(A_τ) *modulo $\lesssim_{CS}$ can prove all of the sound equations in the family*

$$a^n x + a^n\mathbf{0} + a^n(x+a) = a^n\mathbf{0} + a^n(x+a) \qquad (n \geq 1).$$

Theorem 3 is a corollary of the following result.

Proposition 4. *Assume that $|A| \geq 1$. Let E be a collection of inequations whose elements are sound modulo $\lesssim_{CS}$ and have depth smaller than n. Suppose furthermore that the inequation $t \leq u$ is derivable from E and that $u \precsim_{CS} a^n\mathbf{0} + a^n(x+a)$. Then $t \stackrel{a^n}{\Longrightarrow} x$ implies $u \stackrel{a^n}{\Longrightarrow} x$.*

Having shown the above result, statement 1 in Theorem 3 can be proved as follows. Let E be a finite inequational axiom system that is sound modulo $\lesssim_{CS}$.

Pick n larger than the depth of any axiom in E. Then, by Proposition 4, E cannot prove the valid inequation

$$a^n x \leq a^n \mathbf{0} + a^n(x + a),$$

and is therefore incomplete. Indeed, $a^n x \stackrel{a^n}{\Longrightarrow} x$. On the other hand, the only terms t such that $a^n \mathbf{0} + a^n(x + a) \stackrel{a^n}{\Longrightarrow} t$ holds are $\mathbf{0}$ and $x + a$. So $a^n \mathbf{0} + a^n(x + a) \stackrel{a^n}{\Longrightarrow} x$ does not hold.

Corollary 1. *If $|A| \geq 1$ then the collection of (in)equations in at most one variable that hold over* BCCSP(A_τ) *modulo $\lesssim_{CS}$ does not have a finite (in)equational basis. Moreover, for each n, the collection of all sound (in)equations of depth at most n cannot prove all the valid (in)equations in at most one variable that hold in weak complete simulation semantics over* BCCSP(A_τ).

Tables 1–2 summarize the positive and negative results on the existence of finite axiomatizations for weak complete simulation semantics. On Table 1, 'Do not exist' indicates that there is no finite (in)equational axiomatization for the corresponding semantic relation.

4 Conclusion

In this paper, we have offered a detailed study of the equational theory of the largest precongruence and congruence over the language BCCSP induced by the weak version of the classic complete simulation preorder and equivalence, respectively.

On the one hand, for these (pre)congruence we have presented results that show the existence of finite ground-complete equational axiomatizations. In order to obtain a better understanding of the equational theory of weak complete simulation semantics, we have presented both conditional and unconditional versions of such axiomatizations.

On the other hand, we have proved that the (in)equational theory of the weak complete simulation precongruence over BCCSP does not have a finite (in)equational basis. This result is true in the presence of at least one observable action. Moreover, we have shown that the collection of (in)equations in at most one variable that hold in weak complete simulation semantics over BCCSP does not have an (in)equational basis of bounded depth, and therefore not a finite one.

It would be interesting to obtain infinite, but finitely described, complete axiomatizations of weak complete simulation semantics. This is a topic that we leave for future research.

While in [2] we showed that for both weak simulation and weak ready simulation semantics the finite axiomatizability of the semantics was crucially dependent on whether the set of observable actions was finite or infinite, for the weak complete simulation semantics this is not the case. As reflected in Table 1, for any non-empty set of actions, it is impossible to find a finite complete axiomatization.

We find it pleasing that all the known results on the existence of finite bases for the weak, complete and plain simulation semantics (see [10]) in the 'concrete' case, that is without silent moves, are 'lifted' to the weak version of the simulation semantics we have presented. However, it is natural to wonder whether our results for the weak semantics can be obtained in a uniform fashion from those for the concrete ones by applying some 'meta-theorems' linking the equational theories of concrete and weak semantics over some process algebra. Examples of such result are offered in [4,9]. The paper [4] presents a general technique for obtaining new results pertaining to the non-finite axiomatizability of behavioural (pre)congruences over process algebras from known ones. The technique is based on establishing translations between languages that preserve sound (in)equations and (in)equational proofs over the source language, and reflect families of (in)equations responsible for the non-finite axiomatizability of the target language. In [2], we used the reduction method from [4] to lift known axiomatizability results for simulation semantics to the weak setting. So far, however, we have been unable to apply the reduction technique to obtain axiomatizability results for weak semantics that, unlike weak simulation semantics, do not satisfy the equation τe. The development of general links between axiomatizations for weak and concrete semantics, along the lines of those presented in [9], is a very interesting line for future research.

We plan to investigate next the weak versions of process semantics in van Glabbeek's spectrum that are based on notions of decorated traces. We have already started working on this topic and we plan to report on our results in a forthcoming article.

Following the lead of [1,6,23], it would also be interesting to study rule formats for operational semantics that provide congruence formats for the semantics considered in this paper, and to give procedures for generating ground-complete axiomatizations for them for process languages in the given formats.

Acknowledgements. We thank Wan Fokkink for his insightful comments on a previous version of this paper and for bringing [9] to our attention.

References

1. Aceto, L., Bloom, B., Vaandrager, F.W.: Turning SOS rules into equations. Information and Computation 111(1), 1–52 (1994)
2. Aceto, L., de Frutos Escrig, D., Gregorio-Rodríguez, C., Ingólfsdóttir, A.: Axiomatizing Weak Ready Simulation Semantics over BCCSP. In: Cerone, A., Pihlajasaari, P. (eds.) ICTAC 2011. LNCS, vol. 6916, pp. 7–24. Springer, Heidelberg (2011)
3. Aceto, L., Fokkink, W., Ingólfsdóttir, A., Luttik, B.: Finite Equational Bases in Process Algebra: Results and Open Questions. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F., de Vrijer, R. (eds.) Processes, Terms and Cycles: Steps on the Road to Infinity. LNCS, vol. 3838, pp. 338–367. Springer, Heidelberg (2005)
4. Aceto, L., Fokkink, W., Ingólfsdóttir, A., Mousavi, M.: Lifting non-finite axiomatizability results to extensions of process algebras. Acta Informatica 47(3), 147–177 (2010)

5. Baeten, J., Basten, T., Reniers, M.: Process Algebra: Equational Theories of Communicating Processes. Cambridge Tracts in Theoretical Computer Science, vol. 50. Cambridge University Press (2009)
6. Baeten, J.C.M., de Vink, E.P.: Axiomatizing GSOS with termination. Journal of Logic and Algebraic Programming 60-61, 323–351 (2004)
7. Bergstra, J.A., Klop, J.W.: Process algebra for synchronous communication. Information and Control 60(1-3), 109–137 (1984)
8. Bloom, B., Istrail, S., Meyer, A.R.: Bisimulation can't be traced. Journal of the ACM 42(1), 232–268 (1995)
9. Chen, T., Fokkink, W., van Glabbeek, R.: On the axiomatizability of impossible futures (unpublished manuscript, 2011)
10. Chen, T., Fokkink, W., Luttik, B., Nain, S.: On finite alphabets and infinite bases. Information and Computation 206(5), 492–519 (2008)
11. de Frutos-Escrig, D., Gregorio-Rodríguez, C., Palomino, M.: On the unification of process semantics: Equational semantics. Electronic Notes in Theoretical Computer Science 249, 243–267 (2009)
12. de Frutos Escrig, D., Gregorio-Rodríguez, C.: Universal coinductive characterizations of process semantics. In: 5th IFIP International Conference on Theoretical Computer Science. IFIP, vol. 273, pp. 397–412. Springer, Heidelberg (2008)
13. van Glabbeek, R.: The linear time – branching time spectrum I; the semantics of concrete, sequential processes. In: Handbook of Process Algebra, ch. 1, pp. 3–99. Elsevier (2001)
14. Hennessy, M., Milner, R.: Algebraic laws for nondeterminism and concurrency. Journal of the ACM 32, 137–161 (1985)
15. Hoare, C.: Communicating Sequential Processes. Prentice Hall (1985)
16. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. Information and Computation 94(1), 1–28 (1991)
17. Milner, R.: An algebraic definition of simulation between programs. In: Proceedings 2nd Joint Conference on Artificial Intelligence, pp. 481–489. BCS (1971)
18. Milner, R.: Communication and Concurrency. Prentice Hall (1989)
19. Milner, R.: A complete axiomatisation for observational congruence of finite-state behaviors. Information and Computation 81(2), 227–247 (1989)
20. Moller, F.: Axioms for Concurrency. PhD thesis, Report CST-59-89, Department of Computer Science, University of Edinburgh (1989)
21. Park, D.M.: Concurrency and Automata on Infinite Sequences. In: Deussen, P. (ed.) GI-TCS 1981. LNCS, vol. 104, pp. 167–183. Springer, Heidelberg (1981)
22. Plotkin, G.D.: A structural approach to operational semantics. Journal of Logic and Algebraic Programming 60-61, 17–139 (2004)
23. Ulidowski, I.: Axiomatisations of Weak Equivalences for De Simone Languages. In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, pp. 219–233. Springer, Heidelberg (1995)

Complexity Insights of the Minimum Duplication Problem

Guillaume Blin[1], Paola Bonizzoni[2], Riccardo Dondi[3], Romeo Rizzi[4], and Florian Sikora[1,5]

[1] Université Paris-Est, LIGM - UMR CNRS 8049, France
{gblin,sikora}@univ-mlv.fr
[2] DISCo, Universitá degli Studi di Milano-Bicocca, - Milano, Italy
bonizzoni@disco.unimib.it
[3] DSLCSC, Universitá degli Studi di Bergamo, - Bergamo, Italy
riccardo.dondi@unibg.it
[4] DIMI - Università di Udine - Udine, Italy
Romeo.Rizzi@dimi.uniud.it
[5] Lehrstuhl für Bioinformatik, Friedrich-Schiller-Universität Jena, Germany

Abstract. The Minimum Duplication problem is a well-known problem in phylogenetics and comparative genomics. Given a set of gene trees, the Minimum Duplication problem asks for a species tree that induces the minimum number of gene duplications in the input gene trees. More recently, a variant of the Minimum Duplication problem, called Minimum Duplication Bipartite, has been introduced in [14], where the goal is to find all *pre-duplications*, that is duplications that precede, in the evolution, the first speciation with respect to a species tree. In this paper, we investigate the complexity of both Minimum Duplication and Minimum Duplication Bipartite problems. First of all, we prove that the Minimum Duplication problem is APX-hard, even when the input consists of five uniquely leaf-labelled gene trees (progressing on the complexity of the problem). Then, we show that the Minimum Duplication Bipartite problem can be solved efficiently by a randomized algorithm when the input gene trees have bounded depth.

1 Introduction

The evolutionary history of the genomes of eukaryotes is the result of a series of evolutionary events, called *speciations*, that produce new species starting from a common ancestor. This evolutionary history has been deeply studied in Computational Biology, and is usually represented using a special type of phylogenetic tree called *species tree* [9]. A *species tree* is a rooted binary tree whose leaves are uniquely labelled by a set Λ representing the extant species, where the common ancestor of the contemporary species is associated with the root of the tree. The internal nodes represent hypothetical ancestral species (and the associated speciations). Speciations are not the only events that influence the evolution. Indeed, there are other events, such as gene duplication, gene loss and lateral gene

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 153–164, 2012.

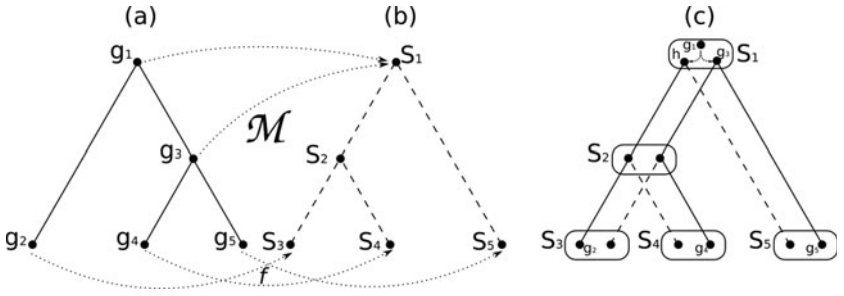


Fig. 1. (a) a gene tree T. (b) a species tree S where $\mathcal{M}$ is the `lca` mapping from T to S; each gene in $\{g_2, g_4, g_5\}$ is mapped by function f in the species that gene belongs to. (c) a reconciled tree for T and S based on the a priori duplication of gene g_1 into genes h and g_3.

transfer that, although not leading to new species, are fundamental in evolution. In this paper, we focus on gene duplications which are known to be essential for the evolution of many eukaryotes groups, such as vertebrates, insects and plants [8]. Gene duplication can be described as the genomic event that causes a gene inside a genome to be copied, resulting in two copies of the same gene that can evolve independently. Genes of extant species are called *homologous* if they evolved from a common ancestor, through speciations and duplications events [10]. Evolution of homologous genes, with regards to the extant species, is usually represented using another special type of phylogenetic tree called *gene tree*. A *gene tree* is a rooted binary tree whose leaves are (not necessarily uniquely) labelled using elements of the set Λ. Indeed, despite the fact that, biologically speaking, leaves in the gene tree represent genes, for ease, the gene tree is labelled according to the species from which the corresponding gene was sampled. Therefore, leaves similarly labelled represent duplicated genes that evolved independently and appear in a common extant species. As in the species tree, the root and the internal nodes respectively represent the common ancestor and ancestral genes explaining their evolution.

With regards to the set of labels Λ, gene and species trees are said to be *comparable*. Nevertheless, due to complex evolutionary processes such as gene duplication and loss, comparable gene and species trees very often present incompatibilities. A challenging problem is then to reconcile the gene and species trees with hypothetical gene duplications. For example, in Fig. 1, given two comparable gene and species trees inducing incompatibilities, one can infer a reconciled tree based on the *a priori* duplication of gene g_1 into genes h and g_3 (h is a hypothetical ancestor of genes g_2, g_4), which afterwards both speciate according to the topology of the species tree. Based on the principle of parsimony, one is interested in finding the minimum number of gene duplications that can explain all the incompatibilities. This last can be inferred by the so-called *lowest common ancestor mapping* (`lca` mapping), denoted by $\mathcal{M}$ and defined as

follows. $\mathcal{M}$ maps any gene of the gene tree to the latest species from which the gene could be sampled. In other words, $\mathcal{M}$ maps each ancestral gene g of the gene tree to the most recent common ancestor of the extant species from which all the descendant of g were sampled.

For example, in Fig. 1, according to $\mathcal{M}$, g_3 is mapped to S_1 since S_1 is the most recent common ancestor of S_4 and S_5 from which were sampled (represented as a function f) respectively the descendant g_4 and g_5 of g_3. Observe that, considering $\mathcal{M}$, any leaf of the gene tree is mapped to the unique leaf of S similarly labelled (according to Λ). Given $\mathcal{M}$, a gene in the gene tree is a gene duplication if it has a descendant with the same $\mathcal{M}$ mapping. Then, the reconciliation cost is defined as the number of gene duplications in the gene tree induced by the species tree. Computation of this distance has been widely investigated in the context of the Minimum Duplication problem [15,13,11,4], where given a set of gene trees, the objective is to compute a species tree that induces a minimum number of gene duplications.

The Minimum Duplication problem is known to be NP-hard [13]. More recently, the Minimum Duplication problem has been related to the Minimum Triplets Consistency [4]. The complexity of Minimum Triplets Consistency has been deeply studied, and the problem is known to be W[2]-hard [5] and inapproximable within factor $O(\log n)$ [5]. These results coupled with the reduction provided in [4] implies that the Minimum Duplication is NP-hard, W[2]-hard (despite of [15]) and inapproximable within factor $O(\log n)$ even in the specific case of a forest composed of an unbounded number of uniquely leaf-labelled gene trees with three leaves [4]. Therefore, different heuristics and Integer Linear Programs have been developed [2,3,7,6]. Recently, the Minimum Duplication Bipartite problem has been introduced to tackle the Minimum Duplication problem [14]. The Minimum Duplication Bipartite problem corresponds to finding all *pre-duplications*; that is duplications that precede, in the evolution, the first speciation with respect to a species tree. Roughly, this means that only the first level of the species tree is of importance. Indeed, one is interested in knowing if a given species belongs to the subtree of S rooted at the left child of the root or at the right one. Therefore, one can view the species tree as a bipartition (Λ_1, Λ_2) of the set of species Λ. Solving the Minimum Duplication Bipartite problem recursively produces a natural greedy heuristic for the Minimum Duplication problem. The Minimum Duplication Bipartite problem was shown to be 2-approximable [14], but its complexity remains open.

In this contribution, we provide results relying both on the Minimum Duplication problem and the Minimum Duplication Bipartite problem. First of all, we prove that the Minimum Duplication problem is APX-hard, even when the input consists of five uniquely leaf-labeled gene trees (that is for a bounded number of gene trees). Then, we show that the Minimum Duplication Bipartite problem can be solved efficiently by a randomized algorithm when the input gene trees have bounded depth. greedy heuristic for the Minimum Duplication problem. Due to space consideration, we do not provide full details and proofs which are deferred to the full version of the paper.

2 On a Tight Inapproximability

We present a reduction from Minimum Vertex Cover on cubic graphs (MVCC) to the specific case of the Minimum Duplication problem – denoted Min-5-Dup – where given a set of five uniquely leaf labelled gene trees $\mathcal{F} = \{T_1, T_2, T_3, T_4, T_5\}$, the objective is to compute a species tree S that induces a minimum number of gene duplications (afterwards denoted as $d(\mathcal{F}, S)$). Let $G = (V_G, E_G)$ be a cubic graph (*i.e.* every vertex has degree three), MVCC problem asks for a subset $V'_G \subseteq V_G$, such that for each edge $(v_i, v_j) \in E_G$, at least one of $\{v_i, v_j\}$ belongs to V'_G. In a first step, starting from any cubic graph $G = (V_G, E_G)$, we will construct an associated input $\mathcal{F} = \{T_1, \ldots, T_5\}$ of Min-5-Dup. Then, we will demonstrate that any species tree S such that $d(\mathcal{F}, S) < q = 6|E_G| + 3|V_G| + 1$ must be *canonical* (defined afterwards). Finally, we will prove that our construction is indeed an L-reduction.

In order to define formally the gene trees, let us first define the central notion of *comb graph*. We will consider a specific subclass of comb graphs corresponding to a binary tree where all the internal nodes lie on a single simple (*i.e.* with no repeated vertices) path referred as the *spine*. For ease, we will nevertheless use the term comb graph in the following to denote those last. Given a sequence $L = \langle l_1, \ldots, l_k \rangle$ of k labels, let $C(L)$ denote the comb graph whose leaves are labelled according to a postorder traversal using L (*i.e.* $l_x \in L$ is the label of the unique leaf of depth x). For example, in Fig. 1, the gene tree (a) corresponds to the comb graph $C(\langle g_2, g_4, g_5 \rangle)$.

Let us now define two operations on trees. Let $T_1 \triangle T_2$ be a tree obtained from two trees T_1 and T_2, by connecting the roots of T_1 and T_2 to a new vertex v which becomes the root of $T_1 \triangle T_2$. Inserting T_2 in the edge e of T_1 will denote the operation that leads to a tree obtained from T_1 and T_2 by replacing the edge $e = (v, v')$ in T_1 by two edges (v, w) and (w, v') and connecting the root of T_2 to the new vertex w.

We are now ready to define the gene trees $T_1, \ldots, T_5$. Roughly, we will associate to each vertex $v \in V_G$, a specific tree T_v and to each edge $e \in E_G$, two trees T^1_e, T^2_e. These trees will be then combined to build the gene trees $T_1, \ldots, T_5$. For ease, let us consider the following order of edges of E_G, $\langle e_1, e_2, \ldots e_{|E_G|} \rangle$ s.t. $\forall e_x = (v_i, v_j)$, $e_y = (v_h, v_k)$ with $x < y$, $i < j$ and $h < k$, either $(i < h)$ or $(i = h$ and $j < k)$. According to this order, we define the following three sequences of labels: $M_1 = \langle m^1_1, m^1_2, \ldots m^1_{|E_G|} \rangle$, $M_2 = \langle m^2_1, m^2_2, \ldots m^2_{|E_G|} \rangle$ and $L = \langle l^1_1, l^2_1, l^1_2, l^2_2, \ldots l^1_{|E_G|}, l^2_{|E_G|} \rangle$. Roughly, any edge e_x is represented by the four labels $\{m^1_x, m^2_x, l^1_x, l^2_x\}$. First of all, for any edge $e_x \in E_G$, let us build the two trees $T^1_{e_x} = C(\langle l^1_x, m^2_x, l^2_x \rangle)$ and $T^2_{e_x} = C(\langle l^2_x, m^2_x, l^1_x \rangle)$. Moreover, for any $v \in V_G$ s.t. v is incident to the edges e_x, e_y and e_z, we build a tree $T_v = \big(C(\langle m^1_x, m^1_y, m^1_z \rangle) \triangle C(\langle l^1_x, l^1_y, l^1_z \rangle)\big) \triangle C(\langle m^2_x, m^2_y, m^2_z \rangle)$ (see Fig. 2).

We will now build the gene trees T_1 to T_5 by starting from a comb graph where subtrees representing vertices and edges will be inserted in. Let T_5 be obtained from $C(\langle f_{|V_G|+1}, f^1_{|V_G|}, \ldots, f^q_{|V_G|}, f_{|V_G|}, \ldots f^1_1, \ldots, f^q_1, f_1 \rangle)$, by inserting in the edge connecting f_1 and its parent the subtree $C(M_1) \triangle C(M_2)$. Regarding

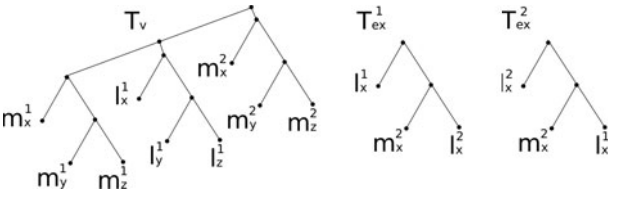


Fig. 2. The trees T_v, $T^1_{e_x}$ and $T^2_{e_x}$ for $v \in V_G$ incident to the edges e_x, e_y and e_z

the construction of T_1 to T_4, let us assume that we are also provided a 4-coloring $\lambda : V_G \mapsto \{1, 2, 3, 4\}$ of G (for example, by applying the polynomial-time greedy Welsh-Powell algorithm [16]). Let any T_i, $1 \leq i \leq 4$, be first define as a the following comb graph: $T_i = C(\langle f_1, f_2, \ldots, f_{|V_G|+1}, f^1_{|V_G|}, \cdots f^q_{|V_G|}, f^1_{|V_G|-1} \ldots, f^q_1 \rangle)$. We then insert, for each $v_i \in V_G$, the tree T_{v_i} in the edge connecting the parents of f_i and f_{i+1} in the gene tree T_x where $x = \lambda(v_i)$ (see Fig. 3). Moreover, for each $e_x = (v_i, v_j) \in E_G$ (ordered from e_1 to $e_{|E_G|}$), the tree $T^1_{e_x}$ is inserted in the edge connecting the parent of f_i and its other child in the gene tree T_x where $x = min\{1, 2, 3, 4\} \setminus \{\lambda(v_i), \lambda(v_j)\}$ (*i.e.* the gene tree having the smallest index and not containing neither T_{v_i}, nor T_{v_j}). Finally, for each $e_x = (v_i, v_j) \in E_G$, the tree $T^2_{e_x}$ is inserted in the edge connecting the parent of f_i and its other child in the gene tree T_x where $x = max\{1, 2, 3, 4\} \setminus \{\lambda(v_i), \lambda(v_j)\}$ (*i.e.* the gene tree having the biggest index and not containing neither T_{v_i}, nor T_{v_j}). A sketch of this construction is given in Fig. 3.

Due to space constraints, we only provide here a sketch of our proof (full details available in appendix). First of all, we can prove that, by construction, all the gene trees are indeed uniquely leaf-labelled. Then, we can prove that only *canonical* solutions are of interest. Roughly, a canonical solution (*i.e.* a species tree S) is a copy of T_5 where extra leaves of $L = \{l^1_x, l^2_x : \forall e_x \in E_G\}$ are each inserted either in $C(M_1)$ or $C(M_2)$. The insertion of l^1_x, l^2_x in $C(M_1)$ or $C(M_2)$ depends on the fact that the edge $e_x = (v_i, v_i)$ is covered by v_i or v_j.

We can, moreover, prove that, in a canonical solution, (1) each vertex on the path from the root of T_j, with $1 \leq j \leq 4$, to the parent of $f_{|V_G|+1}$ (excluding this last) induces a duplication (that is $5|V_G| + 2|E_G|$ in total), (2) each edge $e_x = (v_i, v_j) \in E_G$ induces a duplication in the root of one of $\{T^1_{e_x}, T^2_{e_x}\}$ and a duplication in the root of either T_{v_i} or T_{v_j}. One can then easily see that the minimum number of duplications is then related to the minimum cover size. Hence the following lemma holds.

Lemma 1. *Let $G = (V_G, E_G)$ be an instance of* MVCC *and let $\mathcal{F} = \{T_1, ..., T_5\}$ be the corresponding instance of* Min-5-Dup*. Then, starting from a cover V'_G of G, we can compute in polynomial time a solution S of* Min-5-Dup *for $\mathcal{F}$ s.t. $d(\mathcal{F}, S) \leq 5|V_G| + 3|E_G| + |V'_G|$; starting from a solution S of* Min-5-Dup *for $\mathcal{F}$ s.t. $d(\mathcal{F}, S) \leq 5|V_G| + 3|E_G| + p$, we can compute in polynomial time a cover of G of size at most p.*

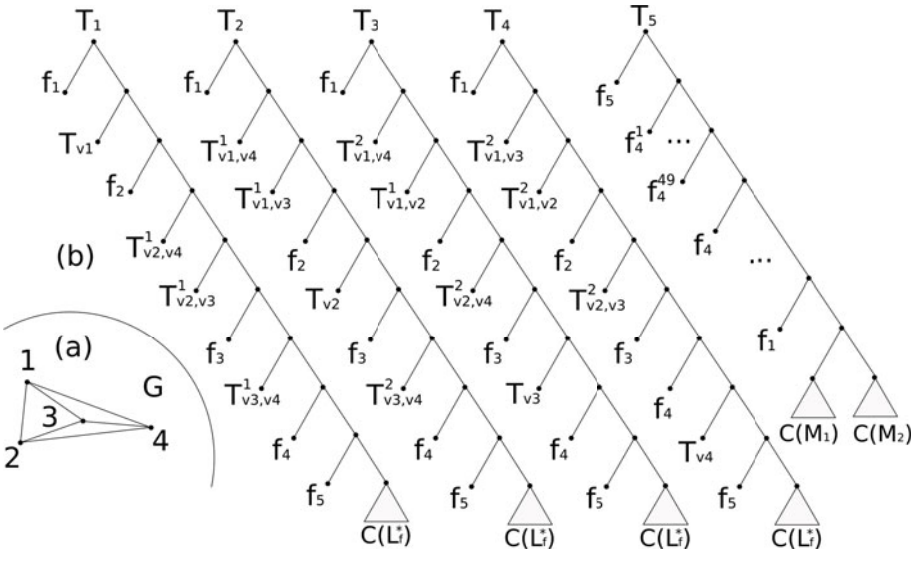


Fig. 3. Gene trees T_1 to T_5 obtained from the cubic graph G where $L_f^* = \langle f^1_{|V_G|}, \ldots, f^q_{|V_G|}, \ldots, f^1_1, \ldots, f^q_1 \rangle$ and $\forall 1 \leq i \leq 4, \lambda(v_i) = i$

Lemma 1 concludes the reduction. Since MVCC is APX-hard [1], provided our L-reduction, we can conclude that Min-5-Dup is also APX-hard.

Theorem 1. *The* Minimum Duplication *problem is* APX*-hard, even when the input consists of five uniquely leaf-labelled gene trees*

3 A Randomized Approach

In this section, we investigate the complexity of the Minimum Duplication Bipartite problem and show that it can be solved efficiently by a randomized algorithm when the input gene trees have bounded depth. A randomized algorithm can be seen simply as an algorithm that is allowed to do some random decisions as it processes the input. Whereas defining a randomized algorithm is quite easy, the performance analysis of this last is more complicated. Indeed, first, one has to compute the probability of success of the randomized algorithm (*i.e.* probability to end up with an optimal solution). Then, one can amplify the probability of success simply by repeatedly running the algorithm, with independent random choices, and taking the best solution found. If one, moreover, prove that the overall running time required to get a high probability of success is polynomial in the size of the input, then it implies that the problem is randomized polynomial (in RP-class). For further details on randomized algorithms, the reader should consider the book of Kleinberg and Tardos [12].

In order to prove that the Minimum Duplication Bipartite problem is randomized polynomial, we first provide a randomized algorithm for a variant of the Minimum Cut problem, called Minimum Cut in Colored Graph. Then, we will prove that the Minimum Duplication Bipartite problem can be translated into a Minimum Cut in Colored Hypergraph problem that can be solved efficiently applying our randomized algorithm on hypergraphs with bounded hyperedges degree. It is of importance to note that, as far as we know, this is the first attempt of solving by randomization the minimum cut in colored hypergraph. Providing a randomized algorithm for general hypergaphs with unbounded hyperedges degree is still open.

Let us first introduce the Minimum Cut in Colored Graph problem: Given a set of colors $\mathcal{C}$ and an undirected colored graph $G = (V, E)$ where any edge is colored with a color from $\mathcal{C}$, find a minimal colored cut of G – that is a partition of V into two non-empty sets A and B such that the number of colors used by the edges having one end in A and the other in B is minimized.

For ease, let $\texttt{col} : E \mapsto \mathcal{C}$ be a function returning the color of a given edge and $\texttt{mul}(c) = |\{e : e \in E \text{ and } \texttt{col}(e) = c\}|$ be a function returning the multiplicity of a given color. Moreover, for sake, given a graph $G = (V, E)$, let $\texttt{col}(G) = \bigcup_{e \in E} \texttt{col}(e)$ denote the set of colors used in G. Let us now describe an algorithm inspired by the folklore Contraction Algorithm [12] used for solving the classical Minimum Cut problem (*i.e.* minimizing the number of edges having one end in A and the other in B) on uncolored graph by randomized algorithm.

As in [12], our Colored Contraction Algorithm uses a connected multigraph $G = (V, E)$ – that is an undirected graph that is allowed to have more then one edge between the same pair of vertices – which is moreover colored. The algorithm starts by choosing, uniformly at random, a color $c \in \texttt{col}(G)$ and contracting any edge $e \in E$ such that $\texttt{col}(e) = c$ (and thus all such edges). Contracting an edge $(u, v) \in E$ will produce a new graph $G' = (V', E')$ in which u and v are identified as a single new vertex w whereas all other vertices are keeping their original identity (*i.e.* $V' = \{V \cup \{w\}\} \setminus \{u, v\}$). In G', $E' = \{E \cup \{(w, v'') : v' \in \{u, v\}, (v', v'') \in E\}\} \setminus \{(v', v'') : v' \in \{u, v\}, v'' \in V\}$. Roughly, E' is a copy of E where any edge (u, v) has been removed whereas any other edge has been preserved, but if one of its ends was equal to u or v, then this end is updated to be equal to the new node w. Note that the contraction operation may end up in a multigraph even when starting from a classical graph G. In this process, contracting all the edges that have the selected color c roughly corresponds to a sequence of $\texttt{mul}(c)$ contractions, each reducing the number of vertices by one. Colored Contraction Algorithm then continues recursively on G', by choosing, uniformly at random, a color $c \in \texttt{col}(G')$ and contracting any edge $e \in E$ such that $\texttt{col}(e) = c$. As these recursive calls proceed, the vertices of V' should be viewed as *supervertices*: each supervertex w corresponds to the subset $\mathcal{S}(w) \subseteq V$ that has been "swallowed up" in the contractions that produced w. The algorithm ends when it reaches a graph G' with only two super-vertices v_A and v_B. We output $(A = \mathcal{S}(v_A), B = \mathcal{S}(v_B))$ as the colored-cut found by the algorithm.

Let us now analyze the performance of the Colored Contraction Algorithm – which cannot be derived directly from the one of the original Contraction Algorithm. Since the algorithm is making random choices, there is some probability that it will succeed in finding a minimum colored-cut (and some probability that it would not). In order to prove that this algorithm is worthwhile, we will prove that the probability of success is only polynomially small; inducing that, by running the algorithm a polynomial number of times and returning the best colored-cut found in any run, one would be able to produce an optimal colored-cut with high probability.

Theorem 2. *The* Colored Contraction Algorithm *returns an optimal colored-cut G with probability at least* $(|V|^{2k})^{-1}$ *where* $k = \text{MAX}_{c \in \mathcal{C}} \texttt{mul}(c)$

Proof. Let us assume that the optimal minimum colored-cut (A, B) of G is of size OPT; that is the set of edges having one end in A and the other end in B (referred afterwards as the *cut-set*) is colored using OPT colors of $\mathcal{C}$. Note that unlike the classical MINIMUM CUT problem, the goal here is to minimize the number of colors in the cut-set itself. Moreover, let $G_{\text{OPT}} = G[A \cup B, \{(u, v) : (u, v) \in E \text{ and } u \in A, v \in B\}]$ corresponds to the bipartite graph representing the cut-set of (A, B). In order to compute a lower bound on the probability that the Colored Contraction Algorithm returns the minimum colored-cut (A, B), we first notice some important properties.

First, remark that any vertex $v \in V$ cannot have a degree less than OPT. Indeed, otherwise, $(\{v\}, V \setminus \{v\})$ would correspond to a colored-cut inducing at most $\text{OPT} - 1$ colors, contradicting our hypothesis that (A, B) is an optimal minimum colored-cut of G. Therefore, any vertex of G is of degree at least OPT; inducing the following lower bound on E: $|E| \geq \frac{\text{OPT}|V|}{2}$. We know moreover that, since each color of $\mathcal{C}$ can be used at most $k = \text{MAX}_{c \in \mathcal{C}} \texttt{mul}(c)$ times in E, we have that $|E| \leq k \cdot |\mathcal{C}|$. This leads to the following inequalities.

$$|V| \cdot \text{OPT} \leq 2 \cdot |E| \leq 2k \cdot |\mathcal{C}| \tag{1}$$

Let us now evaluate the probability $Pr[F_j]$ that the Colored Contraction Algorithm fails at the j^{th} step of the recursion (that is when already $j - 1$ contractions have been done). Considering what could go wrong in the j^{th} step of the Colored Contraction Algorithm, one can check that the unique issue would be that the uniformly at random choice of a color c unfortunately select one color of the set of OPT colors used by the cut-set – which will be then contracted inducing that the algorithm would not be able to find the optimal colored-cut (A, B) since at least a node of A and a node of B would be both contracted into the same supervertex. Hence the probability that an edge of the current graph G' is both in the optima cut-set and contracted is at most $\frac{\text{OPT}}{|\mathcal{C}'|}$, since there are at most OPT edges to be chosen among $|\mathcal{C}'|$ edges, where $\mathcal{C}' = \texttt{col}(G')$. According to Inequality 1, considering that the graph at j^{th} step is G' and $\mathcal{C}' = \texttt{col}(G')$

$$Pr[F_j] \leq \frac{\text{OPT}}{|\mathcal{C}'|} \leq \frac{2k \cdot |\mathcal{C}'|}{|V'| \cdot |\mathcal{C}'|} = \frac{2k}{|V'|} \tag{2}$$

The colored-cut (A, B) will actually be returned by the algorithm if no edge of the cut-set is contracted in any of the at most $|V| - 2$ iterations. If we write S_j for the event that an edge of the cut-set has not been contracted until the j^{th} step, then, according to Inequality 2, $Pr[S_j] \geq 1 - Pr[F_j] = 1 - \frac{2k}{|V'|}$ where the graph at j^{th} step is $G' = (V', E')$. For ease, let us consider the sequence of color choices as being $\mathcal{S}_c = (c_1, c_2 \ldots)$ and $\lambda_j = \sum_{i<j \text{ and } c_i \in \mathcal{S}_c} \texttt{mul}(c_i)$. On the whole the probability that the Colored Contraction Algorithm returns the optimal colored-cut (A, B) is thus at least

$$Pr[Success] \geq \prod_{i=0}^{\lambda_1 - 1} (1 - \frac{2k}{|V| - i}) \cdot \prod_{i=\lambda_2}^{\lambda_3 - 1} (1 - \frac{2k}{|V| - i}) \ldots \prod_{i=\lambda_{|\mathcal{S}_c|-1}}^{\lambda_{|\mathcal{S}_c|} - 1} (1 - \frac{2k}{|V| - i}) \quad (3)$$

$$\geq \prod_{i=0}^{\lambda_1 - 1} (\frac{|V| - i - 2k}{|V| - i}) \cdot \prod_{i=\lambda_2}^{\lambda_3 - 1} (\frac{|V| - i - 2k}{|V| - i}) \ldots \prod_{i=\lambda_{|\mathcal{S}_c|-1}}^{\lambda_{|\mathcal{S}_c|} - 1} (\frac{|V| - i - 2k}{|V| - i}) \quad (4)$$

$$\geq \prod_{i=0}^{\lambda_{|\mathcal{S}_c|} - 1} (\frac{|V| - i - 2k}{|V| - i}) = \frac{\cancel{|V| - 2k}}{|V|} \ldots \frac{|V| - 2k - 2k}{\cancel{|V| - 2k}} \ldots \frac{|V| - (\lambda_{|\mathcal{S}_c|} - 1) - 2k}{\cancel{|V| - (\lambda_{|\mathcal{S}_c|} - 1)}} \quad (5)$$

$$\geq \frac{\prod_{i=2k}^{\lambda_{|\mathcal{S}_c|} - 1} |V| - i - 2k}{\prod_{i=0}^{2k-1} |V| - i} \geq \frac{1}{|V|^{2k}} = (|V|^{2k})^{-1} \quad (6)$$

□

Then according to Theorem 2, we know that a single run of the Colored Contraction Algorithm fails to find an optimal colored-cut with probability at most $(1 - (|V|^{2k})^{-1})$. One can then amplify the probability of success simply by repeatedly running the algorithm, with independent random choices, and taking the best colored-cut found. It is known that the function $(1 - n^{-1})^n$ converges monotonically from $\frac{1}{4}$ up to $\frac{1}{e}$ as n increases from 2 [12]. Thus, if we run the algorithm $|V|^{2k}$ times, then the probability that we fail to find an optimal colored-cut in any run is at most $(1 - (|V|^{2k})^{-1})^{|V|^{2k}} \leq \frac{1}{e}$. As usually done, it is easy to even reduce more the failure probability with further repetitions by running the algorithm $|V|^{2k} \ln |V|$ times which induces a probability of failure of at most $e^{-\ln |V|} = \frac{1}{|V|}$. On the overall, the running time required to get a high probability of success is polynomial in $|V|$, since each run of the Colored Contraction Algorithm takes polynomial time, and we run it a polynomial number of times.

Let us now demonstrate how this result can be used in order to solve the Minimum Duplication Bipartite problem.

Theorem 3. *The* Minimum Duplication Bipartite *problem is randomized polynomial time solvable when the gene trees are of bounded depth.*

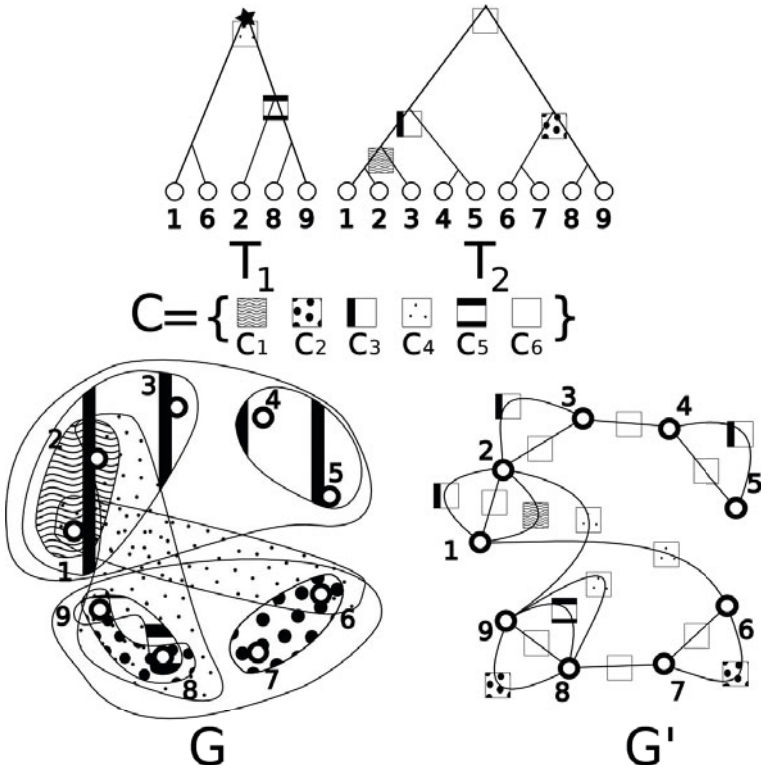


Fig. 4. Illustration of the construction of $G_{\mathcal{F}}$ and G' given $\mathcal{F} = (T_1, T_2)$. Considering the minimum colored-cut $\{1, 2, 3, 4, 5\}, \{6, 7, 8, 9\}$ of size 1, the only induced duplication is represented as a star on T_1.

Proof. In the following, for ease, given a binary tree $T = (V, E)$ and a vertex $v \in V$, let us denote by v^L (resp. v^R) the left (resp. right) child of v and by ζ_v the cluster of v *i.e.* the set of all leaves belonging to the subtree rooted in v. Moreover, for ease, ϑ_T will denote the root of the tree T. Given a gene tree forest $\mathcal{F} = \{T_1 = (V_1, E_1), T_2 = (V_2, E_2), \ldots\}$ built on Λ, considering the definition of the Minimum Duplication Bipartite problem, one wants to define a bipartition (Λ_1, Λ_2) of $\Lambda = \bigcup_{T_i \in \mathcal{F}} V_i$ inducing the minimum number of pre-duplications. In T_i, a node v of V_i is a duplication with respect to (Λ_1, Λ_2), if $\exists v' \in \{v^L, v^R\}$, such that $(\Lambda_1 \bigcap \zeta_{v'} \neq \emptyset) \wedge (\Lambda_2 \bigcap \zeta_{v'} \neq \emptyset)$ is true. In other words, v is a duplication if for one of its children – say $v' - \zeta_{v'}$ contains two leaves not belonging to the same part of the bipartition (Λ_1, Λ_2). Given $\mathcal{F}$ and a set of colors $\mathcal{C}$, we define the following colored hypergraph $G_{\mathcal{F}} = (V, E)$ associated to $\mathcal{F}$. Let $V = \Lambda = \bigcup_{T \in \mathcal{F}} \zeta_{\vartheta_T}$ and there are two hyperedges, for any node v_k of the tree T_i, $\alpha_k^i = \{\zeta_{v_k^L} : |\zeta_{v_k^L}| \geq 2\}$ and $\beta_k^i = \{\zeta_{v_k^R} : |\zeta_{v_k^R}| \geq 2\}$ colored with color $\texttt{col}(\alpha_k^i) = \texttt{col}(\beta_k^i) = c_k^i \in \mathcal{C}$ in E. An illustration of such construction is provided in Fig. 4. Then in $G_{\mathcal{F}}$, a colored-cut of size k' corresponds to a bipartition of the set Λ

inducing k' duplications. Indeed, if the hyperedge α_k^i (resp. β_k^i) belongs to the cut-set, then it induces a duplication for the corresponding vertex v_k in T_i since there exist at least two leaves in $\zeta_{v_k^L}$ (resp. $\zeta_{v_k^R}$) belonging to different parts of the bipartition (Λ_1, Λ_2).

Thus, if one can find a minimum colored-cut in such hypergraphs, then one would be able to solve in polynomial time the Minimum Duplication Bipartite problem. Just consider the Colored Contraction Algorithm presented previously in this section. From any colored hypergraph $G_{\mathcal{F}} = (V, E)$, one may build a colored graph $G' = (V, E')$ where any hyperedge $e = \{v_{i1}, v_{i2} \dots v_{ik}\}$ colored with color $c = \texttt{col}(e)$ has been replaced by a path $v_{i1}, v_{i2} \dots v_{ik}$ colored with c in E' (*i.e.* $E' = \{(v_{ik}, v_{ik+1}) : v_{ik} \in e, e \in E\}$). Notice that an edge $e \in E'$ colored with c is cut if and only if an hyperedge colored c of $G_{\mathcal{F}}$ is cut. Once this colored graph has been obtained, one may apply the Colored Contraction Algorithm which will produce a minimum colored-cut of G' which also induces a minimum colored cut in $G_{\mathcal{F}}$. Since this algorithm has a complexity exponential in the maximum multiplicity of any color of the considered graph, when the size of each hyperedge is bounded, so does the multiplicity of any color since the maximal size of an hyperedge corresponds to the maximal depth of the input gene trees: leading to a randomized polynomial solution for the Minimum Duplication Bipartite problem. □

4 Conclusion

In this paper we have investigated the complexity of two variants of the Minimum Duplication problem. We have proved that the Minimum Duplication problem is APX-hard, even when the input consists of five uniquely leaf-labelled gene trees. Then, we have shown that the Minimum Duplication Bipartite problem can be solved efficiently by a randomized algorithm when the input gene trees have bounded depth.

A natural open problem is the complexity of the Minimum Duplication Bipartite problem when the gene trees have unbounded depth. Furthermore, it would be interesting to deepen the analysis on the complexity of the Minimum Duplication problem, when the input consists of less than five uniquely leaf-labelled gene trees.

References

1. Alimonti, P., Kann, V.: Some APX-completeness results for cubic graphs. Theoretical Comput. Sci. 237(1-2), 123–134 (2000)
2. Bansal, M.S., Burleigh, J.G., Eulenstein, O., Wehe, A.: Heuristics for the Gene-Duplication Problem: A $\Theta(n)$ Speed-Up for the Local Search. In: Speed, T.P., Huang, H. (eds.) RECOMB 2007. LNCS (LNBI), vol. 4453, pp. 238–252. Springer, Heidelberg (2007)
3. Bansal, M.S., Eulenstein, O., Wehe, A.: The Gene-Duplication Problem: Near-Linear Time Algorithms for NNI-Based Local Searches. IEEE/ACM Trans. Comput. Biology Bioinform. 6(2), 221–231 (2009)

4. Bansal, M.S., Shamir, R.: A Note on the Fixed Parameter Tractability of the Gene-Duplication Problem. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB) 8(3), 848–850 (2011)
5. Byrka, J., Guillemot, S., Jansson, J.: New results on optimizing rooted triplets consistency. Discrete Appl. Math. 158(11), 1136–1147 (2010)
6. Chang, W.-C., Burleigh, J.G., Fernández-Baca, D.F., Eulenstein, O.: An ILP solution for the gene duplication problem. BMC Bioinformatics (suppl. 1), S14(12) (2011)
7. Chauve, C., El-Mabrouk, N.: New Perspectives on Gene Family Evolution: Losses in Reconciliation and a Link with Supertrees. In: Batzoglou, S. (ed.) RECOMB 2009. LNCS, vol. 5541, pp. 46–58. Springer, Heidelberg (2009)
8. Eichler, E.F., Sankoff, D.: Structural dynamics of eukaryotic chromosome evolution. Science 301(5634), 521–565 (2003)
9. Felsenstein, J.: Phylogenies from molecular sequences: Inference and reliability. Ann. Review Genet. 22, 521–565 (1988)
10. Fitch, W.M.: Homology a personal view on some of the problems. Trends Genet. 16, 227–231 (2000)
11. Hallett, M.T., Lagergren, J.: New algorithms for the duplication-loss model. In: RECOMB, pp. 138–146 (2000)
12. Kleinberg, J., Tardos, E.: Algorithm Design. Pearson Education (2006)
13. Ma, B., Li, M., Zhang, L.: From Gene Trees to Species Trees. SIAM J. Comput. 30(3), 729–752 (2000)
14. Ouangraoua, A., Swenson, K.M., Chauve, C.: An Approximation Algorithm for Computing a Parsimonious First Speciation in the Gene Duplication Model. In: Tannier, E. (ed.) RECOMB-CG 2010. LNCS, vol. 6398, pp. 290–301. Springer, Heidelberg (2010)
15. Stege, U.: Gene Trees and Species Trees: The Gene-Duplication Problem in Fixed-Parameter Tractable. In: Dehne, F.K.H.A., Gupta, A., Sack, J.-R., Tamassia, R. (eds.) WADS 1999. LNCS, vol. 1663, pp. 288–293. Springer, Heidelberg (1999)
16. Welsh, D.J.A., Powell, M.B.: An upper bound for the chromatic number of a graph and its application to timetabling problems. The Computer Journal 10(1), 85–86 (1967)

A Turing Machine Resisting Isolated Bursts of Faults

Ilir Çapuni and Peter Gács

Boston University, Department of Computer Science,
111 Cummington str, 02215 Boston, USA
{ilir,gacs}@cs.bu.edu

Abstract. We consider computations of a Turing machine under noise that causes consecutive violations of the machine's transition function. Given a constant upper bound β on the size of bursts of faults, we construct a Turing machine $M(\beta)$ subject to faults that can simulate any fault-free machine under the condition that bursts not closer to each other than V for an appropriate $V = O(\beta)$.

1 Introduction

Little is known about the behavior and the power of Turing machines when their operation is subjected to random noise that can change arbitrarily the state and the content of the cell where the head is positioned. The main open question, under every noise model, is whether a machine subject to it can perform arbitrary computations reliably.

Here, we construct a Turing machine that with a constant slowdown can simulate any other Turing machine even if the simulator is subjected to constant size bursts of faults separated by a certain minimum number of steps from each other.

The problem of constructing fault-proof machines from components that can fail was first considered by von Neumann in [11], who addressed the problem in the Boolean circuits model. See new developments based on this work for example in [8,9]. As for uniform models of computation, [10] defines a simple two-dimensional cellular automaton that keeps a bit of information, even though each cell can fail with some small probability. Reliably computing automata of higher dimensions, based on this work, were defined in [6]. Alas, no simple non-ergodic one-dimensional cellular automata appear to exist. The first, complex, such cellular automaton was given in [4] and improved upon in [5]. It supports a hierarchical organization, based on an idea given in [3]. In these constructions, cells are organized in units that perform fault-tolerant simulation of another automaton (of the same kind). The latter simulates even more reliably a third automaton of a similar kind, and so on.

The question of reliable computation with Turing machines (where arbitrarily large bursts may occur with correspondingly small probability) is raised in [5]. As in the case of one-dimensional cellular automata, no simple solution to this

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 165–176, 2012.

problem appears to exist. Our machine is intended as a building block towards the eventual construction of such a hierarchical machine. This follows the pattern of the proof in [4], where each member of the hierarchy of simulations is a similar building block, coping with distant bursts. To the best of our knowledge, this is the first construction of a reliable sequential machine.

In [1], a characterization is given, in terms of the arithmetical hierarchy of Turing machines where data on the storage tapes are exposed to stochastic noise that tends to zero.

Main ideas. We rely on some methods used earlier for cellular automata. *Redundant storage* will counteract losses of *information* (about the simulated machine). *Repeated execution* along with voting over stored candidate results limits the effect of faults during simulation. This extra administration is organized using local addresses and sweep numbers (*structure*). Faults can destroy the structure locally, too. A special local *recovery procedure* is devoted to restoring structure (without attempting to restore "information"). The design and proof of local recovery is the technically most difficult part of the work.

Our contribution uses one of the standard definitions of a Turing machine, with the exception that our machines have no halting state. A Turing machine is a tuple $(\mathcal{Q}, \Sigma, \delta, q_{\text{start}}, F)$, where $\mathcal{Q}$ is a finite set of *states*, Σ is a finite alphabet used in cells of the tape, $\delta : \Sigma \times \mathcal{Q} \to \Sigma \times \mathcal{Q} \times \{-1, 0, +1\}$ is the transition function, q_{start} is called the *starting state*, and F is the set of *final states* (that has the property that whenever M enters a state in F, it can only continue from there to another state in F, without changing the tape). The tape alphabet Σ contains at least the distinguished symbols $\textvisiblespace, 0, 1$ where $\textvisiblespace$ is called the *blank symbol*.

A *configuration* is a tuple (q, h, x), where $q \in \mathcal{Q}$, $h \in \mathbb{Z}$ and $x \in \Sigma^{\mathbb{Z}}$. Here, $x[p]$ is the content of the tape cell at position p. The tape is blank at all but finitely many positions. The work of the machine can be described as a sequence of configurations $C_0, C_1, C_2, \dots$, where C_t is the configuration at time t. Let $M(i, t)$ denote the configuration at time t, when started from the starting state with an input string i written on the tape starting from position 0 where the head is initially. Thus, the symbol at tape position p at time t can be written as $M(i, t).tape[p]$.

The transition function δ tells us how to compute the next configuration from the present one. When the machine is in a state q, at tape position h, and observes tape cell with content a, then denoting $(a', q', j) = \delta(a, q)$, it will change the state to q', change the tape cell content to a' and move to tape position $h+j$, $j \in \{\pm 1, 0\}$. For $q \in F$ we have $a' = a$, $q' \in F$.

We say that a *fault* occurs at time t if the output (a', q', j) of the transition function at this time is replaced with some other value (which is then used to compute the next configuration).

Definition 1 (Trajectory). *Let* Configs_M *denote the set of all possible configurations of a Turing machine* M*. Consider a sequence* $\eta = (\eta(0), \eta(1), \dots)$ *of configurations of* $M = (\mathcal{Q}, \Sigma, \delta)$ *with* $\eta(t) = (q(t), h(t), x(t))$*. This sequence*

will be called a history *of* M *if* $q(0) = q_{\text{start}}$*, further* $x(t+1)[n] = x(t)[n]$ *for all* $n \neq h(t)$*, and* $h(t+1)-h(t) \in \{-1, 0, 1\}$*. A history* η *with* $\eta(t) = (q(t), h(t), x(t))$ *of* M *is called a* trajectory *of* M *if for all* t *we have*

$$(x(t+1)[h(t)], q(t+1), h(t+1) - h(t)) = \delta(x(t), q(t)). \tag{1}$$

If $x \in \Sigma^*$ is a string of tape symbols and $\xi \in \Sigma^{\mathbb{Z}}$ is the tape configuration obtained by surrounding $x = x(0) \cdots x(n)$ by blanks, then η with starting tape configuration ξ is a trajectory of M if $\eta(t) = M(x, t)$ for all t.

We say that a history has a *fault* at time t if (1) is violated at time t. A *burst of faults of length* β is a sequence of at most β consecutive faults.

To define simulation of a noise-free machine M_2 by a noisy machine M_1, we need to specify the correspondence between configurations of these two machines. After a burst, the state of the machine—as well as the content of cells where the head was during the burst, are arbitrary. To proceed with the simulation, the simulating machine must recover any lost information. Redundant storage will help in this. In Sec. 2, we will specify how one step of any machine M_2 is simulated by a bounded number of steps of machine M_1 that will be constructed.

We formalize redundant storage with the help of a code. Let Σ_1, Σ_2 be two finite alphabets. A *block code* is given by a positive integer Q and a pair of functions $\varphi_* : \Sigma_2 \to \Sigma_1^Q$ and $\varphi^* : \Sigma_1^Q \to \Sigma_2$ with the property $\varphi^*(\varphi_*(x)) = x$. A *tape configuration code* is a pair of functions $\varphi_* : \Sigma_2^{\mathbb{Z}} \to \Sigma_1^{\mathbb{Z}}$ and $\varphi^* : \Sigma_1^{\mathbb{Z}} \to \Sigma_2^{\mathbb{Z}}$ that encodes infinite strings of Σ_2 into infinite strings of Σ_1.

In Sec. 3, we will define the relationship that fields of the cell and fields of the head must be in, to help the machine notice that faults have occurred. There we will also present how error-recovery is carried out.

To avoid decoding in the main result, for simplicity we will consider computations whose result is a single symbol in cell 0.

Theorem 1 (Main). *For a given Turing machine* M_2 *and an integer* β *the following items can be constructed:*

1. *Integers* V, Q, C *depending linearly on* β*;*
2. *A block code* (φ_*, φ^*) *of blocksize* Q*;*
3. *A machine* M_1 *whose number of states and alphabet size depend polynomially on those of* M_2 *and on* β*;*

such that the following holds.

Suppose that on input x*, the fault-free machine* M_2 *enters a final state at time* T*. Assume that machine* M_1 *works on input* $\varphi_*(x)$ *in such a way that no bursts of faults occur within* V *steps of each other. Let* t *be any time* $\geq CT$ *such that no fault occurred in the last* Q *steps before and including* t*, then*

$$f(M_1(\varphi_*(x), t).tape[0]) = M_2(x, T).tape[0], \tag{2}$$

for some function f*.*

For a version of the paper containing the full proof of this theorem, see [2].

2 The Structure of the State, Cells, and Simulation

Each state of M_1 will be a tuple, $q = (q_1, q_2, \ldots, q_k)$, where the individual elements of the tuple will be called *fields*, and will have symbolic names.

We will call the direction (-1 for left, 0 for none, and $+1$ for right) of the simulated machine M_2 the *drift*.

The tape of M_1 is split into blocks of Q consecutive cells called *colonies*. One colony of the tape of the simulating machine represents one cell of the simulated machine. The colony that corresponds to the active cell of the simulated machine (that is the cell that the simulated machine is scanning) is called the *base colony*. Once the drift is known the union of the base colony with the neighbor colony in the direction of the drift is called the *extended base colony*.

The head will make some global sweeping movements over the base or extended base colony. We will use the term *sweep direction* of the machine when referring to the direction of the simulating machine in this sweeping movement.

Each cell of the tape of M_1 consists of several fields. Some of these have names identical to fields of the state and will be distinguished by a prefix *c.* (like *c.Info*). The array of values of the same field of the cells will be called a *track*. The basic fields of the state and of cells are listed below. We emphasize that each field of a cell has also a possible value $\emptyset$ corresponding to the case when the state is blank.

1. *Addr* ranges from $-Q$ to $2Q-1$. The values $-Q$ to -1 are taken during a left drift, while the values Q to $2Q-1$ during a right drift.
   Drift stores the direction of the simulated machine M_2. It may have values $\emptyset, -1, 0, 1$. The value $\emptyset$ corresponds to the case when the new *Drift* is still not computed, and will also be the default value (for example in empty cells).
   Field *Sw* numbers the sweeps through the colony. The first right sweep has number 1, and this way each right sweep is odd, each left sweep is even, thus the sweep direction of the head is completely determined by the parity of *Sw* when the head is not on the "turning" points. In the turning points, *Sw* is incremented. Field *c.Sw* holds the number of the most recent sweep. We define $Core = (Addr, Sw, Drift)$ and $c.Core = (c.Addr, c.Sw, c.Drift)$, since this group of fields is crucial in maintaining the simulation structure.
2. The *Info* and *State* tracks represent the tape symbols and the state of the simulated machine M_2.
3. The sweep-through is interrupted by switchbacks called *zigging*. While in the normal mode (see later), the process depends on a fixed parameter $Z = 22\beta$, and is controlled by the fields *ZigDepth* and *ZigDir*. Assume that Q is a multiple of $Z - 4\beta$. Every $Z - 4\beta$ forward steps are accompanied by Z steps backward and forward.
4. The track *c.Rec* will help with restoring structure after a burst. When its subfield *c.Rec.Core* is nonempty, the cell is called *marked*.

One step of machine M_2 is simulated by many steps of M_1 that will consist one unit called *work period*. The *Mode* field of the state takes values in

{Normal, Recovering}. The default is normal mode. On noticing any disorder, the state will switch to recovery mode, with the goal of eventually returning to normal mode.

Let a be the tape configuration of M_2 at time 0. We use a very simple redundant code: the *repetition code*. The tape configuration $a' = \varphi_*(a)$ of M_1 at time 0 is defined such that for each head position h of M_2 and for every $j \in \{0, \ldots, Q-1\}$ $a'[h \cdot Q + j].Info = a[h]$ and $a'[j].State = q_0$. The corresponding block decoding function φ^* is

$$\varphi^*(a[0].f \cdots a[Q-1].f) = \mathrm{maj}(a[0].f, \ldots, a[Q-1].f), \tag{3}$$

where f is the triple of fields (*Info*, *State*, *Drift*) and maj is the majority function defined as follows.

Let $x = (x_1, \ldots, x_n)$ be a sequence of symbols from a finite alphabet $\Sigma = \{a_1, a_2, \ldots, a_m\}$. For each $j = 1, 2, \ldots, m$, let k_j be the number of occurrences of a_j in x, $k_1 + k_2 + \cdots + k_m = n$. Then, $\mathrm{maj}(x_1, x_2, \ldots, x_n) = a_k$, where $k = \arg\max_j k_j$. To compute the majority of a set of fields in an interval of cells—*interval majority*— we use a version of an algorithm from [7] that uses only one pass over the interval, and uses at most 3 extra fields of the state. In all cases of interest for us, this value will exist: if it does not, interval majority is defined arbitrarily. We will also compute the majority of several fields in a single cell. We will call this operation the *field majority*.

The head is initially located at the first cell of the base colony. We assume that this colony and its two neighbor colonies are filled with addresses from $0, \ldots, Q-1$. In cells of the base colony and its left neighbor colony, the *c.Sw* and *c.Drift* are set to Last(+1) and 1 respectively, where Last(δ) denotes the last sweep of the base colony in a working period, and $\delta \in \{\pm 1\}$ is the drift of the majority of cells in the colony . In the right neighbor colony, these values are Last(−1) and −1 respectively. In all other cells, these values are empty. Machine M_1 starts in normal mode, with $Drift = 1$, $Sw = 1$, and $Addr = 0$. All other auxiliary fields have also their initial (or empty) values (see Fig. 1).

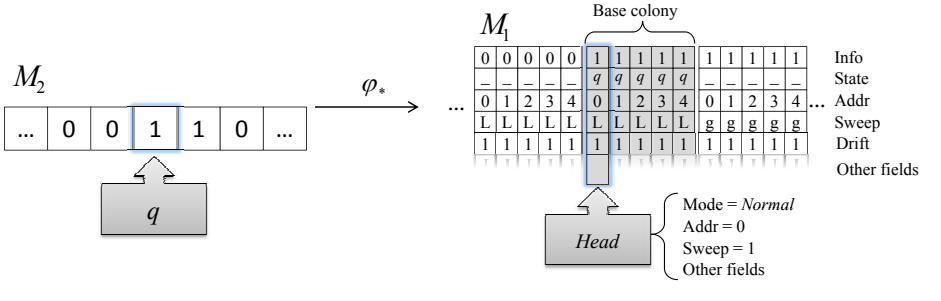


Fig. 1. The initial configuration of machine M_2 is encoded into the initial configuration of M_1, where $L = \mathrm{Last}(1)$ and $g = \mathrm{Last}(-1)$

We introduce $c.Hold[1]$, $c.Hold[2]$, and $c.Hold[3]$ fields of a cell where candidates of the next values of *Info*, *State*, *Drift* will be stored.

2.1 Computation Phase

The aim of this phase is to obtain the new values for the $c.Info$, $c.State$, and $c.Drift$. It consists of the following two parts, performed consecutively:

1. For $i = 1, \ldots, 3$ do the following:
   During the right sweeps, take the colony majority of $c.Info$ and $c.State$ and store these values in a and q respectively. Then compute the triple $\delta_{M_2}(a, q)$, and spread it into the $c.Hold[i]$ track of the base colony on the left sweeps.
2. For each cell of the base colony, compute the field majority of $(c.Info, c.State, c.Drift)$ from $c.Hold[i]$, $i = 1, \ldots, 3$.

2.2 Spreading Phase

The aim of this phase, present only if $Drift \neq 0$, is to spread the new *State* of M_2 into the neighbor colony in the direction of $\delta = Drift$ and to move there.

The $c.Drift$ and $c.Addr$ of the neighbor colony are changed during this sweep.

As in the computing phase, while sweeping the base colony, we compute the majority of $c.State$ and populate the $c.Hold[i]$ tracks of the neighbor colony. After three runs i of this majority computation, in the next two sweeps and back, at every cell of the neighbor colony we set $c.State = \mathrm{maj}(c.Hold[1, \ldots, 3].State)$. Finally, if $Drift = 1$, then move right to cell Q (else stay on the first cell of the base colony).

Now, a new work period begins.

3 Coping with Faults

A burst of faults can change the state to an arbitrary one, and change an interval of cells of size β arbitrarily. We will refer to it by the term "island of bad cells".

Faults cause two kinds of change. One is that they change the information about the represented machine M_2. This problem will be corrected with the help of redundancy (encoding of the information and repetition of the computation). The second kind of change affects the very structure of the simulation. These changes will be detected and corrected locally, by our recovery rules. Certain faults at certain moments of the simulation, by by-passing the coordination check, can leave an island on the tape that could be erased only if the head visits it again.

Let us specify the kind of structural integrity we expect a configuration to have.

Definition 2 (Outer cells). *For $\delta \in \{-1, 1\}$, if a cell is empty or has $0 \leq c.Addr < Q$, $c.Drift = \delta$, $c.Sw = \mathrm{Last}(\delta)$ then it will be called a* right outer cell *if $\delta = -1$. It is a* left outer cell *if $\delta = 1$.*

Healthy Configuration, Workspace. A configuration ξ is *healthy* if the mode is normal, further the following holds. Let d denote the direction of the sweep. We define the position $f = \text{front}(\xi)$, called the *front*, to be the farthest position to which the head has advanced before starting a new backwards zig. Let $\delta = \mathit{Drift}$. We require:

1. The non-blank cells of the tape form a single segment, subdivided into colonies, starting from the *origin* defined by counting back from *Addr*. The leftmost colony and rightmost colony may be only partially filled. The colony starting at the origin is the *base colony*.
   Let us call SpSt the sweep number (an absolute constant) in which the spreading phase starts. The sweep of the first visit to the neighbor colony is called the *spreading sweep*, with sweep number $\text{SpSw}(\delta) = \text{SpSt} + 1 - \max(0, \delta)$. The *extended base colony* is either the base colony or, if $Sw \geq \text{SpSt}$ and $\delta \neq 0$, then the base colony is extended by an adjacent colony on the left or right depending on δ.
2. The front $\text{front}(\xi)$ is in the extended base colony.
3. The drift of nonempty outer cells points towards the base colony. The non-outer cells form a single interval called *workspace*, with the following properties:
   (a) For $Sw < \text{SpSw}(\delta)$, it is equal to the base colony.
   (b) In case of $Sw = \text{SpSw}(\delta)$, the workspace is the smallest interval including the base colony and the cell adjacent to $\text{front}(\xi)$ on the side of the base colony.
   (c) If $\text{SpSw}(\delta) < Sw < \text{Last}(\delta)$, then it is equal to the extended base colony.
   (d) When $Sw = \text{Last}(\delta)$, it is the smallest interval including the future base colony and $\text{front}(\xi)$. The field $c.\mathit{Addr}$ varies continuously over the workspace in all these cases, except possibly $Sw = 1$.
4. For $1 \leq c.Sw \leq \text{Last}(\delta)$, we have $c.Sw(x) = Sw$ in all cells x behind $\text{front}(\xi)$ in the workspace. For $1 < c.Sw$, we have $c.Sw(x) = Sw - 1$ in all cells x ahead of $\text{front}(\xi)$ (inclusive) in the workspace.
5. Consider addresses $c.\mathit{Addr}$ in the workspace. Except for $Sw = 1$, they increase continuously. In the first sweep, the address track $c.\mathit{Addr}$ is either $[-Q, 0)$ or $[Q, 2Q)$, but reduced modulo Q on the segment $[0, \text{front}(\xi))$.
6. If $Sw \geq \text{SpSt}$ or $Sw = 1$ then $c.\mathit{Drift}$ is constant on the workspace.
7. If $Sw = 1$ then $c.\mathit{Info}$ and $c.\mathit{State}$ are constant in each colony.
8. $c.\mathit{Rec}.\mathit{Core} = 0$ throughout, that is all cells are unmarked.

Definition 3 (Coordination). *The state is called* coordinated *with the content of the observed cell if it is possible for them to be in a healthy configuration.*

If at any step of the normal mode, the state fails to be coordinated with the cell it is scanning, a rule `Alarm`, launching recovery, is called.

A *local configuration on* a (finite or infinite) interval I is given by values assigned to the cells of I, along with the following information: whether the head is to the left of, to the right of or inside I, and if it is inside, on which cell, and what is the state. It is easy to see how a local configuration gives rise to a *subconfiguration* on a subinterval I'.

Let ξ be a configuration and $\zeta(I)$ a local configuration that contains the head if and only if $\xi(I)$ contains the head. Then the configuration $\xi|\zeta(I)$ is obtained by replacing ξ with ζ over the interval I, further if ξ contains the head then also replacing $\xi.pos$ with $\zeta.pos$ and $\xi.state$ with $\zeta.state$.

Annotated Configuration. Let $E = 42\beta$. An *annotated configuration* is a quadruple $\mathcal{A} = (\xi, \chi, \mathcal{I}, \mathcal{S}, D)$, with the following meaning. ξ is a configuration, χ is a healthy configuration, $\mathcal{I}$ is a set of intervals of cells called *islands*, further $\mathcal{S} \supset \mathcal{I}$ is a set of intervals of cells called *stains*, and D is an interval containing the head called the *distress area*. The distress area contains any island containing the head. (In islands, the structure may have been damaged, while in stains, only the *Info* and *State* tracks could be. The distress area is where structure is currently being restored.)

Islands and stains are of size $\leq \beta$. The distress area has size $\leq 3E$. We can obtain χ from ξ by changing

1. the $c.Core$ and $c.Rec.Core$ tracks in the islands,
2. the $c.Info$ and $c.State$ track in the stains,
3. the state, the $c.Rec.Core$ track in D, and the head position inside D.

We say that an interval W is the *workspace* of the annotated configuration $\mathcal{A}$ if it is the workspace of χ. The following additional properties are required:

1. There is at most one island intersecting the workspace. There are at most 2 islands in each colony that do not intersect the workspace. If there is more than one, then one is within distance $Z + E$ from the colony boundary towards the base colony.
2. In the base colony, either all stains but one are within a distance E to the left colony boundary (reaching to a distance $\leq E + \beta$), or all but one are within a distance E to the right colony boundary. In all other colonies, all stains but one are within distance E of the boundary towards the base colony.
3. If D is empty then the mode is normal.

We say that a cell is *free* in an annotated configuration when it is not in any island or D. The head is *free* when D is empty.

A *local annotated configuration* on some interval R is obtained from an annotated configuration $\mathcal{A} = (\xi, \chi, \mathcal{I}, \mathcal{S}, D)$ by taking the subconfigurations $\xi(R)$, $\chi(R)$, further the intersections of the islands, stains and distress area of $\mathcal{A}$ with R.

Finally, let us define the kind of configurations that M_1 will produce.

Definition 4 (Admissible configuration). *A configuration ξ is* admissible *if there is an annotated configuration $(\xi, \chi, \mathcal{I}, \mathcal{S}, D)$. In this case, we say that χ is a healthy configuration* satisfying *ξ. Any change to an admissible configuration is called* admissible*, if the resulting configuration is also admissible.*

The following key lemma shows that an admissible configuration can be locally corrected.

Lemma 1 (Correction). *Let Q be a multiple of E. Consider an annotated configuration $\mathcal{A} = (\xi, \chi, \mathcal{I}, \mathcal{S}, D)$, and an interval $R = [a, b]$ of length $2E$, whose ends are divisible by E, and $R_i^j = [a + 0.1iE, b - 0.1jE)$ for $i, j = 0, 2, 4$.*

Assume that either in the left half or the right half of R, at least $E - 3\beta$ cells of $\xi(R)$ are nonempty. Then it is possible to compute from $\xi.c.Core(R)$ an interval $\hat{R} \in \{R, R_0^4, R_4^0\}$, a local configuration $\zeta = \zeta(\hat{R})$ with no empty cells, such that $\chi|\zeta(\hat{R})$ is healthy, and the following holds:

1. *If $\chi.pos \in R_2^2$ then $\hat{R} = R$, $\zeta.pos \in R$, and $\zeta.ZigDepth = 0$.*
   If $\chi.pos < a + 0.2E$ then $\hat{R} = R_4^0$, and $\zeta.pos$ is to the left of $\hat{R}$. Similarly, if $\chi.pos > b - 0.2E$ then then $\hat{R} = R_0^4$, and $\zeta.pos$ is to the right of $\hat{R}$.
2. *The states of nonempty cells of ξ can differ from the corresponding cells of ζ over $\hat{R}$ only in the islands, or in the at most 3β positions between ζ.*front *and χ.*front *in case of $\hat{R} = R$.*
3. *The computation of ζ can be carried out by the machine M_1 (relying only on ξ and R), using a constant number of passes over R, and a constant number of counters of size $\leq Q$.*

3.1 The Recovery Procedure

The recovery procedure opens an interval $R = [a, b]$ to which it applies the correction algorithm of the proof of the Correction Lemma 1.

The recovery procedure is controlled by *Rec.Sw* and *Rec.Addr* and their corresponding fields in a cell. Similarly as before, we define *c.Rec.Core*. With some abuse of notation, we set $c.Rec.Core = 1$ to mean that the cell is *marked* for recovery, but no useful values have been assigned to its *Core* fields. Otherwise, $c.Rec.Core = 0$.

We require that Q is a multiple of E. Let us call an interval *aligned* if its endpoints are divisible by E.

The process makes use of a number of rules: `Alarm`, `Mark`, `Plan`(i), `Mop`(i) for $i = 1, 2$. Whenever we say that a rule "checks" something, it is understood that if the check fails, alarm is called.

1. The rule `Alarm` sets *Mode* $\leftarrow$ Recovering, *Rec.Sw* $\leftarrow 1$, and *Rec.Addr* $\leftarrow 0$.
2. Rule `Mark` locates and marks a recovery area $R = z_1 + [-E, E)$, with *c.Rec.Core* $\leftarrow 1$ as follows.
   (a) It starts from a cell x (where the alarm was called), and moves left to $x - 7\beta$. It remembers the majority of $c.Addr(y) - (y - x) \bmod Q$ for $y \in x + [-7\beta, 0)$ as a candidate modulo Q address λ_{-1} for x. It also computes a majority sweep value σ_{-1} if a majority exists. Now, the machine turns right and while passing over $[x, x + 7\beta)$ it computes λ_1 and σ_1 similarly. Admissibility guarantees that these values exist and are enough to determine aligned interval R with the following properties:
      i. the point x is in R, at least $0.2E$ away from its boundary;
      ii. with respect to sweep direction, the marked area reaches less than $1.2E$ backward from the front χ.front.

From this moment z_1 becomes the center of the recovery and $c.Rec.Addr(z_1) = 0$.

(b) The following rule is going to run simultaneously through all the rest of the recovery procedure.
   i. Update the field *Rec.Addr* in every move, increasing or decreasing it as we move left or right.
   ii. Check the *alignedness* of the interval R every time you pass over an interval of size 7β containing nonempty cells: for all but 3β of the cells $Rec.Addr \equiv c.Addr \pmod{E}$ must hold. If it does not, go back the last 7β steps and call alarm.

(c) In order to mark R, the head moves in a zigging way, similarly to what is done in the main simulation, except that we do not go outside the interval R. Zigging makes sure not to mark too many cells in one sweep or without checking that they are marked consistently with what was marked before. The process is controlled using the fields *Rec.ZigDepth*, *Rec.ZigDir*, and uses the constant parameter $Rec.Z = 11\beta$. It first marks cells in R moving left, then moving right. Every 7β steps forward are accompanied by $Rec.Z$ steps backwards and forward. While zigging, we check $c.Rec.Addr = Rec.Addr$ in the scanned cells. In the zig back the first $Rec.Z$ cells are expected to be marked already: if they are not, alarm is called.
During the third sweep, the head sweeps interval $[z_1 - E, x + 7\beta)$. It then turns and expects to see marked cells over this area. Then, it continues and marks interval $[x + 7\beta, z_1 + E)$.

3. Rule `Calc` carries out, over R, the algorithm of the Correction Lemma 1. If none of the cases apply in the algorithm described in the proof, the rule calls alarm. When $\hat{R} = R_0^4$ (or $\hat{R} = R_4^0$), we put the head to b (or a respectively). In both cases we set $ZigDepth = 0$.
4. Stages Planning_1 and Planning_2 follow each other. Stage Planning_i checks that all cells of R are marked and then calls `Calc`. In case $i = 1$, it writes the resulting $\zeta.c.Core$ values on the $c.Rec.Core$ track of $\hat{R}$, and $c.Rec.Core \leftarrow 1$ into $R \setminus \hat{R}$. In case $i = 2$, it just checks whether the result is equal to the existing values of $c.Rec.Core$.
5. Stages Mopping_1 and Mopping_2 also follow each other. Rule `Mop`(1) unmarks the cells over R, setting $c.Core \leftarrow c.Rec.Core$ at the same time, if $c.Rec.Core \notin \{0, 1\}$.
If $\hat{R} = R_0^4$ then Rule `Mop`(1) moves from the left end of R to the right end while unmarking, and stays there. If it is $\hat{R} = R_4^0$ then it moves from the right end to the left end while unmarking. Otherwise, it first moves backwards from the sweep direction from ζ to the end of R. Then it erases the marks up to position ζ.front. Then Rule `Mop`(2) follows, which is similar, but works from the other direction, ending up at ζ.front with no marked cells. Zigging is used during this stage just as during the marking stage.

Remark 1 (On the choice of R). To explain the need for choosing R in this way (aligned modulo E), assume that a burst of faults creates a stain in one of

the neighbor colonies. If the position of R would depend "continuously" on the position of the cell where alarm is called, then using faults occurring at times distant from each other (and assuming that the simulation never stepped in the neighbor colony), stains can spread slowly without limit, and ruin the majority of *Info* and *State* fields in the neighbor colony.

Computation shows that the recovery procedure needs at most $K_R = 3237\beta$ steps to complete.

4 A Road-Map for the Proof of the Main Theorem

A configuration when a fault occurs is *a noisy configuration*. A computation history is a (β, V)*-noisy trajectory*, if faults in it are confined to bursts of size $\leq \beta$ separated by time intervals of size $\geq V$. A pair of mappings (φ_*, Φ^*) is a (β, V)*-tolerant simulation* of Turing machine M_2 by Turing machine M_1 if for every string $x \in \Sigma_2^*$, every (β, V)-noisy trajectory η of M_1 whose initial configuration is $\varphi_*(x)$, the history $\Phi^*(\eta)$ is a trajectory of M_2.

If the head is in a free cell, in normal mode, then the time (and the configuration) will be called *distress-free*. A time that is not distress-free and was preceded by a distress-free time will be called a *distress event*. Consider a time interval $[t, t+u)$ starting with a distress event and ending with the head becoming free again. It is called a *relief event* of *duration* u if the only possible island that remains from the distress area is due to some burst that occurred at a time intersecting $[t, t+u)$. The *extent* of a relief event is the maximum size interval covering the distress area during the distress.

An *annotated history* is a sequence of annotated configurations if its sequence of underlying configurations is a (β, V)-trajectory, and it satisfies the following requirements.

(a) Islands are only created by noise. Stains and the distress area start out as islands.
(b) Each distress event is followed immediately by a relief event, of duration $\leq 3K_R$ and extent $\leq 3E$.
(c) If a distress-free configuration has $Sw \geq \mathrm{SpSt}$, then the base colony contains no stains from earlier work periods.

The following lemma implies the main theorem.

Lemma 2 (Recovery and simulation). *Assume that machine M_1 starts working on a tape configuration of the form $\varphi_*(x)$.*

Then, every (β, V)-noisy trajectory *of M_1 can be annotated.*

Further, via some simulation function Φ^* the movement of the base colony corresponds to the head movement of the simulated machine M_2 (scaled up by a factor of Q). The array of c.State values in the free cells of the base colony in any configuration with $Sw < \mathrm{Last}(Drift)$ decodes into the state of M_2, and the array of c.Info values at the same times decodes into the current tape cell symbol of M_2.

To prove Lemma 2 we need to show that we can extend the annotation of the history within a constant number of steps. The idea of the proof of relief from damage is the following. If alarm is called and the recovery process is allowed to complete, then it carries out the needed correction, as guaranteed by the Correction Lemma 1. Most complications are due to the fact that the state after a burst is arbitrary. When the mode is normal then zigging guarantees that the effect is limited to near the island where the burst happened: for example, the direction of swing cannot be changed in the middle of the workspace, since then zigging would notice this and call alarm. However, the mode after the burst can be the recovery mode, with arbitrary values for all fields. Moreover, a new burst may occur after an alarm, at an arbitrary stage of the recovery. The "checks and balances" in the recovery mode make sure that even in this case, relief of an extent $\leq 3E$ occurs within at most two runs of the recovery procedure. (This is proved, alas, by painstaking case distinctions.)

References

1. Asarin, E., Collins, P.: Noisy Turing Machines. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1031–1042. Springer, Heidelberg (2005)
2. Çapuni, I., Gács, P.: A Turing Machine Resisting Isolated Bursts Of Faults, http://cs-people.bu.edu/ilir/ftTM/thePaper.pdf
3. Kurdyumov, G.L.: An Example of a Nonergodic One-Dimensional Homogoenous Random Medium With Positive Transition Probabilities. Soviet Math. Dokl. 19(1) (1978)
4. Gács, P.: Reliable Computation with Cellular Automata. Journal of Computer System Science 32(1), 15–78 (1986)
5. Gács, P.: Reliable Cellular Automata with Self-organization. In: Proc. of the 37th IEEE FOCS Symposium, pp. 90–99 (1997)
6. Gács, P., Reif, J.: A simple three-dimensional real-time reliable cellular array. Journal of Computer and System Sciences 36(2), 125–147 (1988)
7. Misra, J., Gries, D.: Finding Repeated Elements. Science of Computer Programming 2, 143–152 (1982)
8. Pippenger, N.: On networks of noisy gates. In: Proc. of the 26th IEEE FOCS Symposium, pp. 30–38 (1985)
9. Spielman, D.: Highly Fault-tolerant Parallel Computation. In: Proc. of the 37th IEEE FOCS Symposium, pp. 154–163 (1996)
10. Toom, A.: Stable and Attractive Trajectories in Multicomponent Systems. In: Dobrushin, R.L. (ed.) Multicomponent Systems. Advances in Probability, vol. 6, pp. 549–575. Dekker, New York (1980) (translation from Russian)
11. von Neumann, J.: Probabilistic Logics And the Synthesis of Reliable Organisms From Unreliable Components. In: Shannon, C., McCarthy (eds.) Automata Studies. Princeton University Press, Princeton (1956)

Properties of SLUR Formulae

Ondřej Čepek[1,2], Petr Kučera[1,⋆], and Václav Vlček[1,⋆⋆]

[1] Department of Theoretical Computer Science and Mathematical Logic, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic
Ondrej.Cepek@mff.cuni.cz,
kucerap@ktiml.ms.mff.cuni.cz,
vlcek@ktiml.mff.cuni.cz
[2] Institute of Finance and Administration (VŠFS), Estonská 500, 100 00 Praha 10, Czech Republic

Abstract. Single look-ahead unit resolution (SLUR) algorithm is a nondeterministic polynomial time algorithm which for a given input formula in a conjunctive normal form (CNF) either outputs its satisfying assignment or gives up. A CNF formula belongs to the SLUR class if no sequence of nondeterministic choices causes the SLUR algorithm to give up on it. The SLUR class is reasonably large. It is known to properly contain the well studied classes of Horn CNFs, renamable Horn CNFs, extended Horn CNFs, and CC-balanced CNFs. In this paper we show that the SLUR class is considerably larger than the above mentioned classes of CNFs by proving that every Boolean function has at least one CNF representation which belongs to the SLUR class. On the other hand, we show, that given a CNF it is coNP-complete to decide whether it belongs to the SLUR class or not. Finally, we define a non-collapsing hierarchy of CNF classes SLUR(i) in such a way that for every fixed i there is a polynomial time satisfiability algorithm for the class SLUR(i), and that every CNF on n variables belongs to SLUR(i) for some $i \leq n$.

Keywords: Boolean functions, CNF satisfiability, unit resolution.

1 Introduction

The satisfiability problem (SAT) is to decide whether a given formula φ in CNF has a satisfying assignment, i.e. whether for some assignment t of values 0 (false) or 1 (true) to variables we have that $\varphi(t)$ evaluates to 1 (true). This problem was the first one shown to be NP-complete [6,10]. Thus, unless P=NP, no polynomial time algorithm can solve this problem. There are, however, many classes of formulae for which polynomial SAT algorithms are known. These classes of formulae include Horn formulae [8,12,14], renamable Horn formulae [13,1], extended Horn formulae [4], and CC-balanced formulae [5]. These four classes share

⋆ The first two authors gratefully acknowledge the support by the Czech Science Foundation (grant No. P202/10/1188).

⋆⋆ The third author gratefully acknowledges the support by the Charles University Grant Agency (grant No. 266111).

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 177–189, 2012.

an interesting property: the satisfiability problem for formulae from these classes can be solved by unit resolution, namely by the single look-ahead unit resolution (SLUR) algorithm [15,9].

The SLUR algorithm works as follows. In every step it nondeterministically chooses a variable and its value and performs unit propagation fixing the values of the variables in unit clauses. If the unit propagation does not yield a contradiction, the algorithm continues with the new formula in the same way. If the contradiction (the empty clause) is derived, the algorithm tries the other value for the chosen variable and continues with the new formula. If both values lead to a contradiction using unit resolutions, the SLUR algorithm gives up. A nice property of Horn, renamable Horn, extended Horn, and CC-balanced formulae is that the SLUR algorithm never gives up on them [15]. Therefore, it is natural to define a generalization of these four classes called the SLUR class as the class of those formulae on which the SLUR algorithm never gives up.

It is quite common that classes of CNFs are used to define classes of Boolean functions. For instance, a Boolean function f is called Horn, if there exists at least one Horn CNF representing f. Similar definitions work in the renamable Horn, extended Horn, CC-balanced, and many other cases, each time defining a subclass of Boolean functions. The first question we answer in this paper is what happens if the same definition is used for the SLUR class. The answer may be somewhat surprising. We prove that every Boolean function can be represented by a CNF which belongs to the SLUR class. In particular we show, that given a Boolean function, its canonical CNF, i.e. the CNF consisting of all prime implicates of the function, is always in the SLUR class. It means, that if we define the class of SLUR functions as those functions which admit at least one representation by a CNF in the SLUR class, then such a class consists of all Boolean functions.

It follows, that being a SLUR CNF is a property of the CNF, not a property of the function it represents. Every function has some CNF representations which belong to the SLUR class and some that do not. The probability that a randomly generated CNF belongs to the SLUR class was studied in [9] where it was proved that a random CNF is not SLUR with high probability. Therefore, it is very natural to ask, whether we can recognize in a polynomial time, if a given CNF belongs to the SLUR class or not. The second result of this paper shows that it is coNP-hard to decide this question, which in turn implies, that no polynomial time algorithm can make such a decision unles P = NP. This solves a problem which was unresolved so far [9].

The third part of this paper deals with a generalization of the SLUR class. The SLUR algorithm is modified in such a way that at every step it nondeterministically chooses not one but i varables and their values. We show that such a hierarchy of classes does not collapse and that each individual class in the hierarchy preserves the main properties of the original SLUR class.

A preliminary version of this paper appeared as [3] (which is a non-archived publication with no ISBN or ISSN distributed in hard copy only to seminar

participants). Because of the page limit we had to omit proofs of several technical lemmas in the current version. An interested reader may contact the authors to obtain the full version of all proofs.

This paper is organized as follows. In Section 2 we introduce necessary definitions and basic known results. In Section 3 we show that a canonical CNF of a Boolean function always belongs to the SLUR class. In Section 4 we show the coNP-completeness of the SLUR membership problem. Finally we close the paper with the hierarchy results in Section 5.

2 Definitions and Results

A *Boolean function* f on n propositional variables $x_1, \ldots, x_n$ is a mapping $\{0,1\}^n \to \{0,1\}$. The propositional variables $x_1, \ldots, x_n$ and their negations $\overline{x}_1, \ldots, \overline{x}_n$ are called *literals* (*positive* and *negative literals*, respectively). An disjunction of literals

$$C = \bigvee_{i \in I} \overline{x}_i \vee \bigvee_{j \in J} x_j \tag{1}$$

is called a *clause*, if every propositional variable appears in it at most once, i.e. if $I \cap J = \emptyset$. The *degree* of a clause C is the number of literals in C. For two Boolean functions f and g we write $f \leq g$ if

$$\forall (x_1, \ldots, x_n) \in \{0,1\}^n \ : \ f(x_1, \ldots, x_n) = 1 \quad \Longrightarrow \quad g(x_1, \ldots, x_n) = 1 \tag{2}$$

Since each clause is itself a Boolean function, formula (2) also defines the meaning of inequalities $C_1 \leq C_2$, $C_1 \leq f$, and $f \leq C_1$, where C_1, C_2 are clauses and f is a Boolean function.

We say that a clause C_1 *subsumes* another clause C_2 if $C_1 \leq C_2$ (e.g. the clause $\overline{x} \vee z$ subsumes the clause $\overline{x} \vee \overline{y} \vee z$). A clause C is called an *implicate* of a function f if $f \leq C$. An implicate C is called *prime* if there is no distinct implicate C' subsuming C, or in other words, an implicate of a function is prime if dropping any literal from it produces a clause which is not an implicate of that function.

It is a well-known fact that every Boolean function f can be represented by a conjunction of clauses (see e.g. [11]). Such an expression is called a *conjunctive normal form* (or CNF) of the Boolean function f. It should be noted that a given Boolean function may have many CNF representations. If two distinct CNFs, say ϕ_1 and ϕ_2, represent the same function, we say that they are *equivalent*, and denote this fact by $\phi_1 \equiv \phi_2$. A CNF ϕ representing a function f is called *prime* if each clause of ϕ is a prime implicate of function f. The unique CNF consisting of all prime implicates of function f is called the *canonical CNF* of f. A CNF ϕ representing a function f is called *irredundant* if dropping any clause from ϕ produces a CNF that does not represent f. We shall often treat a CNF as a set of its clauses.

For a Boolean function f on n variables, a variable x, and a value $e \in \{0,1\}$ we denote by $f[x := e]$ the Boolean function on $n-1$ variables which results from f by assigning the value e to variable x. Similarly, for a CNF φ we denote by $\varphi[x := e]$ the CNF which results from φ by substituting e for all appearances of x (and $1-e$ for all appearances of $\overline{x}$) in φ. A partial assignment is a mapping $p : V \mapsto \{0,1,*\}$, where V is the set of variables and the value $*$ means an unspecified value. Alternatively, we will also identify a partial assignment with a set of literals S where $x \in S$ if $p(x) = 1$, $\overline{x} \in S$ if $p(x) = 0$, and $\{x, \overline{x}\} \cap S = \emptyset$ if $p(x) = *$.

Two clauses C_1 and C_2 are said to be *resolvable* if they contain exactly one complementary pair of literals, i.e. if there exists exactly one propositional variable that appears uncomplemented in one of the clauses and complemented in the other. That means that we can write $C_1 = \tilde{C}_1 \vee x$ and $C_2 = \tilde{C}_2 \vee \overline{x}$ for some propositional variable x and clauses $\tilde{C}_1$ and $\tilde{C}_2$ which contain no complementary pair of literals. The clauses C_1 and C_2 are called *parent clauses* and the disjunction $R(C_1, C_2) = \tilde{C}_1 \vee \tilde{C}_2$ is called the *resolvent* of the parent clauses C_1 and C_2. Note that the resolvent is a clause (does not contain a propositional variable and its negation). A clause which consists of a single literal is called a *unit clause* and resolution in which one of the parent clauses is a unit clause is called *unit resolution*.

The following is an easy lemma [2].

Lemma 1. *Let C_1 and C_2 be two resolvable implicates of a Boolean function f. Then $R(C_1, C_2)$ is also an implicate of f.*

The wellknown satisfiability problem is defined as follows.

Satisfiability (SAT)	
Instance :	A formula φ in CNF.
Question :	Is there an assignment t to variables of φ such that $\varphi(t) = 1$?

Although this problem is NP-complete in general [6,10], there are many classes of Boolean formulas, for which SAT problem can be solved in polynomial time. One of these classes is defined using the SLUR (or single-look ahead unit resolution) algorithm [15,9]. The basic operation used by this algorithm is unit propagation. Function *unitprop*(φ) for a given formula φ in CNF returns a pair of values (φ', t), where φ' is the CNF formula that results from repeatedly performing *unit resolution* until no unit clauses remain in the formula, and t is the partial assignment which satisfies unit clauses found and eliminated during unit propagation. It is known, that unitprop can be implemented in time linear in the length of formula φ [7].

Algorithm 1. SLUR(φ)

Input: A CNF formula φ with no empty clause
Output: A satisfying partial truth assignment for the variables in φ, or "unsatisfiable", or "give up".

```
 1: (φ, t) := unitprop(φ)
 2: if φ contains the empty clause then return "unsatisfiable" endif
 3: while φ is not empty
 4: do
 5:    Select a variable v appearing as a literal of φ
 6:    (φ1, t1) := unitprop(φ ∧ v̄)
 7:    (φ2, t2) := unitprop(φ ∧ v)
 8:    if both φ1 and φ2 contain the empty clause then return "give up"
       endif
 9:    if φ1 contains the empty clause
10:    then
11:       (φ, t) := (φ2, t ∪ t2)
12:    else if φ2 contains the empty clause
13:    then
14:       (φ, t) := (φ1, t ∪ t1)
15:    else
16:       Arbitrarily do one of the following:
17:          (φ, t) := (φ1, t ∪ t1)
18:          (φ, t) := (φ2, t ∪ t2)
19:    enddo
20:    return t
```

We shall say, that CNF φ is a SLUR CNF, if for all possible sequences of nondeterministic choices in steps 5 and 16, the SLUR algorithm does not give up. If the SLUR algorithm gives up on a CNF φ then we shall say that the SLUR algorithm *gives up on a CNF φ with a variable v and an assignment t*, if v is the last selected variable in step 5 before giving up and t is the assignment to the variables at the time of giving up (not including value of v and values assigned by unit propagation which follows after the selection of v). Finally, the SLUR class is the set of all SLUR CNFs.

3 Every Canonical CNF Is a SLUR CNF

In this section we shall show that every Boolean function f has a representation by a SLUR CNF. We shall start by showing that if a CNF φ contains all prime implicates of a function f it represents, then this property is preserved under partial assignment.

Lemma 2. *Let f be an arbitrary Boolean function on variables $x_1, \ldots, x_n, x_{n+1}$ and let φ be a CNF which contains all prime implicates of f. Let $e \in \{0, 1\}$ and*

let $f' = f[x_{n+1} := e]$, then $\varphi' := \varphi[x_{n+1} := e]$ contains all prime implicates of f'. I.e. if φ is the canonical CNF of f then after subsumed clauses are removed from φ', it is also a canonical CNF of f'.

Proof. Let us without loss of generality assume that $e = 0$ (the case $e = 1$ is similar). Let C be a prime implicate of f'. Because it is an implicate of $f' = f[x_{n+1} := 0]$, it follows that $C \vee x_{n+1}$ is an implicate of f. This is because if $f(t) = 1$ for some assignment $t \in \{0,1\}^{n+1}$, then either $t[x_{n+1}] = 0$ in which case also $C(t) = 1$ and hence $(C \vee x_{n+1})(t) = 1$ (C is an implicate of f'), or $t[x_{n+1}] = 1$ in which case $(C \vee x_{n+1})(t) = 1$ as well. In either case $f(t) = 1$ implies $(C \vee x_{n+1})(t) = 1$ for any assignment t and hence $C \vee x_{n+1}$ is an implicate of f.

Let C' be a prime implicate of f which subsumes $(C \vee x_{n+1})$. If x_{n+1} is not contained in C', then C' is in fact an implicate of f'. This is because if $f'(t) = 1$ for an assignment $t \in \{0,1\}^n$, then $f(t,0) = 1$, hence $C'(t,0) = 1$ and hence $C'(t) = 1$ because C' does not depend on x_{n+1}. In this case C' also subsumes C and hence $C = C'$. Since C' is a prime implicate of f, it is contained in φ and because it does not depend on x_{n+1}, we get that $C = C'$ is contained in φ'.

If x_{n+1} is contained in C', then let $C'' = C'[x_{n+1} := 0]$ (i.e. C'' is C' with x_{n+1} removed). C'' is an implicate of f', it is in fact directly contained in φ', because C' as a prime implicate of f is contained in φ. C'' subsumes C and hence $C = C''$ because C is a prime implicate of f'. Thus also in this case $C \in \varphi'$. □

Now it is not hard to show that the canonical representation of a Boolean function f always belongs to the SLUR class.

Theorem 2. *Let f be a Boolean function and let φ be its canonical CNF, then φ is a SLUR CNF.*

Proof. The canonical CNF has the property, that φ is unsatisfiable, if and only if it contains only the empty clause. Thus if φ is unsatisfiable then it is recognized by the SLUR algorithm in step 2. If the SLUR algorithm proceeds to the while cycle in step 3, it means that φ is satisfiable. In this case let v be an arbitrary variable chosen by the SLUR algorithm in step 5, then $\varphi[v := 0]$ contains all prime implicates of $f[v := 0]$ and $\varphi[v := 1]$ contains all prime implicates of $f[v := 1]$. Because φ is satisfiable, one of $\varphi \wedge \overline{v}$ and $\varphi \wedge v$ is satisfiable as well, let us assume without loss of generality, that it is $\varphi \wedge \overline{v} \equiv \varphi[v := 0]$. Because $unitprop(\varphi \wedge \overline{v})$ consists of repeated partial assignments of values to unit clauses, we can use for each of these partial assignments Lemma 2 to show that φ_1 generated by $unitprop(\varphi \wedge \overline{v})$ contains all prime implicates of $f(t_1)$ where t_1 is the partial assignment defined on line 6 of the SLUR algorithm. Because $\varphi \wedge \overline{v}$ is satisfiable and this property is preserved during unit propagation as it only assigns values forced by unit clauses, we have that φ_1 is satisfiable and hence it does not contain the empty clause. Since φ_1 contains all prime implicates of $f(t_1)$, we can use induction to show that the SLUR algorithm never gives up on a canonical CNF. □

It should be noted that a canonical CNF may be exponentially long w.r.t. the input CNF. In fact even if the canonical CNF is short (e.g. if the input CNF is unsatisfiable) generating it is still a computationally hard problem as it solves

the SAT problem for the input CNF. It is an interesting open question whether given a CNF, there exists a polynomially long logically equivalent SLUR CNF, i.e. whether it suffices to add polynomially many prime implicates of the input CNF to the input formula to get into the SLUR class.

4 Testing Whether a Given CNF Is SLUR Is coNP Complete

In this section we shall show that the following problem is coNP complete.

SLUR MEMBERSHIP (SLUR)	
Instance :	A CNF φ.
Question :	Does φ belong to the SLUR class?

We shall use transformation from 3D Matching problem, which is known for a very long time to be NP-complete [10].

3D MATCHING (3DM)	
Instance :	Sets X, Y, Z of the same size $\|X\| = \|Y\| = \|Z\| = q$ and a set of triples $W \subseteq X \times Y \times Z$.
Question :	Does W contain a matching of size q? I.e. is there a subset $M \subseteq W$ of size $\|M\| = q$ such that for any two different triples $E, E' \in M$ we have that $E \cap E' = \emptyset$?

We shall associate a CNF φ_W with every instance X, Y, Z, W of 3DM in the following way. We shall assume that $X = \{x_1, \ldots, x_q\}$, $Y = \{y_1, \ldots, y_q\}$, $Z = \{z_1, \ldots, z_q\}$, and $W = \{E_1, \ldots, E_w\}$, where $w = |W|$. We shall assume, that $E_j = (x_{f(j)}, y_{g(j)}, z_{h(j)})$, where f, g, and h are functions determining which elements of X, Y, and Z belong to E_j.

Let us denote the sets of variables in φ_W as follows:

$$\begin{aligned}
\mathcal{A} &= \{a_1, \ldots, a_{q+1}, u\}\ , \\
\mathcal{B}_i &= \{b_i^1, \ldots, b_i^w\} \qquad \text{for every } i \in \{1, \ldots, q\}, \\
\mathcal{B} &= \bigcup_{i=1}^{q} \mathcal{B}_i\ , \\
\mathcal{C}_i &= \{c_i^1, \ldots, c_i^w\} \qquad \text{for every } i \in \{1, \ldots, q\}, \\
\mathcal{C} &= \bigcup_{i=1}^{q} \mathcal{C}_i\ , \text{ and} \\
\mathcal{V} &= \mathcal{A} \cup \mathcal{B} \cup \mathcal{C}
\end{aligned}$$

– For every $i \in \{1, \ldots, q\}$ let us denote $A_i = (a_i \vee \overline{a_{i+1}})$, where $a_1, \ldots, a_{q+1}$.

- For every $i \in \{1, \ldots, q\}$ and $j \in \{1, \ldots, w\}$ let us denote $B_i^j = (\overline{b_i^1} \vee \cdots \vee \overline{b_i^{j-1}} \vee b_i^j \vee \overline{b_i^{j+1}} \vee \cdots \vee \overline{b_i^w})$, i.e. B_i^j denotes a clause on variables $b_i^1, \ldots, b_i^w$, in which every literal is negative except b_i^j.
- For every $i \in \{1, \ldots, q\}$ and $j \in \{1, \ldots, w\}$ let us denote $C_i^j = (\overline{c_i^1} \vee \cdots \vee \overline{c_i^{j-1}} \vee c_i^j \vee \overline{c_i^{j+1}} \vee \cdots \vee \overline{c_i^w})$, i.e. C_i^j denotes a clause on variables $c_i^1, \ldots, c_i^w$, in which every literal is negative except c_i^j.
- Given a triple $E_j \in W$, let $D_j = (A_{f(j)} \vee B_{g(j)}^j \vee C_{h(j)}^j)$.
- Finally let $\varphi_W = \bigwedge_{j=1}^w D_j \wedge (a_{q+1} \vee a_1) \wedge (\overline{a_1} \vee u) \wedge (\overline{u} \vee \overline{a_1})$.

We shall start by showing some properties of φ_W which will be useful in the subsequent proofs. We say, that a clause is trivial if it is empty or it evaluates to 1, otherwise, we say, it is nontrivial.

Lemma 3. *Let $t : \mathcal{V} \mapsto \{0, 1, *\}$ be any partial assignment of variables in φ_W, and let $j, k \in \{1, \ldots, w\}$ be arbitrary and let us assume that both $D_j(t)$ and $D_k(t)$ are nontrivial. Then $D_j(t)$ and $D_k(t)$ are resolvable if and only if $E_j \cap E_k = \emptyset$, i.e. D_j and D_k are resolvable over a variable in $\mathcal{A}$, and moreover*

- *$f(j) = f(k) + 1$ and $t[a_{f(j)}] = *$, or*
- *$f(j) = f(k) - 1$ and $t[a_{f(k)}] = *$.*

Proof. Let us at first assume that $D_j(t)$ and $D_k(t)$ are resolvable, which means that they have a conflict in exactly one variable. It immediately follows that $j \neq k$. Let us at first show, that E_j and E_k must be disjoint. For contradiction let us at first assume, that they share an element of Z, i.e. that $h(j) = h(k)$. In this case also $C_{h(j)}^j$ and $C_{h(k)}^k$ are on the same set of variables $\mathcal{C}_{h(j)} = \mathcal{C}_{h(k)}$. But according to definition of $C_{h(j)}^j$ and $C_{h(k)}^k$ we have that they have conflict in two variables $c_{h(j)}^j$ and $c_{h(j)}^k$. If t assigns value to neither of them, then D_j and D_k would not be resolvable, if t assigns value to one of them then no matter which value it is, we have that one of $D_j(t)$ and $D_k(t)$ would be equal to constant 1. Hence it must be the case that $h(j) \neq h(k)$. Similarly we can show that $g(j) \neq g(k)$ we only consider set Y instead of Z. It also follows, that if $D_j(t)$ has one conflict variable with $D_k(t)$, then it must be either $a_{f(j)}$, or $a_{f(k)}$. The statement of the lemma follows from definition of $A_{f(j)}$ and $A_{f(k)}$.

Now let us assume, that $E_j \cap E_k = \emptyset$, $f(j) = f(k) + 1$, and $t[a_{f(j)}] = *$ (the case when $f(j) = f(k) - 1$ is similar and is left to the reader). In this case there is only one conflict variable of D_j and D_k and it is $a_{f(j)}$, which is left intact by t and hence it is also a conflict variable of $D_j(t)$ and $D_k(t)$, which means that $D_j(t)$ and $D_k(t)$ are resolvable. □

Lemma 4. *Let $t : \mathcal{V} \mapsto \{0, 1, *\}$ be any partial assignment of variables in φ_W and let $j \in \{1, \ldots, w\}$. If $D_j(t)$ is not equal to 1 and it contains exactly one literal with variable in $\mathcal{B}_{g(j)}$ ($\mathcal{C}_{h(j)}$ respectively), then no other clause in $\varphi_W(t)$ can contain a variable in $\mathcal{B}_{g(j)}$ ($\mathcal{C}_{h(j)}$ respectively).*

Proof. The proof is similar to the proof of Lemma 3. For details see [3]. □

As an easy corollary of Lemma 4 we get the following.

Corollary 1. *Let $t : \mathcal{V} \mapsto \{0, 1, *\}$ be any partial assignment of variables in φ_W such that $\varphi_W(t)$ does not contain the empty clause and let $v \in \mathcal{B} \cup \mathcal{C}$ be a variable, which is not set by t (i.e. $t(v) = *$). Then one of $\varphi_W(t)[v := 0]$ and $\varphi_W(t)[v := 1]$ does not contain the empty clause.*

Proof. If the empty clause occurs in $\varphi_W(t)[v := 0]$, then it is because literal u is present in $\varphi_W(t)$, according to Lemma 4 we have that this literal u is the only occurrence of variable u in $\varphi_W(t)$, hence setting u to 1 cannot produce the empty clause. □

Now we are ready to show that the only variables whose assignment can cause SLUR to give up belong to $\mathcal{A}$.

Lemma 5. *If the SLUR algorithm gives up on φ_W with variable v and assignment $t : \mathcal{V} \mapsto \{0, 1, *\}$, then $v \in \mathcal{A}$.*

Proof. If SLUR algorithm gives up on φ_W with variable v and assignment t, then unit propagation on both $\varphi_W(t)[v := 0]$ and $\varphi_W(t)[v := 1]$ generate the empty clause. Let us assume by contradiction that $v \in \mathcal{B} \cup \mathcal{C}$.

If the empty clause is directly present in $\varphi_W(t)[v := 0]$ or $\varphi_W(t)[v := 1]$, it would mean that we have a unit clause in $\varphi_W(t)$, which is not possible, because performing unit propagation each time after assigning a value to a variable ensures, that in SLUR algorithm at the begining of *while* cycle, formula φ contains neither a unit clause nor the empty clause.

It follows that both $\varphi_W(t)[v := 0]$ and $\varphi_W(t)[v := 1]$ contain new unit clause which causes unit propagation to generate the empty clause, let us assume, that $v = b_i^j$ for some $i \in \{1, \ldots, q\}$ and $j \in \{1, \ldots, w\}$ (the case of variable from $\mathcal{C}$ is the same). Let D_k be a clause for which $g(k) = i$ and $D_k(t)$ is a quadratic clause containing a literal on variable b_i^j. If the other variable in $D_k(t)$ belongs to $\mathcal{B} \cup \mathcal{C}$, then no resolution can occur by setting b_i^j to 0 or 1 according to Lemma 3, and therefore unit propagation cannot generate the empty clause. If the other variable belongs to $\mathcal{A}$, then b_i^j occurs only once in $\varphi_W(t)$ according to Lemma 4 and therefore $D_k(t)[b_i^j := 0] \equiv 1$ or $D_k(t)[b_i^j := 1] \equiv 1$, which means that in one of these cases unit propagation will not do anything and in particular it will not generate the empty clause. □

Lemma 6. *If the SLUR algorithm gives up on φ_W with variable $v \in \mathcal{A}$ and assignment $t : \mathcal{V} \mapsto \{0, 1, *\}$, then $A_1, \ldots, A_q, (a_{q+1} \vee a_1), (\overline{a_1} \vee u), (\overline{u} \vee \overline{a_1})$ are all clauses in $\varphi_W(t)$.*

Proof. The idea of the proof is as follows. According to Lemma 3 if two clauses in $\varphi_W(t')$ for any partial assignment t' are resolvable, then they are resolvable over a variable in $\mathcal{A}$, thus to generate the empty clause in $\varphi_W(t)[v := 0]$ we must follow chain of resolutions over variables from $\mathcal{A}$, the only chain of such resolutions in whole formula is formed by clauses among $A_1, \ldots, A_q, (a_{q+1} \vee a_1), (\overline{a_1} \vee u), (\overline{u} \vee \overline{a_1})$. If all these clauses are present in $\varphi_W(t)$, then setting any variable in $\mathcal{A}$ to any value causes unit propagation to generate the empty clause. On the other hand, we shall show, that if this chain is broken, i.e. some of these clauses do not

appear directly in $\varphi_W(t)$ (although thay may appear as a proper subclause of a clause in $\varphi_W(t)$), then in one of the assignments unit propagation stops before it generates the empty clause. For details see [3]. □

We shall show, that φ_W is not SLUR if and only if W contains a perfect matching.

Lemma 7. *Instance X, Y, Z, W of 3DM contains a perfect matching if and only if φ_W is not SLUR.*

Proof. First let us assume, that W contains a perfect matching $M \subseteq W$, and let $M = \{E_{j_1}, \ldots, E_{j_q}\}$. Let t be a partial assignment, which assigns variables in $B^{j_1}_{g(j_1)}, \ldots, B^{j_q}_{g(j_q)}$ and $C^{j_1}_{h(j_1)}, \ldots, C_{h(j_q)}$ and no other variables, moreover this assignment sets the variables in such a way that $B^j_{g(j)}(t) = 0$ and $C^j_{h(j)}(t) = 0$ for every $j \in \{j_1, \ldots, j_q\}$. Note, that since triples in M are pairwise disjoint, all variables appear at most once in $B^{j_1}_{g(j_1)}, \ldots, B^{j_q}_{g(j_q)}$ and $C^{j_1}_{h(j_1)}, \ldots, C_{h(j_q)}$, and thus such assignment t exists and is unique. If SLUR chooses variables assigned by t in any order and if it chooses their values according to t, then we can observe, that no unit resolution will ever occur, because no unit clause will be produced by these assignments. Hence the SLUR algorithm will not fail in the process and in each step it will be able to choose the assignment of a variable according to t. When all variables are set according to t, we thus get formula $\varphi_W(t)$, in which from every clause D_j for $j \in \{j_1, \ldots, j_q\}$ remained only A_j. Thus we get, that $\varphi_W(t)$ contains the following subformula:

$$\left(\bigwedge_{j \in \{j_1, \ldots, j_q\}} A_j \right) \wedge (a_{q+1} \vee a_1)(\overline{a_1} \vee u)(\overline{u} \vee \overline{a_1}) \tag{3}$$

Because M is a matching and thus every $x_i \in X$ appears in exactly one triple of M, we get that CNF (3) is equivalent to:

$$\left(\bigwedge_{j=1}^{q} A_j \right) \wedge (a_{q+1} \vee a_1)(\overline{a_1} \vee u)(\overline{u} \vee \overline{a_1}) \tag{4}$$

It follows, that $\varphi_W(t)$ is actually equivalent to (4), because each clause in $\varphi_W(t)$ is absorbed by one of the clauses in (4). Using definition of A_j we get that this is equivalent to:

$$(a_1 \vee \overline{a_2})(a_2 \vee \overline{a_3}) \ldots (a_q \vee \overline{a_{q+1}}) \wedge (a_{q+1} \vee a_1)(\overline{a_1} \vee u)(\overline{u} \vee \overline{a_1}) \tag{5}$$

Observe, that CNF (5) is an unsatisfiable quadratic CNF (it reduces to $a_1 \wedge \overline{a_1}$ after resolutions are made) and therefore it is not SLUR. Hence whole formula φ_W is not SLUR.

Now let us assume that SLUR fails on φ_W. According to Lemma 5 it must fail on a variable from $\mathcal{A}$, let $t : \mathcal{V} \mapsto \{0, 1, *\}$ be a partial assignment produced by SLUR algorithm before it chooses a variable on which it fails. Then according to

Lemma 6 we must have that all clauses $A_1, \ldots, A_q \in \varphi_W(t)$, let $D_{j_1}, \ldots, D_{j_q}$ be some clauses such that $D_{j_1}(t) = A_1, \ldots, D_{j_q}(t) = A_q$. Now let $a, b \in \{1, \ldots, q\}$ such that $a \neq b$. Because $A_a \neq A_b$ we have that $D_{j_a} \neq D_{j_b}$, and thus $j_a \neq j_b$. Because $f(j_a) = a$ and $f(j_b) = b$, we also have that $x_{f(j_a)} \neq x_{f(j_b)}$. By the same arguments as in only if part of the proof of Lemma 3 we can show, that in fact $E_{j_a} \cap E_{j_b} = \emptyset$. In particular let us assume that $h(j_a) = h(j_b)$ (the case with $g(j_a) = g(j_b)$ is the same), in this case both $C^{j_a}_{h(j_a)}$ and $C^{j_b}_{h(j_b)}$ are on the same set of variables $\mathcal{C}_{h(j_a)} = \mathcal{C}_{h(j_b)}$ and each of these variables is assigned a value 0 or 1 by t. By definition there is a variable, in which $C^{j_a}_{h(j_a)}$ and $C^{j_b}_{h(j_b)}$ have a conflict in a variable, that implies that one of $C^{j_a}_{h(j_b)}(t)$ and $C^{j_b}_{h(j_b)}(t)$ is evaluated to 0, which is in contradiction with fact that $D_{j_a}(t) = A_a$ and $D_{j_b}(t) = A_b$. □

The main result of this paper is contained in the following theorem.

Theorem 3. *Problem SLUR membership is coNP-complete.*

Proof. Problem SLUR membership belongs to coNP because if we are given sequence of nondeterministic choices made by SLUR algorithm in steps 5 and 16, we can check in polynomial time if these choices lead to giving up. The coNP hardness follows from the transformation described in this section and in particular from Lemma 7. □

5 Hierarchy SLUR(i)

In this section we shall show how the SLUR class can be easily generalized into a hierarchy of classes of CNFs. For each fixed $i \geq 1$ we define a class SLUR(i) as follows. Instead of selecting a single variable on line 5, the parametrized version of the SLUR(φ) algorithm (let us denote it by SLUR(i,φ)) nondeterministically selects i variables, and instead of running unit propagation after substituting the two possible values for the selected variable on lines 6 and 7 it runs unit propagation on all possible 2^i assignments. If all assignments produce the empty clause in the first iteration (after selecting the first i-tuple of variables) SLUR(i,φ) returns "unsatisfiable" (note that this constitutes a difference between SLUR(φ) and SLUR(1, φ)). If all assignments produce the empty clause in any of the subsequent iterations SLUR(i,φ) gives up. If at least one of the assignments does not produce the empty clause SLUR(i,φ) nondeterministically chooses one of such assignments and continues in the same manner. The class SLUR(i) is then defined as the class of CNFs φ on which SLUR(i,φ) never gives up regardless of the choices made. Note that the SLUR class is a strict subset of SLUR(1) since every CNF on which the SLUR algorithm gives up after selecting the first variable is not in the SLUR class but belongs to SLUR(1).

It is obvious from the definition, that SLUR(i,φ) provides for every fixed i a polynomial time SAT algorithm with respect to the length of the input CNF φ (of course, the time complexity grows exponentially in i). It is also clear from the definition that every CNF on n variables belongs to SLUR(n), and hence the hierarchy (i.e. the infinite union of SLUR(i) classes) contains all CNFs.

In the rest of this section we show two more results. First, we prove, that the SLUR(i) hierarchy does not collapse, and then we extend the coNP-completeness recognition result for SLUR(1) to SLUR(i) for an arbitrary i.

Theorem 4. *For each i there is a formula F_{i+1} such that $F_{i+1} \in SLUR(i+1) \setminus SLUR(i)$.*

Proof. Let us construct F_{i+1} as the zero function expressed by the CNF formula on $i+2$ variables $V = \{x_1, \ldots, x_{i+2}\}$. It contains each possible combination of positive and negative literals and can be written as:

$$F_{i+1} = \bigwedge_{P \subseteq V} \left(\bigvee_{v \in P} v \vee \bigvee_{v \in V \setminus P} \overline{v} \right).$$

If we assign values to any i-tuple of variables some of the clauses disappear and the rest of the formula expresses the zero function using two variables (e.g. if we assign arbitrary zero-one values to $x_1, \ldots, x_i$, we get $(x_{i+1} \vee x_{i+2}) \wedge (x_{i+1} \vee \overline{x_{i+2}}) \wedge (\overline{x_{i+1}} \vee x_{i+2}) \wedge (\overline{x_{i+1}} \vee \overline{x_{i+2}})$). *Unitprop* will not derive the empty clause on such CNF and the algorithm SLUR(i,F_{i+1}) will have to give up in the next step.

On the other hand, if we assign values to any $(i+1)$-tuple we will get a CNF of the $x \wedge \overline{x}$ form. *Unitprop* will derive the empty clause on such a formula and so the algorithm SLUR($i+1$,F_{i+1}) returns "unsatisfiable". □

Theorem 5. *For each i the membership problem for the class SLUR(i) is coNP-complete.*

Proof. Similarly as in the SLUR case, the order of selected variables and the values assigned to them which force SLUR(i,φ) to give up on φ, serve as a polynomially verifiable certificate that φ is not in SLUR(i). Therefore, the membership problem for SLUR(i) is in coNP.

To prove coNP-hardness we modify the proof used for the SLUR case. We take i copies of formula φ_W defined in Section 4, each of them on a new set of variables, and add one more clause containing disjunction of another i new variables. That is, given an instance of 3DM we construct a CNF

$$\varphi = {\varphi_W}^{(1)} \wedge {\varphi_W}^{(2)} \wedge \ldots \wedge {\varphi_W}^{(i)} \wedge (n_1 \vee \ldots \vee n_i).$$

It now suffices to prove that the input instance of 3DM contains a perfect matching if and only if φ does not belong to the class SLUR(i).

Let us first assume that the input instance of 3DM contains a perfect matching. Using Lemma 7 we get that the original SLUR algorithm gives up on φ_W for some order of variable assignments. Now we construct an order of variable assignments which will force SLUR(i,φ) to give up on φ. First the algorithm picks the dummy variables $\{n_1, \ldots n_i\}$ and assigns some values that lead to the satisfying assignment of the last clause of φ (this step prevents the algorithm to return "unsatisfiable" instead of giving up in the case when the original SLUR algorithm gives up after assigning just one variable). Now SLUR(i,φ) will follow

the order of variable assignments by using the one which forces the SLUR algorithm to give up on φ_W. Every time the SLUR algorithm pick a variable and assigns a value to it, SLUR(i,φ) does the same in all i copies of φ_W which sit inside of φ. This order clearly leads SLUR(i,φ) to give up on φ, proving that φ does not belong to the class SLUR(i).

Now let us assume that φ does not belong to the class SLUR(i). That means that SLUR(i,φ) gives up on φ for some order of variable assignments. Since φ consists of $i+1$ subCNFs on disjoint sets of variables, it follows that SLUR(i,φ) derives the empty clause by unit propagation from one of these subCNFs for all possible assignments of the last selected i-tuple of variables. Clearly, this cannot happen for the last clause in φ, so it must happen for one of the copies of φ_W. But now restricting the variable assignment only to the variables from this particular copy of φ_W, we get an order of variable assignments which makes the original SLUR algorithm give up on φ_W. Thus φ_W is not SLUR and using Lemma 7 we get that the input instance of 3DM contains a perfect matching. □

References

1. Aspvall, B.: Recognizing disguised nr(1) instances of the satisfiability problem. Journal of Algorithms 1(1), 97–103 (1980)
2. Buning, H.K., Letterman, T.: Propositional Logic: Deduction and Algorithms. Cambridge University Press, New York (1999)
3. Čepek, O., Kučera, P.: Various notes on SLUR formulae. In: Proceedings of the 13th Czech-Japan Seminar on Data Analysis and Decision Making in Service Science, Otaru, Japan, pp. 85–95 (2010)
4. Chandru, V., Hooker, J.N.: Extended horn sets in propositional logic. J. ACM 38(1), 205–221 (1991)
5. Conforti, M., Cornuéjols, G., Vuskovic, K.: Balanced matrices. Discrete Mathematics 306(19-20), 2411–2437 (2006)
6. Cook, S.A.: The complexity of theorem-proving procedures. In: STOC 1971: Proceedings of the Third Annual ACM Symposium on Theory of Computing, New York, NY, USA, pp. 151–158 (1971)
7. Dala, M., Etherington, D.W.: A hierarchy of tractable satisfiability problems. Information Processing Letters 44(4), 173–180 (1992)
8. Dowling, W.F., Gallier, J.H.: Linear time algorithms for testing the satisfiability of propositional horn formulae. Journal of Logic Programming 3, 267–284 (1984)
9. Franco, J., Van Gelder, A.: A perspective on certain polynomial-time solvable classes of satisfiability. Discrete Appl. Math. 125(2-3), 177–214 (2003)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, San Francisco (1979)
11. Genesereth, M.R., Nilsson, N.J.: Logical Foundations of Artificial Intelligence. Morgan Kaufmann, Los Altos (1987)
12. Itai, A., Makowsky, J.A.: Unification as a complexity measure for logic programming. Journal of Logic Programming 4, 105–117 (1987)
13. Lewis, H.R.: Renaming a set of clauses as a horn set. J. ACM 25(1), 134–135 (1978)
14. Minoux, M.: Ltur: A simplified linear time unit resolution algorithm for horn formulae and computer implementation. Information Processing Letters 29, 1–12 (1988)
15. Schlipf, J.S., Annexstein, F.S., Franco, J.V., Swaminathan, R.P.: On finding solutions for extended horn formulas. Inf. Process. Lett. 54(3), 133–137 (1995)

Unique-Maximum and Conflict-Free Coloring for Hypergraphs and Tree Graphs

Panagiotis Cheilaris[1], Balázs Keszegh[2], and Dömötör Pálvölgyi[3]

[1] Center for Advanced Studies in Mathematics,
Ben-Gurion University, Be'er Sheva, Israel
[2] Alfréd Rényi Institute of Mathematics, Budapest, Hungary
[3] Eötvös University, Budapest, Hungary

Abstract. We investigate the relationship between two kinds of vertex colorings of hypergraphs: unique-maximum colorings and conflict-free colorings. In a unique-maximum coloring, the colors are ordered, and in every hyperedge of the hypergraph the maximum color in the hyperedge occurs in only one vertex of the hyperedge. In a conflict-free coloring, in every hyperedge of the hypergraph there exists a color in the hyperedge that occurs in only one vertex of the hyperedge. We define corresponding unique-maximum and conflict-free chromatic numbers and investigate their relationship in arbitrary hypergraphs. Then, we concentrate on hypergraphs that are induced by simple paths in tree graphs.

1 Introduction

A hypergraph H is a pair (V, E), where E (the hyperedge set) is a family of non-empty subsets of V (the vertex set). A vertex coloring of a hypergraph $H = (V, E)$ is a function $C\colon V \to \mathbb{Z}^+$.

A hypergraph is a generalization of a graph. Therefore, it is natural to consider how to generalize proper vertex coloring of a graph to a vertex coloring of a hypergraph. (In a proper vertex coloring of a graph, any two vertices neighboring with an edge in the graph have to be assigned different colors by the coloring function C.) Vertex coloring in hypergraphs can be defined in many ways, so that restricting the definition to simple graphs coincides with proper graph coloring.

At one extreme, it is only required that the vertices of each hyperedge are not all colored with the same color (except for singleton hyperedges). This is called a *non-monochromatic* coloring of a hypergraph. The minimum number of colors necessary to color in such a way a hypergraph H is the (non-monochromatic) chromatic number of H, denoted by $\chi(H)$.

At the other extreme, we can require that the vertices of each hyperedge are all colored with different colors. This is called a *colorful* or *rainbow* coloring of H and we have the corresponding rainbow chromatic number of H, denoted by $\chi_{\mathrm{rb}}(H)$.

In this paper we investigate the following two types of vertex colorings of hypergraphs that are between the above two extremes.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 190–201, 2012.

Definition 1. A *unique-maximum coloring* of $H = (V, E)$ with k colors is a function $C\colon V \to \{1, \ldots, k\}$ such that for each $e \in E$ the *maximum color in* e *occurs exactly once* on the vertices of e. The minimum k for which a hypergraph H has a unique-maximum coloring with k colors is called the *unique-maximum chromatic number* of H and is denoted by $\chi_{\mathrm{um}}(H)$.

Definition 2. A *conflict-free coloring* of $H = (V, E)$ with k colors is a function $C\colon V \to \{1, \ldots, k\}$ such that for each $e \in E$ there is *a color in* e *that occurs exactly once* on the vertices of e. The minimum k for which a hypergraph H has a conflict-free coloring with k colors is called the *conflict-free chromatic number* of H and is denoted by $\chi_{\mathrm{cf}}(H)$.

We also introduce a new coloring, that proves useful in showing lower bounds, and that could be of independent interest.

Definition 3. An *odd coloring* of $H = (V, E)$ with k colors is a function $C\colon V \to \{1, \ldots, k\}$ such that for each $e \in E$ there is *a color that occurs an odd number of times* on the vertices of e. The minimum k for which a hypergraph H has an odd coloring with k colors is called the *odd chromatic number* of H and is denoted by $\chi_{\mathrm{odd}}(H)$.

Every rainbow coloring is unique-maximum, every unique-maximum coloring is conflict-free, and every conflict-free coloring is odd and non-monochromatic. Therefore, for every hypergraph H, $\max(\chi(H), \chi_{\mathrm{odd}}(H)) \leq \chi_{\mathrm{cf}}(H) \leq \chi_{\mathrm{um}}(H) \leq \chi_{\mathrm{rb}}(H)$. Note that an odd coloring can be monochromatic.

The study of conflict-free coloring hypergraphs started in [8,19], with an emphasis in hypergraphs induced by geometric shapes. The main application of conflict-free coloring is that it models a frequency assignment for cellular networks. A cellular network consists of two kinds of nodes: *base stations* and *mobile agents*. Base stations have fixed positions and provide the backbone of the network; they are represented by vertices in V. Mobile agents are the clients of the network and they are served by base stations. This is done as follows: Every base station has a fixed frequency; this is represented by the coloring C, i.e., colors represent frequencies. If an agent wants to establish a link with a base station it has to tune itself to this base station's frequency. Since agents are mobile, they can be in the range of many different base stations. To avoid interference, the system must assign frequencies to base stations in the following way: For any range, there must be a base station in the range with a frequency that is not used by some other base station in the range. One can solve the problem by assigning n different frequencies to the n base stations. However, using many frequencies is expensive, and therefore, a scheme that reuses frequencies, where possible, is preferable. Conflict-free coloring problems have been the subject of many recent papers due to their practical and theoretical interest (see e.g. [17,9,6,7,1]).

Most approaches in the conflict-free coloring literature rely on the stronger unique-maximum colorings (a notable exception is the 'triples' algorithm in [1]), because unique-maximum colorings are easier to argue about in proofs, due to their additional structure. Another advantage of unique-maximum colorings

is the simplicity of computing the unique color in any range (it is always the maximum color), given a unique-maximum coloring, which can be helpful if very simple mobile devices are used by the agents.

Other hypergraphs that have been studied with respect to these colorings, are ones which are induced by a graph and (a) its neighborhoods or (b) its paths:

(a) Given a graph G, consider the hypergraph with the same vertex set as G and a hyperedge for every distinct vertex neighborhood of G; such conflict-free colorings have been studied in [4,16].
(b) Given a graph G, consider the hypergraph H with the same vertex set as G and a hyperedge for every distinct vertex set that can be spanned by a *simple* path of G. A unique-maximum (respectively conflict-free, odd) coloring of H is called a unique-maximum (respectively conflict-free, odd) coloring of G with respect to paths; we also define the corresponding graph chromatic numbers, $\chi^{\mathrm{p}}_{\mathrm{um}}(G) = \chi_{\mathrm{um}}(H)$, $\chi^{\mathrm{p}}_{\mathrm{cf}}(G) = \chi_{\mathrm{cf}}(H)$ and $\chi^{\mathrm{p}}_{\mathrm{odd}}(G) = \chi_{\mathrm{odd}}(H)$. Sometimes to improve readability of the text, we simply talk about the UM (respectively CF, ODD) chromatic number of a graph.

Unique-maximum colorings with respect to paths of graphs are known alternatively in the literature as *ordered colorings* or *vertex rankings*, and the unique-maximum chromatic number is also known as *tree-depth* [15]. The problem of computing such unique-maximum colorings is a well-known and widely studied problem (see e.g. [11]) with many applications including VLSI design [12] and parallel Cholesky factorization of matrices [13]. The problem is also interesting for the Operations Research community, because it has applications in planning efficient assembly of products in manufacturing systems [10]. In general, it seems that the vertex ranking problem can model situations where interrelated tasks have to be accomplished fast in parallel (assembly from parts, parallel query optimization in databases, etc.). For general graphs, finding the exact unique-maximum chromatic number with respect to paths of a graph is NP-complete [18,14,2,15] and there is a polynomial time $O(\log^2 n)$ approximation algorithm [3], where n is the number of vertices.

The paper [5] studied the relationship between the two graph chromatic numbers, $\chi^{\mathrm{p}}_{\mathrm{um}}(G)$ and $\chi^{\mathrm{p}}_{\mathrm{cf}}(G)$, showing that for every graph G, $\chi^{\mathrm{p}}_{\mathrm{um}}(G) \leq 2^{\chi^{\mathrm{p}}_{\mathrm{cf}}(G)} - 1$, and providing a sequence of graphs for which the ratio $\chi^{\mathrm{p}}_{\mathrm{um}}(G)/\chi^{\mathrm{p}}_{\mathrm{cf}}(G)$ tends to 2. Moreover, the authors of [5] proved that even checking whether a given coloring of a graph is conflict-free is coNP-complete (whereas the same problem for unique-maximum colorings is in P).

Our Results

In this work, we study the relationship between unique-maximum and conflict-free colorings. First, we give an exact answer to the question "How much larger than $\chi_{\mathrm{cf}}(H)$ can $\chi_{\mathrm{um}}(H)$ be?" for a general hypergraph H. In section 2, we show that if for a hypergraph H, $\chi_{\mathrm{cf}}(H) = k > 1$, then $\chi_{\mathrm{um}}(H)$ is bounded from above, roughly, by $\frac{k-1}{k}|V|$, and this is tight; the result remains true even if we

restrict ourselves to uniform hypergraphs. Then we turn to hypergraphs induced by paths in tree graphs and prove an upper bound for $\chi^{\mathrm{p}}_{\mathrm{um}}(T)$ that is polynomial in $\chi^{\mathrm{p}}_{\mathrm{cf}}(T)$, where T is a tree graph. We study trees because for general graphs the only known upper bound for $\chi^{\mathrm{p}}_{\mathrm{um}}(G)$ is exponential in $\chi^{\mathrm{p}}_{\mathrm{cf}}(G)$; see [5]. In section 3, we show that for every tree graph T, $\chi^{\mathrm{p}}_{\mathrm{um}}(T) \leq (\chi^{\mathrm{p}}_{\mathrm{cf}}(T))^3$ and provide a sequence of trees for which the ratio $\chi^{\mathrm{p}}_{\mathrm{um}}(T)/\chi^{\mathrm{p}}_{\mathrm{cf}}(T)$ tends to a constant c with $1 < c < 2$. Conclusions and open problems are presented in section 4.

1.1 Preliminaries

Observation 4. Each of the graph chromatic numbers $\chi^{\mathrm{p}}_{\mathrm{um}}$, $\chi^{\mathrm{p}}_{\mathrm{cf}}$, and $\chi^{\mathrm{p}}_{\mathrm{odd}}$, is monotone with respect to subgraphs, i.e., if $H \subseteq G$, then $\chi^{\mathrm{p}}_{\diamond}(H) \leq \chi^{\mathrm{p}}_{\diamond}(G)$, where $\diamond \in \{\mathrm{um}, \mathrm{cf}, \mathrm{odd}\}$.

Proof. A subgraph H of a graph G contains a subset of the paths of G. □

Definition 5 (Parity vector). Given a coloring $C\colon V \to \{1, \ldots, k\}$ and a set $e \subseteq V$, the *parity vector* of e is an element of $\{0,1\}^k$ in which the i^{th} coordinate equals the parity (0 or 1) of the number of elements in e colored with i.

Remark 1. A coloring of a hypergraph is odd if and only if the parity vector of every hyperedge is not the all-zero vector.

2 General Hypergraphs

In general it is not possible to bound χ_{cf} with a function of χ_{odd} because if we take our hyperedges to be all triples of $\{1, \ldots, n\}$, for the resulting hypergraph H we have $\chi_{\mathrm{odd}}(H) = 1$ and $\chi_{\mathrm{cf}}(H) = \lceil \frac{n}{2} \rceil$. Although $\chi_{\mathrm{cf}}(H) = 1$ implies $\chi_{\mathrm{um}}(H) = 1$, we can have a big gap as is shown by the following theorem.

Theorem 6. *For a hypergraph H on n vertices, $\chi_{\mathrm{um}}(H) \leq n - \lceil n/\chi_{\mathrm{cf}}(H) \rceil + 1$. Moreover, this is the best possible bound, i.e., for any positive integer n there exists a hypergraph on n vertices for which equality holds.*

Proof. A simple algorithm achieving the upper bound is the following. Given a hypergraph H with $\chi_{\mathrm{cf}}(H) = k$, take a conflict-free coloring of H with k colors, color the largest color class with color 1, all the other vertices with all different colors (bigger than 1). It is not difficult to see that this is a unique-max coloring, and it uses at most $n - \lceil n/k \rceil + 1$ colors.

For a given n and k equality holds for the hypergraph H whose n vertices are partitioned into k almost equal parts, all of size $\lceil n/k \rceil$ or $\lceil n/k \rceil - 1$ and its edges are all sets of size 2 and 3 covering vertices from exactly 2 parts.

We have $\chi_{\mathrm{cf}}(H) = k$ because in any conflict-free coloring of H there are no two vertices in different parts having the same color and $\chi_{\mathrm{um}}(H) \geq n - \lceil n/k \rceil + 1$ because in any unique-max coloring of H all vertices must have different colors except the vertices of one part. □

For uniform hypergraphs without small hyperedges, we can make the inequality tighter.

Theorem 7. *If $l \geq 3$ then for an arbitrary l-uniform hypergraph H with $\chi_{\mathrm{cf}}(H) = k$ having $n \geq 2kl$ vertices we have $\chi_{\mathrm{um}}(H) \leq n - \lceil n/k \rceil - l + 4$. Moreover, this is the best possible bound, i.e., for arbitrary $n \geq 2kl$ there exists a hypergraph for which equality holds.*

The proof is similar to the previous one, although longer, and is omitted in this version of our work, due to space considerations.

3 Tree Graphs

In this section, to ease readability we use UM for $\chi^{\mathrm{p}}_{\mathrm{um}}$, CF for $\chi^{\mathrm{p}}_{\mathrm{cf}}$ and ODD for $\chi^{\mathrm{p}}_{\mathrm{odd}}$. We denote by P_n the path graph with n vertices. As a warm-up we prove a simple claim about the odd chromatic number of the path graph. Our proof is a showcase of a parity vector argument, which we are going to also use later. For completeness, we include a computation of the conflict-free and unique-maximum chromatic numbers of the path graph [8]. (Throughout this paper we use base 2 logarithms, which are denoted by "log".)

Claim 8. *For $n \geq 1$, $ODD(P_n) = CF(P_n) = UM(P_n) = \lceil \log(n+1) \rceil$.*

Proof. It is easy to see that $UM(P_n) \leq \lceil \log(n+1) \rceil$: assign the biggest color only to a median vertex of the path and then use recursion. Since we know that $UM(P_n) \geq CF(P_n) \geq ODD(P_n)$, it is enough to prove that $2^{ODD(P_n)} > n$. Take the n paths starting from one endpoint. If there were two with the same parity vector, their symmetric difference (which is also a path) would contain an even number of each color. Thus we have at least n different parity vectors, none of which is the all-zero vector. But the number of non-zero parity vectors is at most $2^{ODD(P_n)} - 1$. □

3.1 Upper Bound for Unique-Maximum Number of Binary Trees

We denote by B_d the (rooted) complete binary tree with d levels (and $2^d - 1$ vertices). By convention, B_0 is the empty graph. It is easy to see that $UM(B_d) = d$; for an optimal unique-maximum coloring, color the leaves of B_d with color 1, their parents with color 2, and so on, until you color the root with color d; for a matching lower bound, use induction on d. In this section, we prove an upper bound for $UM(B_d)$ that is quadratic on $CF(B_d)$. In fact, we will prove a stronger statement, that is, a bound for $UM(B_d)$ that is quadratic on $ODD(B_d)$. Moreover, instead of proving a bound just for complete binary trees, we are going to prove a bound for subdivisions of complete binary trees, because we will need that later in subsection 3.2. We first need the following definitions.

Definition 9. A graph H is a *subdivision* of G if H is obtained by substituting every edge uv of G by a path of new internal vertices between u and v. The original vertices of G in H are called *branch vertices*.

Definition 10. Given is a rooted tree T and a rooted subtree T' of T. We say that T' is *compatible with* T if any two vertices of T' have the same ancestor-descendant relation in both T' and T.

We are now ready to state the following useful lemma.

Lemma 1. *Let B^* be a subdivision of B_d. Suppose we color (without any restrictions) the vertices of B^* with k colors. Then, there exists a vector $a = (a_1, a_2, \ldots, a_k)$ such that $\sum_{i=1}^{k} a_i \geq d$ and for every $i \in \{1, \ldots, k\}$, B^* contains a subdivision T^i of B_{a_i} such that (a) T^i is compatible with B^* and (b) the branch vertices of T^i are all colored with i.*

Proof. We construct the vector a by induction on d.

For $d = 1$, B^* has exactly one vertex, say with color i. Then, a is such that $a_i = 1$ and every other coordinate is 0.

For $d > 1$, consider the tree B^* rooted at the branch vertex v that corresponds to the root of B_d. Each of the left and right subtrees of v contains a subdivision of B_{d-1}. Thus, by the inductive hypothesis, we construct vector a' for the left subtree and a'' for the right subtree. If $a' \neq a''$, then a is such that $a_i = max(a'_i, a''_i)$, for $i \in \{1, \ldots, k\}$. If $a' = a''$, then a is such that $a_i = a'_i + 1$ for the color i of the root v and $a_j = a'_j$ for $j \neq i$. □

Theorem 11. *For $d \geq 1$ and for every subdivision B^* of B_d, $ODD(B^*) \geq \sqrt{d}$.*

Proof. Fix an optimal odd coloring with k colors. Fix a color $i \in \{1, \ldots, k\}$ for which in lemma 1 we have $a_i \geq d/k$.

Consider the $2^{a_i - 1}$ paths that originate in a leaf of the B_{a_i} subdivision and end in its root branch vertex. We claim that the parity vectors of the 2^{a_i-1} paths must be all different. Indeed, if there were two paths with the same parity vector, then the symmetric difference of the paths plus their lowest common vertex would form a path where the parity of each color is even, except maybe for color i, but since this new path starts and ends with color i, deleting any of its ends yields a path whose parity vector is the all-zero vector, a contradiction.

There are at most $2^k - 1$ parity vectors, thus $2^k - 1 \geq 2^{a_i - 1} \geq 2^{\lceil d/k \rceil - 1}$. From this we get $k > \lceil \frac{d}{k} \rceil - 1$ which is equivalent to $k \geq \lceil \frac{d}{k} \rceil$ using the integrality. Thus, $k \geq \sqrt{d}$. □

3.2 Upper Bound for Unique-Maximum Number of Arbitrary Trees

We will try to find either a long path or a subdivision of a deep complete binary tree in every tree with high UM chromatic number. For this, we need the notion of UM-critical trees and their characterization from [11].

Definition 12. A graph is *UM-critical*, if the UM chromatic number of any of its subgraphs is smaller than its UM chromatic number. We also say that a graph is *k-UM-critical*, if it is UM-critical and its UM chromatic number equals k.

Example 1. The complete graph K_k and the path with 2^{k-1} vertices are both k-UM-critical. For $k \leq 3$ there is a unique k-UM-critical tree, the path with 2^{k-1} vertices. Consider the following tree T on 8 vertices: Take two copies of P_4 and draw an edge from an internal vertex of one P_4 to an internal vertex of the other P_4. Tree T is 4-UM-critical and $CF(T) = 3$. (T is the smallest tree where the CF and UM chromatic numbers differ.)

Theorem 13 (Theorem 2.1 in [11]). *For $k > 1$, a tree is k-UM-critical if and only if it has an edge that connects two $(k-1)$-UM-critical trees.*

Remark 2. A k-UM-critical tree has exactly 2^{k-1} vertices and the connecting edge must always be the central edge of the tree. This implies that there is a unique way to partition the vertices of the k-UM-critical tree to two sets of vertices, each inducing a $(k-1)$-UM-critical tree, and so on.

Now we can define the *structure trees* of UM-critical trees.

Definition 14. For $l \in \{0, \ldots, k-1\}$, the *l-deep structure tree* of a k-UM-critical tree is the tree graph with a vertex for every one of the 2^l $(k-l)$-UM-critical subtrees that we obtain by repeatedly applying theorem 13, and an edge between two vertices if the corresponding $(k-l)$-UM-critical subtrees have an edge between them in the k-UM-critical tree.

Example 2. The 0-deep structure tree of any UM-critical tree is a vertex. The 1-deep structure tree of any UM-critical tree is an edge. The 2-deep structure tree of any UM-critical tree is a path with 4 vertices. The $(k-1)$-deep structure tree of a k-UM-critical tree is itself.

Remark 3. It is not difficult to prove that the l-deep structure tree of a UM-critical tree is an $(l+1)$-UM-critical tree.

We start with a few simple observations.

Observation 15. If an $(l+1)$-UM-critical tree has no vertex of degree at least 3, then it is the path with 2^l vertices.

Proof. Delete the central edge and use induction. □

Observation 16. If an $(l+2)$-UM-critical tree has only one vertex of degree at least 3, then it contains a path with 2^l vertices that ends in this vertex.

Proof. After deleting its central edge, one of the resulting $(l+1)$-UM-critical trees must be a path that was connected to the rest of the graph with one of its ends, thus we can extend it until the high degree vertex. □

Observation 17. If a tree contains two non-adjacent vertices with degree at least 3, then it contains a subdivision of B_3.

Proof. The non-adjacent degree 3 vertices will be the second level of the binary tree, and any vertex on the path connecting them the root. □

Claim 18. *An $(l+2)$-UM-critical tree contains a path with 2^l vertices or a subdivision of B_3.*

Proof. Because of the previous observations, we can suppose that our tree has exactly two vertices with degree at least 3 and these are adjacent. If the central edge is not the one between these vertices, then the graph must contain an $(l+1)$-UM-critical subgraph without any vertex with degree at least 3, thus it is the path with 2^l vertices because of observation 15. If it connects the two high degree vertices, then, using observation 16, we have two paths with 2^{l-1} vertices in the $(l+1)$-UM-critical subgraphs obtained by deleting the central edge ending in these vertices, thus with the central edge they form a path with 2^l vertices. □

We are now ready to prove our main lemma, before the proof of the upper bound.

Lemma 2. *For $k \geq 3$ and any l, every k-UM-critical tree contains a path with 2^l vertices or a subdivision of $B_{\lceil \frac{k+l+3}{l+2} \rceil}$.*

Proof. The proof is by induction on k. For $3 \leq k \leq l+1$, the statement is true since $B_2 = P_3$. For $l+2 \leq k \leq 2l+3$, the statement is equivalent to our claim 18. For $k > 2l+3$, take the $(l+2)$-deep structure tree S of the tree. If S does not contain a path with 2^l vertices, then, using claim 18, S contains a subdivision of B_3. Every one of the four leaf branch vertices of the above B_3 subdivision corresponds to a $(k-l-2)$-UM-critical subtree of the original tree. By induction, each one of the above four subtrees must contain a path with 2^l vertices or a subdivision of the complete binary tree with $\left\lceil \frac{k-l-2+l+3}{l+2} \right\rceil = \left\lceil \frac{k+l+3}{l+2} \right\rceil - 1$ levels. If any of them contains the path, we are done. If each one of them contains a $B_{\lceil \frac{k+l+3}{l+2} \rceil - 1}$ subdivision, then for every one of the four leaves, we can connect at least one of the two disjoint $B_{\lceil \frac{k+l+3}{l+2} \rceil - 2}$ subdivisions of the $B_{\lceil \frac{k+l+3}{l+2} \rceil - 1}$ subdivision in the leaf (as in figure 1, where each of the four relevant $B_{\frac{k+l+3}{l+2}-2}$ subdivisions and the paths connecting them are shown with heavier lines) to obtain a subdivision of a complete binary tree with $\left\lceil \frac{k+l+3}{l+2} \right\rceil - 2 + 2$ levels, thus we are done. □

Theorem 19. *For every tree T, $ODD(T) \geq (UM(T))^{\frac{1}{3}} - O(1)$.*

Proof. If $UM(T) = k$, then T contains a k-UM-critical tree, which (according to lemma 2) contains a P_{2^l} or a subdivision B^* of $B_{\lceil \frac{k+l+3}{l+2} \rceil}$. Using monotonicity of ODD with respect to subgraphs (observation 4), together with $ODD(P_{2^l}) = l+1$ (claim 8) and $ODD(B^*) \geq \sqrt{\frac{k+l+3}{l+2}}$ (from theorem 11), we get $ODD(T) \geq \max\left(l+1, \sqrt{\frac{k+l+3}{l+2}}\right)$. Choosing l to be the closest integer to the solution of $l+1 = \sqrt{\frac{k+l+3}{l+2}}$, we get $l = k^{\frac{1}{3}} + \Theta(1)$. Therefore, $ODD(T) \geq (UM(T))^{\frac{1}{3}} - O(1)$. □

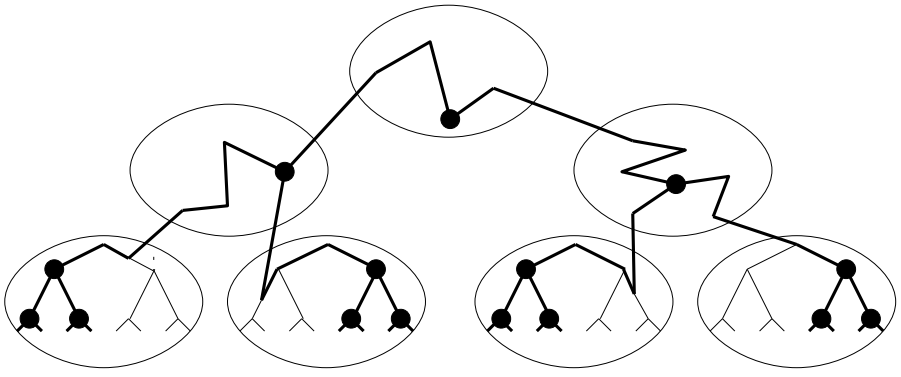


Fig. 1. Constructing a deep binary tree using induction for structure trees

3.3 Trees with Different Unique-Maximum and Conflict-Free Numbers

We have seen that $UM(B_d) = d$. We intend to show conflict-free colorings of some complete binary trees that use substantially less colors. We start with a simple example demonstrating our method.

Claim 20. $CF(B_7) \leq 6$.

Proof. Color the root with 1, the second level with 2. Deleting the colored vertices leaves four B_5 subtrees. In each of these subtrees, every level will be monochromatic. From top to bottom, in the first use the colors $3, 4, 5, 1, 2$, in the second $4, 5, 6, 1, 2$, in the third $5, 6, 3, 1, 2$ and in the forth $6, 3, 4, 1, 2$. It is not difficult to verify that this is indeed a conflict-free coloring (but it will also follow from later results). Observe that in the top 2 levels 2 colors are used, in the next 3 levels 4 colors, and in the last 2 levels the same 2 colors are used as the ones in the top level. □

Corollary 1. $CF(B_{2(r+1)+3r}) \leq 4r + 2$.

Proof. In the previous construction, every leaf had color 2 and their parents had color 1. Every such three vertex part can be the top of a new tree, similar to the original, and replacing $3, 4, 5, 6$ with four new colors. This gives a tree with 12 levels and 10 colors. It is not difficult to verify that this is indeed a conflict-free coloring (but it will also follow from later results). Repeatedly applying this procedure, so that we have colors 1, 2 appearing in $2(r + 1)$ levels and r disjoint sets of 4 colors each, we get a coloring of $B_{2(r+1)+3r}$ using $4r + 2$ colors. □

To examine more closely why these colorings are conflict-free, we need to define some notions.

Definition 21. A family $\mathcal{F}$ of ordered sets is said to be *prefix set-free*, if any prefix of any ordered set is different from any other ordered set as a set (without

the ordering). If the ground set has n elements, every sequence has length at least k and the cardinality of $\mathcal{F}$ is at least 2^d, then we say that $\mathcal{F}$ is a $[k, d, n]$ *PSF family*.

Example 3. $\{\langle 1,3\rangle, \langle 1,2,3\rangle\}$ is a $[2,1,3]$ PSF family and $\{\langle 1\rangle, \langle 2,1\rangle, \langle 2,3\rangle, \langle 3,1\rangle, \langle 3,1,2\rangle\}$ is a $[1,2,3]$ PSF family but $\{\langle 2,1\rangle, \langle 1,2,3\rangle\}$ is not a PSF family.

Claim 22. *For any $[k, d, n]$ PSF family $d \leq \log \sum_{i=k}^{n} \binom{n}{i}$.*

Proof. Any two ordered sets must differ as sets. □

Claim 23. *There is a $[k, d, n]$ PSF family with $d = \lfloor \log \binom{n}{k} \rfloor$.*

Proof. Take all k element subsets of $\{1, \ldots, n\}$ and order each arbitrarily. □

Since these bounds do not differ much if $k > (\frac{1}{2} + \epsilon)n$, we do not attempt to get sharper bounds.

Theorem 24. *If there is a $[k, d, n]$ PSF family where the size of every set is at most $k + d$, then $CF(B_{d(r+1)+kr}) \leq nr + d$.*

Proof. First, we show that $CF(B_{k+2d}) \leq n + d$. Color the top d levels with d colors. Remove the colored vertices and consider the 2^d B_{k+d} subtrees left. To each associate an ordered set from the $[k, d, n]$ PSF family and color the whole i^{th} level with one color, the i^{th} element of the associated ordered set. Deleting also these colored vertices, we are left with subtrees with at most d levels, which we can color with (at most) the same d colors we used for the top levels. It is not difficult to check that the above procedure produces a conflict-free coloring. By repeating the above procedure r times for $B_{d(r+1)+kr}$, as in corollary 1, we obtain $CF(B_{d(r+1)+kr}) \leq nr + d$. □

Corollary 2. *For the sequence of complete binary trees, $\{B_i\}_{i=1}^{\infty}$, the limit of the ratio of the UM to the CF chromatic number is at least* $\log 3 \approx 1.58$.

Proof. We omit the technical proof of the existence of the limit.

Since $CF(B_{d(r+1)+kr}) \leq nr + d$, the ratio of UM to CF for $B_{d(r+1)+kr}$ is at least $(d(r+1) + kr)/(nr + d)$, which tends to $(d + k)/n$ as $r \to \infty$. From claim 23 we can choose $d = \lfloor \log \binom{n}{k} \rfloor$. If we substitute k with xn, then a short calculation shows that to maximize $(d + k)/n$ we have to maximize $x + H(x)$, where $H(x) = -x \log x - (1 - x) \log(1 - x)$ (entropy). The function $x + H(x)$ attains its maximum at $x = 2/3$, giving a value of $\log 3$ as a lower bound for the limit. □

4 Discussion and Open Problems

In the literature of conflict-free coloring, hypergraphs that are induced by geometric shapes have been in the focus. It would be interesting to show possible relations between unique-maximum and conflict-free chromatic numbers in this setting.

The exact relationship between the two chromatic numbers with respect to paths for general graphs still remains an open problem. In [5], only graphs which have unique-maximum chromatic number about twice the conflict-free chromatic number were exhibited, but the only bound proved on $\chi^{\mathrm{p}}_{\mathrm{um}}(G)$ was exponential in $\chi^{\mathrm{p}}_{\mathrm{cf}}(G)$. In fact it is even possible that $\chi^{\mathrm{p}}_{\mathrm{um}}(G) \leq 2\chi^{\mathrm{p}}_{\mathrm{cf}}(G) - 2$. The first step to prove this would be to show that $\chi^{\mathrm{p}}_{\mathrm{um}}(T) = O(\chi^{\mathrm{p}}_{\mathrm{cf}}(T))$ for trees. It would also be interesting to extend our results to other classes of graphs.

Acknowledgments. We would like to thank Géza Tóth for fruitful discussions and ideas about improving the lower bound in corollary 2.

References

1. Bar-Noy, A., Cheilaris, P., Smorodinsky, S.: Deterministic conflict-free coloring for intervals: from offline to online. ACM Transactions on Algorithms 4(4), 44.1–44.18 (2008)
2. Bodlaender, H.L., Deogun, J.S., Jansen, K., Kloks, T., Kratsch, D., Müller, H., Tuza, Z.: Rankings of graphs. SIAM Journal on Discrete Mathematics 11(1), 168–181 (1998)
3. Bodlaender, H.L., Gilbert, J.R., Hafsteinsson, H., Kloks, T.: Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. Journal of Algorithms 18(2), 238–255 (1995)
4. Cheilaris, P.: Conflict-free coloring. Ph.D. thesis, City University of New York (2009)
5. Cheilaris, P., Tóth, G.: Graph unique-maximum and conflict-free colorings. Journal of Discrete Algorithms 9, 241–251 (2011)
6. Chen, K., Fiat, A., Kaplan, H., Levy, M., Matoušek, J., Mossel, E., Pach, J., Sharir, M., Smorodinsky, S., Wagner, U., Welzl, E.: Online conflict-free coloring for intervals. SIAM Journal on Computing 36(5), 1342–1359 (2007)
7. Elbassioni, K.M., Mustafa, N.H.: Conflict-Free Colorings of Rectangles Ranges. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 254–263. Springer, Heidelberg (2006)
8. Even, G., Lotker, Z., Ron, D., Smorodinsky, S.: Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks. SIAM Journal on Computing 33, 94–136 (2003)
9. Har-Peled, S., Smorodinsky, S.: Conflict-free coloring of points and simple regions in the plane. Discrete and Computational Geometry 34, 47–70 (2005)
10. Iyer, A.V., Ratliff, H.R., Vijayan, G.: Optimal node ranking of trees. Information Processing Letters 28, 225–229 (1988)
11. Katchalski, M., McCuaig, W., Seager, S.: Ordered colourings. Discrete Mathematics 142, 141–154 (1995)
12. Leiserson, C.E.: Area-efficient graph layouts (for VLSI). In: Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 270–281 (1980)
13. Liu, J.W.: The role of elimination trees in sparse factorization. SIAM Journal on Matrix Analysis and Applications 11(1), 134–172 (1990)
14. Llewellyn, D.C., Tovey, C.A., Trick, M.A.: Local optimization on graphs. Discrete Applied Mathematics 23(2), 157–178 (1989)

15. Nešetřil, J., de Mendez, P.O.: Tree-depth, subgraph coloring and homomorphism bounds. European Journal of Combinatorics 27, 1022–1041 (2006)
16. Pach, J., Tardos, G.: Conflict-free colourings of graphs and hypergraphs. Combinatorics, Probability and Computing 18, 819–834 (2009)
17. Pach, J., Tóth, G.: Conflict free colorings. In: Discrete and Computational Geometry. The Goodman-Pollack Festschrift, pp. 665–671. Springer, Heidelberg (2003)
18. Pothen, A.: The complexity of optimal elimination trees. Tech. Rep. CS-88-16, Department of Computer Science, Pennsylvania State University (1988)
19. Smorodinsky, S.: Combinatorial Problems in Computational Geometry. Ph.D. thesis, School of Computer Science, Tel-Aviv University (2003)

Minimal Dominating Sets in Graph Classes: Combinatorial Bounds and Enumeration*

Jean-François Couturier[1], Pinar Heggernes[2], Pim van't Hof[2], and Dieter Kratsch[1]

[1] LITA, Université Paul Verlaine - Metz, 57045 Metz Cedex 01, France
{couturier,kratsch}@univ-metz.fr
[2] Department of Informatics, University of Bergen, N-5020 Bergen, Norway
{pinar.heggernes,pim.vanthof}@ii.uib.no

Abstract. The maximum number of minimal dominating sets that a graph on n vertices can have is known to be at most 1.7159^n. This upper bound might not be tight, since no examples of graphs with 1.5705^n or more minimal dominating sets are known. For several classes of graphs, we substantially improve the upper bound on the maximum number of minimal dominating sets in graphs on n vertices. In some cases, we provide examples of graphs whose number of minimal dominating sets exactly matches the proved upper bound for that class, thereby showing that these bounds are tight. For all considered graph classes, the upper bound proofs are constructive and can easily be transformed into algorithms enumerating all minimal dominating sets of the input graph.

1 Introduction

Combinatorial questions of the type *"What is the maximum number of vertex subsets satisfying a given property in a graph?"* have found interest and applications in computer science, especially in exact exponential algorithms [5]. The question has been studied recently for minimal feedback vertex sets, minimal separators, potential maximal cliques, and for minimal feedback vertex sets in tournaments [3,6,7]. A famous classical example is the highly cited theorem of Moon and Moser [14], which states that the maximum number of maximal cliques and maximal independent sets, respectively, in any graph on n vertices is $3^{n/3}$. Although the original proof of the upper bound in [14] is by induction, it is not hard to transform it into a branching algorithm enumerating all maximal independent sets of a graph in time $O^*(3^{n/3})$, where the O^*-notation suppresses polynomial factors. These results were used by Lawler [13] to give an algorithm for graph coloring, which was the fastest algorithm for this purpose for over two decades. A faster algorithm for graph coloring was obtained by Eppstein [2] by improving the upper bound on the maximum number of maximal independent sets of small size.

* This work has been supported by the Research Council of Norway (SCOPE 197548/F20) and ANR Blanc AGAPE (ANR-09-BLAN-0159-03).

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 202–213, 2012.

The number of papers on domination in graphs is in the thousands, and several well known surveys and books are dedicated to the topic (see, e.g., [9]). It is surprising that a first non-trivial answer to the Moon and Moser type question *"What is the maximum number of minimal dominating sets in a graph?"* was established only recently. Fomin, Grandoni, Pyatkin and Stepanov [4] showed that the maximum number of minimal dominating sets in a graph on n vertices is at most 1.7159^n. This result was used to derive an $O(2.8718^n)$ algorithm for the Domatic Number problem [4]. Although examples of graphs with 1.5704^n minimal dominating sets have been identified [4] (see Fig. 1), it is not known whether graphs with 1.5705^n or more minimal dominating sets exist.

Our interest in this combinatorial question was triggered by the large gap between the best known lower and upper bound for general graphs and the exact exponential algorithms background of the problem. We provide upper and lower bounds for the maximum number of minimal dominating sets in a variety of graph classes. Our upper bounds heavily rely on structural graph properties. Typically, either we have tight bounds, i.e., matching upper and lower bounds, that are proved using combinatorial arguments, or we have asymptotic bounds that are proved using branching algorithms. Our findings are summarized in Table 1.

Graph Class	Lower Bound	Upper Bound
general [4]	1.5704^n	1.7159^n
chordal	1.4422^n	1.6181^n
cobipartite	1.3195^n	1.5875^n
split	1.4422^n	1.4656^n
proper interval	1.4422^n	1.4656^n
cograph*	1.5704^n	1.5705^n
trivially perfect*	1.4422^n	1.4423^n
threshold*	$\omega(G)$	$\omega(G)$
chain*	$\lfloor n/2 \rfloor + m$	$\lfloor n/2 \rfloor + m$

Table 1. Lower and upper bounds on the maximum number of minimal dominating sets. The bounds for graph classes marked with an asterisk are tight; differences in the last digit are caused by rounding.

Very recently Kanté et al. [11] showed that all minimal dominating sets in a split graph G can be enumerated in time polynomial in the number of minimal dominating sets of G; however they did not study the number of such sets. As an important byproduct of our results, we obtain algorithms to enumerate all minimal dominating sets for graphs in each of the studied graph classes. In fact, all our upper bound proofs are constructive and the enumeration algorithms are easy consequences of them: simply check for all the generated candidate sets whether they are indeed minimal dominating sets. For each graph class with an exponential upper bound, say $O(c^n)$, the running time of the corresponding enumeration algorithm is $O^*(c^n)$. The enumeration algorithms for threshold graphs and chain graphs have polynomial running times. We believe that our

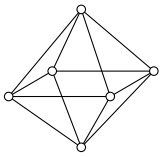


Fig. 1. The graph G^*, which has 6 vertices and 15 minimal dominating sets. The graph G_n^* on $n = 6k$ vertices, consisting of k disjoint copies of G^*, has $15^{n/6} \approx 1.5704^n$ mds.

enumeration algorithms might have non-trivial algorithmic applications in domination type problems like Domatic Number.

Our paper is organized as follows. In Sections 3 and 4, we use branching algorithms to establish upper bounds on the number of minimal dominating sets in chordal graphs and split graphs, respectively. In Section 5, a combinatorial argument is applied to establish an upper bound for cobipartite graphs. In Sections 6 and 7, we determine *tight* upper bounds for cographs and chain graphs. The bounds for the remaining three graph classes are proved by similar techniques; the details have been omitted due to page restrictions.

2 Preliminaries

We work with simple undirected graphs. We denote such a graph by $G = (V, E)$, where V is the set of vertices and E is the set of edges of G. We adhere to the convention that $n = |V|$ and $m = |E|$. When the vertex set and the edge set of G are not specified, we use $V(G)$ and $E(G)$ to denote these, respectively. The set of *neighbors* of a vertex $v \in V$ is the set of vertices adjacent to v, and is denoted by $N_G(v)$. The *closed neighborhood* of v is $N_G[v] = N(v) \cup \{v\}$. For a set $S \subseteq V$, we define analogously $N_G(S) = \bigcup_{v \in S} N_G(v) \setminus S$ and $N_G[S] = N_G(S) \cup S$. We will omit the subscript G when there is no ambiguity. A vertex v is *universal* if $N[v] = V$ and *isolated* if $N(v) = \emptyset$. The subgraph of G *induced* by S is denoted by $G[S]$. For ease of notation, we use $G-v$ to denote the graph $G[V \setminus \{v\}]$, and $G-S$ to denote the graph $G[V \setminus S]$. A graph is *connected* if there is a path between every pair of its vertices. A maximal connected subgraph of G is called a *connected component* of G. A set $S \subseteq V$ is called an *independent set* if $uv \notin E$ for every pair of vertices $u, v \in S$, and S is called a *clique* if $uv \in E$ for every pair of vertices $u, v \in S$. An independent set or a clique is *maximal* if no proper superset of it is an independent set or a clique, respectively.

A vertex set $S \subseteq V$ is a *dominating set* of G if $N[S] = V$. Every vertex v of a dominating set *dominates* the vertices in $N[v]$. A dominating set S is a *minimal dominating set* (*mds*) if no proper subset of S is a dominating set. It is an easy observation that, if S is a mds, then for every vertex $v \in S$, there is a vertex $x \in N[v]$ which is dominated only by v. We will call such a vertex x a *private neighbor* of v, since x is not adjacent to any vertex in $S \setminus \{v\}$. Note that a vertex in S might be its own private neighbor. The number of mds in a graph

G is denoted by $\mu(G)$. The following observation follows from the fact that every mds of G is the union of a mds of each connected component of G.

Observation 1. *Let G be a graph with connected components $G_1, G_2, \ldots, G_t$. Then $\mu(G) = \prod_{i=1}^{t} \mu(G_i)$.*

Each of the graph classes that we study will be defined in the section containing the results on that class. All of the graph classes mentioned in this paper can be recognized in linear time, and are closed under taking induced subgraphs [1,8]. We now define two graph families that are useful for providing examples of lower bounds on the maximum number of mds. We write H_n to denote the graph on $n = 3k$ vertices which is the disjoint union of k triangles. We write S_n to denote the graph on $n = 3k$ vertices which consists of a clique C of size $2k$ and an independent set I of size k, such that each vertex of I has exactly two neighbors in C, and no two vertices in I have a common neighbor. It can be verified easily that $\mu(H_n) = \mu(S_n) = 3^{n/3} \approx 1.4422^n$. For several of our graph classes, the graph families H_n and S_n provide the best known lower bound on the maximum number of mds.

2.1 Preliminaries on Branching

To prove the results given in the next two sections, we use branching algorithms to generate a collection of vertex subsets at the leaves of the corresponding search tree, containing *all* mds and possibly also subsets that are not mds. Consequently, the number of leaves of the search tree gives an upper bound on $\mu(G)$. By simply checking whether each generated vertex subset is indeed a mds, one can also obtain an enumeration algorithm for all mds of a graph G belonging to the studied graph class. Typically, every recursive call has input (G', D), where G' is an induced subgraph of the input graph G, and D is a subset of $V(G) \setminus V(G')$. The subset D contains vertices outside of G' that have been chosen to be in a possible minimal dominating set of G. Initially, no vertex has been chosen for a dominating set, so the algorithm starts with the call $(G, \emptyset)$. At every step, we make choices that result in new subproblems in which the size of G' decreases and the size of D possibly increases. For every such set D, either there is a leaf of the search tree corresponding to a mds of G that contains D as a subset, or no mds of G contains D as a subset. Our algorithms always proceed in such a way that a vertex of G' is never needed to dominate a vertex outside of G'. As a consequence, no vertex of G' has a private neighbor outside of G'.

For the analysis of the number of leaves $T(n)$ in the search tree, we use standard terminology [5]. In particular, if at each step of the branching we make t new subproblems, where the size of the instance is decreased by $c_1, c_2, \ldots, c_t$ in each subproblem, respectively, we obtain a recurrence $T(n) \leq T(n - c_1) + T(n - c_2) + \ldots + T(n - c_t)$. Such a recurrence is said to have *branching vector* $(c_1, c_2, \ldots, c_t)$. The number of leaves in the search tree is upper bounded by $O^*(\alpha^n)$, where α is the largest real root of $x^n - x^{n-c_1} - \ldots - x^{n-c_t} = 0$ [5]. The number α is called the *branching number* of this branching vector. It is common

to round α to the fourth digit after the decimal point. By rounding the last digit up, we can use O notation instead of O^* notation [5]. If different branching vectors are involved at different steps of an algorithm, then the branching vector with the highest branching number gives an upper bound on the number of leaves. In our results, we will not do the calculations of α explicitly, but just say, e.g., that branching vector $(2, 2)$ has branching number 1.4143, which implies that the number of leaves in the search tree is bounded by $O(1.4143^n)$ in a branching algorithm where only branching vector $(2, 2)$ occurs.

3 Chordal Graphs

A *chord* of a cycle is an edge between two non consecutive vertices of the cycle. A graph is *chordal* if every cycle of length at least 4 has a chord. A vertex v is called *simplicial* if $N(v)$ is a clique. Every chordal graph has a simplicial vertex [8]. Observe that H_n and S_n are chordal, giving us examples of chordal graphs with $3^{n/3} \approx 1.4422^n$ mds.

Theorem 1. *A chordal graph has at most* $O(1.6181^n)$ *minimal dominating sets.*

Proof. Given an instance (G', D), we say that a vertex v of G' is *already dominated* if D contains a vertex of $N_G(v)$. Our branching algorithm picks a simplicial vertex x of G'. If x is isolated: if x is already dominated, then we do not add x to D, otherwise we add x to D. No branching is involved; we delete x from G' and continue with another simplicial vertex of G'. From now on, we assume that x has at least one neighbor in G'. We take action depending on whether or not x is already dominated, and on the number of neighbors x has in G'. Note that only one of the cases below applies, and will be executed by the algorithm.

Case 1: x is Already Dominated. We branch on the choice of either adding x to D or discarding x from inclusion in a possible mds containing D.

- *Add:* $x \in D$. Since x is already dominated, it needs a private neighbor in $N_{G'}(x)$. Because x is simplicial, $N_{G'}(x)$ is a clique and this means that no vertex of $N_{G'}(x)$ can appear in a mds containing D as a subset. Consequently, we can safely delete $N_{G'}[x]$, which results in the instance $(G' - N_{G'}[x], D \cup \{x\})$, and gives a decrease of at least 2 vertices.
- *Discard:* $x \notin D$. Since x is already dominated, it is safe to simply delete x from G'. This results in the instance $(G' - x, D)$, and gives a decrease of 1 vertex.

Case 2: x is not already dominated and has at least 2 *neighbors in G'.* Let y be any neighbor of x in G'. We branch on the choice of either adding y to D or discarding y with respect to D.

- *Add:* $y \in D$. When y is added to D, it dominates x. Then x will never be part of a mds containing D, since it would need a private neighbor, which does

not exist since $N_{G'}(x) \subseteq N_{G'}(y)$ and every vertex in $G - V(G')$ is already dominated by D. We can therefore safely delete both x and y, which results in the instance $(G'-\{x,y\}, D \cup \{y\})$, and gives a decrease of 2.
- *Discard:* $y \notin D$. In this case, we simply delete y from G', which is safe for the following reason. Vertex x is not deleted and still needs to be dominated, and every neighbor of x in G' is also a neighbor of y in G', since x is simplicial. Hence, when x becomes dominated, then so will y. This might also happen by x being added to D at a later step. Hence we create a new instance $(G'-y, D)$, which gives a decrease of 1.

Case 3: x is not already dominated and has exactly one neighbor y in G'. Since x is not already dominated and is only adjacent to y in G', either x or y needs to be added to D to ensure that x is dominated. We branch on these possibilities.

- $x \in D$. In this case, y becomes dominated, and no mds containing D as a subset can contain y, since then x would not have a private neighbor. Consequently, we can safely delete x and y in this case. We get the instance $(G'-\{x,y\}, D \cup \{x\})$, and a decrease of 2.
- $y \in D$. Now x becomes dominated, and it can never become a member of a mds containing D, as x would then not have a private neighbor. Again, we can safely delete x and y, yielding the instance $(G'-\{x,y\}, D \cup \{y\})$, and a decrease of 2.

The branching vectors obtained in Cases 1, 2, and 3 are $(2,1)$, $(2,1)$, and $(2,2)$, respectively. Branching vector $(2,1)$ has the largest branching number, namely 1.6181, resulting in an upper bound of $O(1.6181^n)$. □

For split graphs, that form a subset of chordal graphs, we are able to give a better upper bound in the next section.

4 Split Graphs

A graph $G = (V, E)$ is a *split graph* if V can be partitioned into a clique C and an independent set I, where (C, I) is called a *split partition* of G. A split partition can be computed in linear time [8], and is not necessarily unique. In particular, if a vertex $c \in C$ has no neighbors in I, then $(C \setminus \{c\}, I \cup \{c\})$ is also a split partition of G. In the remainder of this section, we will assume that I is maximal, i.e., every vertex of C has a neighbor in I. Note that a mds of G cannot contain a vertex $u \in I$ together with a neighbor of u. Since S_n is a split graph, there are split graphs with $3^{n/3} \approx 1.4422^n$ mds.

Theorem 2. *A split graph has at most $O(1.4656^n)$ minimal dominating sets.*

Proof. As in the previous section, we use branching, but this time we use a two-phase branching algorithm. Our initial input is a split graph G and a split partition (C, I) of G, where I is maximal. At each step of the algorithm, the

subproblem at hand is described by (G', C', I', D), where G' is an induced subgraph of G and (C', I') is a split partition of G' such that $C' \subseteq C$ and $I' \subseteq I$. Our algorithm proceeds in such a way that no vertex of I' is already dominated by a vertex in D. Consequently, if a vertex of I' has no neighbor in C', then it is added to D; this rule requires no branching. If no such isolated vertex exists in I', the algorithm chooses a vertex $c \in C'$ such that c has a maximum number of neighbors in I', as long as this maximum number is at least 2. Then it branches and recursively solves two subproblems: either it selects c to be added to D and removes c and all its neighbors in I' from the current graph, or it discards c from being added to D and removes c from the graph. Hence, the decrease in the size of the graph is at least 3 in the first subproblem, and 1 in the second subproblem. This implies the branching vector $(3, 1)$, and its branching number is 1.4656. This completes the description of the first phase of the algorithm.

Let us consider a leaf of the corresponding search tree, and the instance (G', C', I', D) at this leaf. Contrary to the analysis in the previous section, a leaf of the search tree may lead to more than one mds, since we stopped branching when the vertices in C' each have at most one neighbor in I', and thus $V(G')$ might be non-empty. We claim that there are at most $3^{t/3}$ mds of G that can be obtained from this instance, where t is the number of vertices of G'. To show this, we now describe the second phase of the algorithm. If there is a vertex c in C' that does not have any neighbor in I', then we know that c is already dominated. This is because every vertex of C originally had neighbors in I, and since the neighbors in I of c were deleted, while c itself was not deleted, a neighbor $x \in C$ of c must already have been placed in D, by the description of the algorithm. Hence c is dominated by a vertex in $C \cap D$ and c has no private neighbor, which implies that c cannot be added to D. Consequently, as a first step, we remove all vertices of C' that do not have any neighbor in I'. Then, we add all isolated vertices of I' to D and remove them from G'. When no such vertices are present, we choose a vertex u of I' and branch into the following $|N_{G'}(u)| + 1$ subproblems: for every neighbor c of u, add c to D and remove u and all its neighbors from G', and, in the last subproblem, add u to D and remove u and all its neighbors. Since every vertex in C' has exactly one neighbor in I', exactly one vertex of $N_{G'}[u]$ belongs to a mds of G containing D, showing the correctness of the branching. The subproblems are solved recursively.

Assuming the vertex u has j neighbors, there are $j + 1$ subproblems, and for each one the decrease is $j + 1$. Simple analysis leads to the recurrences $T(t) = (j + 1) \cdot T(t - j - 1)$ for $j \geq 1$ for the number of leaves in the search tree. This is a well-known recurrence and its solution is $T(t) = 3^{t/3}$ (see e.g. [5]). Thus the number of leaves of the search tree for the second branching algorithm is at most $3^{t/3}$, and now each leaf contains at most one mds of G.

To establish an upper bound on $\mu(G)$, notice that the number of leaves of the first branching algorithm containing an instance with t vertices is at most $O(1.4656^{n-t})$, since those leaves correspond to a total decrease of $n - t$ (from G to the graph G' of the leaf) of the measure. Consequently, the number of leaves of the search tree of the first branching is at most $\sum_{t=0}^{n} O(1.4656^{n-t})$. Since

each of those leaves leads to at most $3^{t/3} \approx 1.4423^t$ mds of G, we conclude that the maximum number of mds in a split graph G is at most $\sum_{t=0}^{n} O(1.4656^{n-t} \cdot 1.4423^t) \leq \sum_{t=0}^{n} O(1.4656^n) = O(1.4656^n)$. □

5 Cobipartite Graphs

A graph $G = (V, E)$ is *cobipartite* if V can be partitioned into two cliques. To obtain a lower bound, we define a graph family B_n as follows. For $n = 5k$, start with two disjoint cliques X and Y, where $|X| = k$ and $|Y| = 4k$. Make every vertex in X adjacent to exactly four vertices in Y, such that every vertex in Y is adjacent to exactly one vertex in X. The graph B_n has $4^{n/5} \approx 1.3195^n$ mds that are subsets of Y, and $O(n^2)$ minimal dominating sets of the form $\{x, y\}$ with $x \in X$ and $y \in Y$.

Theorem 3. *A cobipartite graph has at most $O(1.5875^n)$ minimal dominating sets.*

Proof. Let $G = (V, E)$ be a cobipartite graph on n vertices, and let (X, Y) be a partition of V such that X and Y are cliques. Assume, without loss of generality, that $|X| = \alpha n$ with $0.5 \leq \alpha \leq 1$, and $|Y| = (1 - \alpha)n$. Let D be a mds of G. If $|D| = 1$, then $D = \{v\}$ for some universal vertex of G. Hence G has at most n mds of size 1. If $|D| \geq 2$ and $D \cap X \neq \emptyset$ and $D \cap Y \neq \emptyset$, then we must have $D = \{x, y\}$ for some vertices $x \in X$ and $y \in Y$, since every vertex in D needs a private neighbor. Hence there are at most $n^2/4$ mds of this type. Now assume that $|D| \geq 2$, and that we either have $D \subseteq X$ or $D \subseteq Y$. Clearly there are at most $2^{|Y|} \leq 2^{n/2}$ mds D of G with $D \subseteq Y$. It remains to find an upper bound on the number of mds D of G satisfying $D \subseteq X$. Let $|D| = \beta n$, where $\beta \leq \alpha$ and $2 \leq \beta n \leq |X|$. Every vertex of D must have a private neighbor; this can only be a vertex of Y, since D contains at least two vertices of X. This implies that $\beta \leq 1 - \alpha$. The number of subsets of X of size βn is $\binom{\alpha n}{\beta n}$. For every fixed α, the value of $\binom{\alpha n}{\beta n}$ is maximized for $\beta = \alpha/2$. To maximize the value of $\binom{\alpha n}{\beta n}$, note that $\beta = \alpha/2 \leq 1 - \alpha$ implies $\alpha \leq 2/3$. Hence the number of mds D, with $|D| \geq 2$ and $D \subseteq X$, is at most $\binom{2n/3}{n/3}$, which is less than or equal to $2^{2n/3}$, i.e., the number of all subsets of a set of size $2n/3$. In total, there are at most $n + n^2/4 + 2^{n/2} + 2^{2n/3} = O(2^{2n/3}) = O(1.5875^n)$ mds in G. □

We now move on to graph classes with tight upper bounds.

6 Cographs

Cographs are of particular interest in the study of the maximum number of mds, as the only known examples of graphs with 1.5704^n mds are cographs. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs. The *disjoint union* of G_1 and G_2 is the graph $G_1 \uplus G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. The *join* of G_1 and G_2 is the graph $G_1 \bowtie G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup \{v_1 v_2 \mid v_1 \in V_1, v_2 \in V_2\})$. A graph G is a

cograph if it can be constructed from isolated vertices by disjoint union and join operations. The graph G^* depicted in Fig. 1, is a cograph. It has 6 vertices and 15 mds. Any graph G_n^* on $n = 6k$ vertices consisting of k disjoint copies of G^* is a cograph containing $15^{n/6} \approx 1.5704^n$ mds. No example of a graph with 1.5705^n or more minimal dominating sets is known.

Theorem 4. *A cograph has at most $15^{n/6}$ minimal dominating sets, and there are cographs with $15^{n/6}$ minimal dominating sets.*

Proof. G_n^*, defined before the theorem, is a cograph with $15^{n/6}$ mds, so it remains to prove the upper bound. It can be verified exhaustively that the theorem holds for all cographs on at most 6 vertices. Let G be a cograph on $n \geq 7$ vertices. We prove the theorem by induction on the number of vertices. By the definition of a cograph, there exist two subgraphs G_1 and G_2 of G such that $G = G \uplus G_2$ or $G = G_1 \bowtie G_2$. Let $n_i = |V(G_i)|$ for $i = 1, 2$. If $G = G_1 \uplus G_2$, then by Observation 1, we have $\mu(G) = \mu(G_1) \cdot \mu(G_2) \leq 15^{n_1/6} \cdot 15^{n_2/6} = 15^{n/6}$. Suppose that $G = G_1 \bowtie G_2$. Then any mds of G_1 dominates every vertex in G_2, and vice versa. This means that any mds of G_1 is a mds of G, and the same holds for any mds of G_2. Since G is the complete join of G_1 and G_2, no mds of G contains more than one vertex from G_1 and more than one vertex from G_2. This means that any mds of G that is not a mds of G_1 or G_2 is of the form $\{v_1, v_2\}$, where $v_1 \in V(G_1)$ and $v_2 \in V(G_2)$. Hence $\mu(G) = \mu(G_1)+\mu(G_2)+n_1n_2 \leq 15^{n_1/6}+15^{n_2/6}+n_1n_2 = 15^{n_1/6}+15^{(n-n_1)/6}+n_1(n-n_1)$. Since we assumed that $1 \leq n_1 \leq n-1$ and $n \geq 7$, the function $15^{n_1/6} + 15^{(n-n_1)/6} + n_1(n - n_1)$ is maximal when $n_1 \in \{1, n-1\}$. In both cases, we get $\mu(G) \leq 15^{(n-1)/6} + 15^{1/6} + n - 1$, which is less than $15^{n/6}$ for any $n \geq 7$. □

7 Chain Graphs

In this section, we combine a study of structural properties of mds in chain graphs with combinatorial arguments to exactly determine the maximum number of mds in a chain graph on n vertices. A bipartite graph $G = (A, B, E)$ is a *chain graph* if there is an ordering $\sigma_A = \langle a_1, a_2, \ldots, a_k \rangle$ of the vertices of A such that $N(a_1) \subseteq N(a_2) \subseteq \cdots \subseteq N(a_k)$, as well as an ordering $\sigma_B = \langle b_1, b_2, \ldots, b_\ell \rangle$ of the vertices of B such that $N(b_1) \supseteq N(b_2) \supseteq \cdots \supseteq N(b_\ell)$ [15]. The orderings σ_A and σ_B together form a *chain ordering* of G. Note that if a chain graph G is disconnected, then at most one connected component of G contains edges.

In any graph, every maximal independent set is a minimal dominating set, and we start with the following result on chain graphs, which will help us prove the bound on mds, and which is also interesting on its own.

Lemma 1. *A chain graph has at most $\lfloor n/2 \rfloor + 1$ maximal independent sets, and there are chain graphs that have $\lfloor n/2 \rfloor + 1$ maximal independent sets.*

Proof. Let $G = (A, B, E)$ be a chain graph on n vertices and assume, without loss of generality, that $|A| \leq \lfloor n/2 \rfloor$. Since any isolated vertex must belong to

every maximal independent set, we may assume that G is connected. Let $\sigma_A = \langle a_1, a_2, \ldots, a_k \rangle$ and $\sigma_B = \langle b_1, b_2, \ldots, b_\ell \rangle$ be a chain ordering of G. Observe that b_1 dominates A and a_k dominates B. Let $\nu_i(G)$ be the number of maximal independent sets in G containing a_i, but not containing any vertex a_j with $j > i$. Consider $\nu_i(G)$ for some $i \in \{1, \ldots, k\}$. If every maximal independent set in G contains a vertex a_j with $j > i$, then $\nu_i(G) = 0$. Suppose there exists a maximal independent set S containing a_i, but not containing any vertex a_j with $j > i$. Clearly, none of the neighbors of a_i can be in S. Since σ_A and σ_B form a chain ordering of G, $N(a_p) \subseteq N(a_i)$ for every $p < i$. This means in particular that there is no edge between any vertex in $\{a_1, \ldots, a_{i-1}\}$ and any vertex in $B \setminus N(a_i)$. Hence the set $\{a_1, \ldots, a_i\} \cup B \setminus N(a_i)$ is the unique maximal independent set in G containing a_i and not containing any a_j with $j > i$. As a result, $\nu_i(G) \leq 1$ for every $i \in \{1, \ldots, k\}$. Note that B forms the only maximal independent set in G containing none of the vertices of A, and that every other maximal independent set in G contains at least one vertex from A. Since $\nu_i(G) \leq 1$ for every $i \in \{1, \ldots, k\}$ and $k \leq \lfloor n/2 \rfloor$ by assumption, we conclude that G has at most $\lfloor n/2 \rfloor + 1$ maximal independent sets.

For every even $n \geq 2$, let G_n be the chain graph obtained from two independent sets $A = \{a_1, \ldots, a_{n/2}\}$ and $B = \{b_1, \ldots, b_{n/2}\}$ by making a_i adjacent to every vertex in $\{b_1, \ldots, b_i\}$, for $i = 1, \ldots, n/2$. For every even $n \geq 2$, the graph G_n contains exactly $\lfloor n/2 \rfloor + 1$ maximal independent sets. □

Lemma 2. *For every minimal dominating set S of a chain graph G, the graph $G[S]$ contains at most one edge.*

Proof. Let S be a mds of a chain graph $G = (A, B, E)$. We first show that every connected component of $G[S]$ contains at most two vertices. Suppose, for contradiction, that $G[S]$ contains a connected component on more than two vertices. Since G is bipartite, this means that $G[S]$ contains an induced path on three vertices. Without loss of generality, let $a', a'' \in A$ and $b \in B$ be three vertices such that $\{a', b, a''\}$ induces a path on three vertices in $G[S]$. Note that b dominates both a' and a''. Hence, in order for a' and a'' to have private neighbors, there must exist vertices b' and b'' such that a' is the only vertex of S dominating b', and a'' is the only vertex of S dominating b''. This contradicts that there is a chain ordering involving a and a', and hence that G is a chain graph.

Let $\sigma_A = \langle a_1, \ldots, a_k \rangle$ and $\sigma_B = \langle b_1, \ldots, b_\ell \rangle$ form a chain ordering of G. Suppose $G[S]$ contains at least one edge, and let $a_i b_j$ be an edge of $G[S]$. We already showed that $G[S]$ does not contain an induced path on three vertices, so none of the vertices of $N(a_i) \setminus \{b_j\}$ and $N(b_j) \setminus \{a_i\}$ is in S. This means that if $G[S]$ contains an edge other than $a_i b_j$, then this edge is of the form $a_p b_q$ with $p < i$ and $q > j$, where $a_p \notin N(b_j)$ and $b_q \notin N(a_i)$. But then $N(a_p) \not\subseteq N(a_i)$, contradicting the assumption that σ_A is an ordering of the vertices of A such that $N(a_1) \subseteq \cdots \subseteq N(a_q) \subseteq \cdots \subseteq N(a_i) \subseteq \cdots \subseteq N(a_k)$. We conclude that every connected component of $G[S]$, apart from the component containing a_i and b_j, contains exactly one vertex. □

The following lemma is an easy consequence of Lemma 2.

Lemma 3. *Let ab be an edge of a chain graph $G = (A, B, E)$ with $a \in A$ and $b \in B$. If a or b has degree 1, then there is no minimal dominating set in G containing both a and b. If both a and b have degree at least 2, then there is exactly one minimal dominating set in G containing both a and b.*

From Lemmas 1, 2, and 3 we can readily deduce that every chain graph has at most $\lfloor n/2 \rfloor + m + 1$ mds. This bound is tightened below.

Theorem 5. *A chain graph on at least 2 vertices has at most $\lfloor n/2 \rfloor + m$ minimal dominating sets, and there are chain graphs with $\lfloor n/2 \rfloor + m$ minimal dominating sets.*

Proof. Let $G = (A, B, E)$ be a chain graph on n vertices, and let $\sigma_A = \langle a_1, .., a_k \rangle$ and $\sigma_B = \langle b_1, \ldots, b_\ell \rangle$ form a chain ordering of G. Without loss of generality, assume that $|A| \leq \lfloor n/2 \rfloor$. Since any isolated vertex must belong to every minimal dominating set, we may assume that G is connected. It is easy to check that G contains at most $\lfloor n/2 \rfloor + m$ mds in case $|A| = 1$ or $|B| = 1$. We therefore assume below that both A and B contain at least two vertices.

First suppose that at least one edge of G has an endpoint of degree 1. Then G has at most $m-1$ mds S such that $G[S]$ contains an edge as a result of Lemmas 2 and 3. Every other mds in G must be a maximal independent set in G, and G has at most $\lfloor n/2 \rfloor + 1$ such sets by Lemma 1. Hence G has at most $\lfloor n/2 \rfloor + m$ mds in this case.

Now suppose that for every edge of G both endpoints have degree at least 2. In particular, b_ℓ has degree at least 2, which means that b_ℓ is adjacent to a_{k-1}, which in turn implies that a_{k-1} is adjacent to every vertex in B. Recall that a_k also dominates B. Let S be a maximal independent set in G containing a_{k-1}. Since a_{k-1} dominates B, S does not contain any vertex of B. Since S is a maximal independent set of G, we must have $S = A$. In particular, $a_k \in S$. Let $\nu_i(G)$ be the number of maximal independent sets in G containing a_i, but not containing any vertex a_j with $j > i$. Note that $\nu_{k-1}(G) = 0$. As shown in the proof of Lemma 1, $\nu_i(G) \leq 1$ for every $i \in \{1, \ldots, k\}$. The set B is the only maximal independent set in G containing no vertices of A. Every other maximal independent set contains at least one vertex of A. The assumption that $|A| \leq \lfloor n/2 \rfloor$, together with $\nu_{k-1}(G) = 0$ and $\nu_i(G) \leq 1$ for every $i \in \{1, \ldots, k\}$, implies that G has at most $\lfloor n/2 \rfloor$ maximal independent sets, each of which forms a mds in G. Due to Lemmas 2 and 3, G has exactly m mds S for which $G[S]$ contains an edge. Hence G contains at most $\lfloor n/2 \rfloor + m$ mds in total.

Recall the graph G_n that was defined in the proof of Lemma 1. For every even $n \geq 2$, let G'_n be the graph obtained from G_n by adding the edge $a_{k-1}b_\ell$. The graph G'_n contains exactly $\lfloor n/2 \rfloor + m$ mds: one for each of the $\lfloor n/2 \rfloor$ maximal independent sets, and one for each edge of G'_n, apart from the edge a_1b_1. □

8 Conclusions

We established new upper bounds for the number of mds in graphs on n vertices of various graph classes. All our bounds are significantly lower than the known upper bound for general graphs.

Could our enumeration algorithms be used to establish fast exact exponential algorithms solving Domatic Number or Connected Dominating Set on split and chordal graphs? We point out that both problems are NP-complete on split graphs, and thus also on chordal graphs [10,12].

The maximum number of mds in a general graph on n vertices is still unknown. It is conjectured that $15^{n/6}$, the best known lower bound, is indeed the correct answer [4]. We have shown that a counterexample to this conjecture cannot belong to any of the graph classes studied in this paper, with the exception of cobipartite and chordal graphs. A lower bound for bipartite graphs is $6^{n/4} \approx 1.5650^n$, which is the number of mds in the disjoint union of $n/4$ cycles of length 4. Is there an upper bound for bipartite graphs which is better than 1.7159^n? Finally, we conjecture that the maximum number of mds in proper interval and split graphs is $3^{n/3}$.

References

1. Brandstädt, A., Le, V.B., Spinrad, J.: Graph Classes: A Survey. SIAM Monographs on Discrete Mathematics and Applications (1999)
2. Eppstein, D.: Small maximal independent sets and faster exact graph coloring. J. Graph Algor. Appl. 7(2), 131–140 (2003)
3. Fomin, F.V., Gaspers, S., Pyatkin, A.V., Razgon, I.: On the minimum feedback vertex set problem: Exact and enumeration algorithms. Algorithmica 52(2), 293–307 (2008)
4. Fomin, F.V., Grandoni, F., Pyatkin, A.V., Stepanov, A.A.: Combinatorial bounds via measure and conquer: Bounding minimal dominating sets and applications. ACM Trans. Algorithms 5(1) (2008)
5. Fomin, F.V., Kratsch, D.: Exact Exponential Algorithms. Texts in Theoretical Computer Science. Springer, Heidelberg (2010)
6. Fomin, F.V., Villanger, Y.: Finding induced subgraphs via minimal triangulations. In: Proceedings of STACS 2010, pp. 383–394 (2010)
7. Gaspers, S., Mnich, M.: Feedback Vertex Sets in Tournaments. In: de Berg, M., Meyer, U. (eds.) ESA 2010. LNCS, vol. 6346, pp. 267–277. Springer, Heidelberg (2010)
8. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs. Annals of Disc. Math. 57 (2004)
9. Haynes, T.W., Hedetniemi, S.T. (eds.): Domination in graphs. Marcel Dekker Inc., New York (1998)
10. Laskar, R.C., Pfaff, J.: Domination and irredundance in split graphs. Technical Report 430, Clemson University (1983)
11. Kanté, M.M., Limouzy, V., Mary, A., Nourine, L.: Enumeration of Minimal Dominating Sets and Variants. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 298–309. Springer, Heidelberg (2011)
12. Kaplan, H., Shamir, R.: The domatic number problem on some perfect graph families. Inform. Proc. Lett. 49(1), 51–56 (1994)
13. Lawler, E.L.: A note on the complexity of the chromatic number problem. Inform. Proc. Lett. 5(3), 66–67 (1976)
14. Moon, J.W., Moser, L.: On cliques in graphs. Israel J. Math. 3, 23–28 (1965)
15. Yannakakis, M.: Node deletion problems on bipartite graphs. SIAM J. Comput. 10, 310–327 (1981)

Randomized Group Testing Both Query-Optimal and Minimal Adaptive

Peter Damaschke and Azam Sheikh Muhammad

Department of Computer Science and Engineering
Chalmers University, 41296 Göteborg, Sweden
{ptr,azams}@chalmers.se

Abstract. The classical group testing problem asks to determine at most d defective elements in a set of n elements, by queries to subsets that return Yes if the subset contains some defective, and No if the subset is free of defectives. By the entropy lower bound, $\log_2 \sum_{i=0}^{d} \binom{n}{i}$ tests, which is essentially $d \log_2 n$, are needed at least. We devise group testing strategies that combine two features: They achieve this optimal query bound asymptotically, with a factor $1 + o(1)$ as n grows, and they work in a fixed number of stages of parallel queries. Our strategies are randomized and have a controlled failure probability, i.e., constant but arbitrarily small. We consider different settings (known or unknown d, probably correct or verified outcome), and we aim at the smallest possible number of stages. In particular, 2 stages are sufficient if d grows slowly enough with n, and 4 stages are sufficient if $d = o(n)$.

1 Introduction

Suppose that a set of n elements contains an unknown subset of defective elements ("defectives" for brevity). A group test takes any subset, called a pool, and returns a binary answer: The pool is positive if it contains at least one defective, and otherwise negative. The group testing problem is the problem of identifying the defectives using a minimum number of group tests, also called queries. An upper bound d on the number of defectives may be known in advance, or the number d of defectives may be unknown. However, we assume that $d \ll n$.

Group testing is a classical combinatorial search problem [10] and an important example of the "exact learning by queries" model. It has applications in biological and chemical testing and diagnosis [10,11,18], communication networks [4,9,13], and streaming algorithms [5], to mention only a few domains.

A group testing strategy works in stages, where the pools for every stage are prepared prior to the stage, and then queried in parallel. The pools for the next stage, however, may depend on all previous answers. A strategy with one query per stage is called adaptive. In most group testing applications, highly parallel strategies working in a few stages are preferable because, on the one hand, the tests are time-consuming, and on the other hand, many pools can be tested simultaneously.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 214–225, 2012.

As a notational remark, we omit ceiling brackets in expressions, in order to avoid bulky notation. Logarithms are always base 2.

Due to the entropy lower bound, also known as the information-theoretic lower bound, at least $\log_2 \sum_{i=0}^{d} \binom{n}{i} \approx d \log(n/d)$ queries are needed even by adaptive strategies. If defectives are rare, this expression simplifies to $d \log n$, subject to negligible terms. If d is known in advance, essentially $d \log n$ queries are also sufficient, and if d is unknown, still $1.5d \log n$ queries are sufficient [25]. There exist strategies using $O(d \log n)$ queries that need only two stages when d is known [8,12,3]. The query number in [3] tends to $1.44d \log n$ as d grows. A query number of the form $O(d \log n)$ cannot be achieved by any deterministic strategy in only one stage, as a consequence of known lower bounds for so called $\bar{d}$-separable pooling designs [2], which are exactly the sets of pools that can distinguish between any sets of at most d defectives. As opposed to this, there is a randomized one-stage strategy that succeeds with any prescribed probability, using asymptotically $1.45d \log n$ queries [3]. For the case of unknown d we proved in [6] that no deterministic strategy can manage with $O(d \log n)$ queries in any constant number of stages, but randomized strategies can, in only two stages.

For applications where defectives are rare but tests are expensive, it would be worthwhile to have strategies where the test number is as close as possible to the entropy lower bound, not only within some constant factor. *The principal contribution of the present paper is to show that, in fact, there exist randomized strategies that combine the two desirable features of query-optimality (at least in an asymptotic sense) and minimal adaptivity:* The constant factor in the leading term $d \log n$ of the query complexity tends to 1 as n grows, and the strategies work in a constant number of stages.

Although these results are not particularly hard to obtain, to our best knowledge this is the first paper presenting group testing strategies with this combination of desirable properties, and the way of combining known ingredients seems to be novel. The strategies also look simple enough for real use. Only elementary methods are needed to construct and analyze them.

We highlight the main results briefly, while the technical statements and also variations of the results are deferred to the following sections. If d grows slower than any power function of n, we achieve query-optimality already in 2 stages. Due to a recent negative result [23], this is not possible if d grows like $d = n^{\delta}$ for some constant exponent $\delta < 1$. But in this case we manage with 3 stages. Finally, if $d = o(n)$ we still get an asymptotically optimal query number in 4 stages. More precisely, we consider any fixed defective rate $r = d/n$ and show that our strategy approaches the entropy lower bound if $r \to 0$. This asymptotic behaviour matches known upper bounds for sequential group testing strategies, therefore one may appreciate that a constant number of stages suffices. It remains open whether even 3 stages would be enough.

An earlier negative result [6] implies that our strategies cannot be derandomized, but apparently they can be turned into deterministic strategies for the statistical model of group testing where elements are defective independently and with some fixed probability.

Due to space limitations, some parts of the proofs are only sketched, but in principle we include complete proofs. Some technicalities are omitted. For instance, when we use random subsets in a strategy, we do not always clearly distinguish between their expected and actual sizes, which however does not affect the asymptotic analysis for large n.

2 Minimal Adaptive Group Testing Close to the Entropy Lower Bound

The following observation is folklore; for completeness we give the proof.

Lemma 1. *If only one defective is present, it can be found by* $\log n + 1$ *queries in one stage.*

Proof. We introduce dummy elements if n is not a power of 2; this can at most double the number of elements. Then we index the elements as bit vectors of length $\log n + 1$. For each i we query a pool consisting of all elements that have entry 1 at the ith position. Obviously, the answers localize one defective provided that there is exactly one. □

Remark: This pooling design cannot check whether $d = 1$. A very minor issue is that we cannot see whether the element indexed by the zero vector is defective, if all pools were negative. Obviously we can catch up this case by one additional query (in the same stage) to this element. Much more importantly, if $d > 1$, it is possible that the strategy cannot safely identify any defective.

We also apply Theorem 10 from [3] that we rewrite as follows:

Lemma 2. *With prescribed probability* $1-\epsilon_1$ *one can correctly identify at most* d *defectives using* $O(d(\log n+\log(1/\epsilon_1)))$ *queries in one stage. The hidden constant factor is at most 1.9 and converges to 1.45 as* d *grows.* □

Theorem 1. *Using* $d \log n + O(d \log d) + O(d \log(1/\epsilon))$ *queries in two stages we can, with probability* $1 - \epsilon$*, correctly identify all defectives, provided that at most* d *defectives are present. The hidden constant factors in the lower-order terms are below* 3.8*, and tend to* 2.9 *as* d *grows.*

Proof. The overall scheme is very simple: In stage 1 we separate the defectives with probability $1 - \epsilon$, that is, we divide the elements into disjoint subsets each containing exactly one defective (plus one subset without defectives). In stage 2 we apply Lemma 1 to every such subset. It remains to discuss stage 1 in detail.

For some q to be specified below, we assign every element one of q labels, each with probability $1/q$. Elements with the same label form one cell. Like pools, a cell is said to be positive if it contains a defective, and otherwise negative. Then we apply Lemma 2 to the set of cells rather than individual elements: We can,

with prescribed probability $1-\epsilon_1$, correctly identify the (at most d) positive cells using $O(d(\log q + \log(1/\epsilon_1)))$ queries in one stage. The positive cells are our disjoint sets to be used in stage 2.

The probability that any two defectives collide, i.e., get into the same cell, is at most $\binom{d}{2}/q < d^2/2q$. In order to keep this probability below some ϵ_2 we choose $q = d^2/2\epsilon_2$, thus $\log q = 2\log d + \log(1/\epsilon_2) - 1$. The strategy gives an incorrect result with probability at most $\epsilon := \epsilon_1 + \epsilon_2$. Now, minimizing the query bound is equivalent to minimizing $\log(1/\epsilon_1) + \log(1/\epsilon_2)$ under the constraint $\epsilon = \epsilon_1 + \epsilon_2$. A standard calculation yields $\epsilon_1 = \epsilon_2 = \epsilon/2$, and obvious further manipulations give the final query bound. The constants follow from Lemma 2. □

Remarks:
(1) For every fixed d, this bound converges to the entropy lower bound as n grows. This asymptotic optimality holds even for d growing slowly with n (e.g., polylogarithmic). It remains open how many randomized queries would be actually needed in one stage. To our best knowledge, the current upper bound is the mentioned $1.45d\log n$ from [3]. Is it possible to identify d defectives, with fixed probability $1-\epsilon$, by essentially $d\log n$ queries in only one stage? Or can a non-trivial lower bound $a(\epsilon)d\log n$ for some $a(\epsilon) > 1$ be proved?

(2) The known deterministic two-stage strategies using $O(d\log n)$ queries, however with a constant strictly larger than 1, determine $O(d)$ candidate positives in stage 1, and need only $O(d)$ queries in stage 2 to test them [8,12,3]. Amazingly, in our randomized strategy the situation is exactly the opposite: The complexity of stage 1 does not depend on n, and the main work is done in stage 2. An interesting question is whether there exists a query-optimal two-stage strategy where the workload is balanced.

(3) The strategy in Theorem 1, with $q = \Theta(d^2)$, is designed for any constant failure probability. By choosing q as a larger polynomial in d, or even as a slow function of n, we can make the failure probability vanish asymptotically, without destroying the asymptotic query-optimality. Depending on the choice of q, different patterns of asymptotic behaviour can be achieved.

With one additional stage we can improve the query bound:

Theorem 2. *Using $d\log(n/d) + O(d\sqrt{\log d}\log\log d)$ queries in three stages we can, with probability $1-\epsilon$, correctly identify all defectives, provided that at most d defectives are present.*

Proof. We give a high-level description of the strategy: Partition the elements randomly into d bags[1] of size n/d. Due to well-known load balancing results (see [20]), with high probability all bags contain fewer than $\log d$ defectives. We call a bag sparse/dense if it has fewer/more than $\sqrt{\log d}$ defectives. In stage 1 we distinguish between sparse and dense bags using $L(n/d)$ queries in each bag, where L is any sublogarithmic function. It suffices to query random pools of size

[1] We call them "bags" because their role is different from the "cells" used earlier.

around $n/(d\sqrt{\log d})$ and decide sparse or dense based on the fraction of positive answers. We skip the details, since the only crucial point is that the pool number increases with n/d, thus we can make the error probability arbitrarily small. The rest is to apply the strategy from Theorem 1 in parallel to each bag. In the, up to d, sparse bags we may use $q = \Theta((\sqrt{\log d})^3)$, thus $O(d\sqrt{\log d}\log\log d)$ queries are needed in all sparse bags. In the, up to $d/\sqrt{\log d}$, dense bags we may use $q = \Theta((\log d)^3)$, thus $O((d/\sqrt{\log d})\log d\log\log d)$ queries are needed also in all dense bags. Here, exponent 3 in q is chosen to keep the failure probability in each bag $O(1/d)$. In the final stage we search for the separated defectives individually, among at most n/d elements. □

The advantage of Theorem 2 is that this complexity approaches the entropy lower bound for larger d, such as $d = n^\delta$, $\delta < 1$. Interestingly, it is known that two stages are not enough for that, due to a lower bound of $(\log e)^2 d\log(n/d)$ if d grows like $d = n^\delta$ [23]. (Actually, this result was derived for the statistical model of group testing with independent random defectives, but asymptotically the models are equivalent.)

Our next issue is that the outcome in Theorem 1 is correct with some prescribed probability, but in every specific case the searcher cannot be sure that the returned set of defectives is correct. Trivially, any group testing result can be verified in another stage with $d+1$ queries. But can we accomplish a correct *and* verified outcome without the extra stage? When determining the positive cells in stage 1 we may get some false positives as well. However, the subroutine from [3] never yields false negatives, and the false positive cells will be detected in stage 2. The real difficulty is that the separation can fail, too. More than one defective can get into one cell, and then the simple search as in Lemma 1 does not work; remember the remark after Lemma 1. However, with a slight increase of the test number we can also verify the outcome, as we will see below. First we need another search method for single defectives, known from [26]:

Lemma 3. *Using* $\log n + 0.5\log\log n + o(\log\log n)$ *queries in one stage, we can achieve the following: If only one defective is present, we identify it and confirm that it is the only one. If more defectives are present, we recognize this fact (but we do not necessarily identify some of the defectives in this case). Moreover, this query number is optimal for this purpose.* □

For clarity we outline the (known) strategy: The design consists of t pools, where each of the n elements is in exactly $t/2$ pools. Choose t even (or round $t/2$), and make t large enough so that $\binom{t}{t/2} = n$.

Along the lines of Theorem 1 it follows immediately:

Theorem 3. *Using* $d(\log n + 0.5\log\log n + o(\log\log n)) + O(d\log d) + O(d\log(1/\epsilon))$ *queries in two stages we can, with probability* $1-\epsilon$*, identify all defectives and verify that we found them all, provided that at most* d *defectives are present.* □

Note that the extra terms are $o(\log n)$, hence this result still matches asymptotically the entropy lower bound. Nevertheless it is interesting to ask if the

$\log\log n$ term is avoidable. For two stages we must leave this as an open question. In three stages we can get rid of the $\log\log n$ term, by applying Lemma 1 and an obvious verification step.

In the following we give a side result related to that. It further extends the optimality statement from Lemma 3, in that it shows that one cannot even narrow down the candidates for the defective to a small set, in one stage with fewer queries. Group testing strategies that apply some pooling design in stage 1 and then test the candidates individually in stage 2 are well established as "trivial strategies" (which is perhaps a misleading name). They are of particular practical interest because no pools at all depend on test results and must be created on-the-fly: Stage 1 is prepared in advance, and only trivial testing is done in stage 2.

Theorem 4. *Suppose that actually one defective is present (but the searcher is not sure about the number of defectives and needs to confirm it). Then, with fewer than* $\log n + 0.5 \log\log n - \Theta(\log c(n))$ *queries in one stage it is impossible to narrow down the candidate set for the defective to size* $c(n)$*. Here,* c *is any function with* $c(n) = o(\log n)$*.*

Proof. Consider any design of t pools, arbitrarily indexed $1, \ldots, t$. We define the indicator of an element to be the t-bit vector x where the ith position x_i is 1 if the element belongs to the ith pool, and $x_i = 0$ else. Imagine that an adversary declares one element defective, chosen randomly with probability $1/n$. For a t-bit vector x, let $p(x)$ denote the probability that the defective has indicator x. In other words, $p(x) \cdot n$ elements have indicator x.

For two t-bit vectors x and y, symbol $y \leq x$ means that y is bitwise smaller than x, that is, $y_i = 1$ implies $x_i = 1$. If the defective has indicator x then exactly those elements with indicators $y \leq x$ are candidates for being defective: Note that all pools i with $x_i = 1$ answered positively, and elements with indicators $y \leq x$ occur in positive pools only, thus the searcher cannot surely recognize them as negative.

We conclude that the (conditional) expected number of candidates is now $n \sum_{y \leq x} p(y)$ for any fixed indicator x of the defective, hence the expected number of candidates is $n \sum_{y \leq x} p(x)p(y)$, where the sum is now taken over all such pairs (y, x). In order to get a lower bound for this expression, we choose the distribution $p(x)$ so as to minimize $n \sum_{y \leq x} p(x)p(y)$ under the constraint $\sum_x p(x) = 1$.

In fact, this optimization problem is not hard to solve. Define the support of a distribution to be the set of all x with $p(x) > 0$. First we claim that, in some optimal solution, the support is an antichain (set of pairwise incomparable vectors) in the set of t-bit vectors partially ordered by $\leq$. If the support A is not an antichain, take some minimal vector $y \in A$ that is smaller than some other vectors in A, and move $p(y)$ arbitrarily to these larger members of A. It is easy to see that this cannot increase our double sum (since no further "comparable pairs" are created). Hence we can repeat this manipulation until A is an antichain. But then our double sum simplifies to $n \sum_{x \in A} p(x)^2$. A sum of squares of numbers with fixed sum is minimized if all these numbers are equal. With $a := |A|$ we get $na(1/a)^2 = n/a$ expected candidates. In order to keep

this number below c, we need $a \geq n/c$. Due to Sperner's Theorem [27,22], the largest antichain in the partial order of t-bit vectors has size $a = \binom{t}{t/2}$. Thus, the known lower bound $\log a + 0.5 \log\log a$ for t yields the asserted lower bound in argument n.

To conclude, when the defective is chosen at random, then any deterministic strategy with fewer queries returns a candidate set whose expected size is not bounded by c. Hence there exists an element v such that, if v is the defective, more than c candidates actually remain. With Yao's technique (see, e.g., Section 2.2.2 in [24]), the same lower bound follows for randomized designs. □

Possibly this negative statement could also be derived from [21], but in order to make the paper more self-contained we keep our shorter proof of the explicit bound.

The results so far were formulated for the case of a known d, or more realistically, a known upper bound d. With an additional stage using a procedure from [6] we get rid of this restrictive assumption. For this we need a slight adaptation of a result from [6] saying that $O(\log n)$ nonadaptive random queries are sufficient to find, with any fixed success probability, an upper bound $O(d)$ for d. The basic idea is to test random pools of exponentially growing size, and then the cut-off point between negative and positive pools gives an estimate of d. We remark that $\Omega(\log n)$ queries are also necessary, at least for some restricted but very natural type of random pools, as shown in [7].

Theorem 5. *For an arbitrarily small fixed $g > 0$ and for any fixed constant success probability $1 - \epsilon$, using $(d + g) \log n + O(d \log d)$ queries in three stages we can correctly identify all d defectives even without prior knowledge of d.*

Proof. In stage 1 is we use $g \log n$ pools to output an upper bound $O(d)$ for d, where the hidden constant in $O(d)$ depends only on g and on a prescribed failure probability (of underestimating d) [6]. Stage 2 and 3 consist of the strategy from Theorem 1, with the only modification that the number q of cells is chosen based on the upper bound for d returned by stage 1. Since this upper bound is $O(d)$, only the constant factor in the $O(d \log d)$ term is affected. □

Note that also this result gets arbitrarily close to the entropy lower bound as n grows. Moreover, factor $1 + g/d$ of the dominating term $d \log n$ can be bounded arbitrarily close to 1, uniformly for all d (by choosing g small enough), and for every fixed g it converges to 1 for growing d. An open question is whether we can accomplish the same characteristics as in Theorem 5 already in two stages. If d happens to be $o(\log n / \log\log n)$ (but the magnitude of d is still unknown in advance), we can actually manage this task in two stages, by applying the strategy from Theorem 1 or 3 with some $q = o((\log n / \log\log n)^2)$. But we conjecture that this is no longer possible for larger d.

3 Linear versus Sublinear Growth of the Defectives

The previous results hold for cases when d grows slower than n. However, in many practical settings one would rather expect a constant rate of defectives

$r := d/n$. In the following we also address this case. We assume r to be known in advance, otherwise we can first estimate r by $O(\log n)$ randomized nonadaptive group tests [6]. While the hidden constant depends on the accuracy of estimating r, the query number becomes negligible as n grows, since $\log n/(d\log(n/d)) = \log n/(nr\log(1/r))$ tends to 0.

This section consists of two parts. As a benchmark we first discuss adaptive, i.e., sequential testing. Then we show that 4 stages are enough to achieve a similar test complexity.

We call the model of group testing with a specified number d of defectives (which is either a known or a maximum number) the *combinatorial model*. In the *statistical model* of group testing, elements are defective independently and with some fixed probability r. When we have a strategy S for the statistical model and an input with d defectives, we may shuffle the elements and then apply strategy S for $r = d/n$. Since pools being significantly larger than $1/r$ are almost certainly positive and give little information, we can restrict pools to sizes $O(1/r)$ regardless of n. Thus, for large n the statistical model with probability r can be adopted instead of the combinatorial model with exactly d defectives. In the remainder of the section we assume the statistical model.

The entropy lower bound is now $r\log(1/r) + (1-r)\log(1/(1-r))$ queries per element, or equivalently,

$$\log(1/r) + (1-r)\log(1/(1-r))/r$$

queries per defective. This follows easily from the additivity of entropy. For small r this simplifies to $\log(1/r) + \log e$ queries per defective. It might be interesting to notice that this lower bound also holds for any randomized strategy that identifies d defectives in the combinatorial model, although an exact d means some more prior knowledge for the searcher. This follows from a more general fact (not referring especially to the group testing problem):

Proposition 1. *Let H be a set of h hypotheses, and suppose that a searcher can ask binary queries. Then no randomized strategy can guarantee to identify the correct hypothesis using an expected number of less than* $\log h$ *queries.*

Proof. Suppose that an adversary selects every hypothesis with probability $1/h$ as the true one. Then any deterministic strategy needs an expected number of at least $\log h$ queries, because every strategy can be viewed as a Huffman code with the expected query number as the average path length, and then the claim is easily seen from Huffman's algorithm [16] applied to the equal-probability case. From this, Yao's lower bound technique yields the assertion as follows. Any randomized strategy R can be seen as a probability distribution on the deterministic strategies. Hence the expected query number of R on the randomized input is at least $\log h$. It follows the existence of a specific hypothesis where R needs at least $\log h$ expected queries. □

In our case, this lower bound is $\log\binom{n}{d}$ and amounts to the same bound as before (with $-o(1)$ terms neglected), by routine calculations using Stirling's

formula. Recall that we aim at strategies with an expected query number as close as possible to the lower bound. In a special type of sequential strategies, elements are arranged as a sequence, in any fixed linear order, and then they search for the leftmost defective by querying only pools that are prefixes of this sequence. This restriction leads to a well-studied problem from quality control [14,15,1]. Known results from there can be rephrased as follows.

Proposition 2. *The group testing problem with fixed rate r of defectives can be solved sequentially with $\log(1/r) + O(1) = (1 + o(1))\log(1/r)$ queries per defective, where $o(1)$ vanishes for $r \to 0$.* □

The $o(1)$ term cannot be avoided, even in sequential strategies. Therefore it is interesting that this asymptotic behaviour, perhaps with an $o(1)$ term going slower to 0, can be achieved already in a small constant number of stages. For this result we can, in the following, focus attention on "small" r, which also allows us to neglect some technicalities like rounding. We stress that the announced result does not follow from the techniques of Section 2: Observe that we needed $d\log(n/d) + dM(d)$ queries, for some unbounded monotone function M. These are $\log(1/r) + M(d)$ queries per defective, that is, the additive term would grow infinitely with the input size even if r is fixed. In fact, we will need some more stages to avoid that.

Finally, as a preparation we reconsider one of the strategies in [3] and present a version that it guaranteed to find all defectives in two stages. Note that query numbers stated below are meant to be expected numbers.

Lemma 4. *In two stages using $1.9\log(1/r)+1$ queries per defective, where $r := d/n$, we can identify all d defectives.*

Proof. Query nonadaptively a sufficient number of random pools of size $1/r$, and discard the elements in negative pools. (The information in positive pools is not used further.) Every non-defective shall be kept with probability at most er, where e denotes Euler's number. If k denotes the number of negative pools, it is sufficient to have $(1 - 1/(rn))^k = er$. For large n this can be transformed into $e^{-k/rn} = er$, hence $k/rn = \ln(1/r) - 1$, which means $\ln(1/r) - 1$ negative pools per defective. Since a pool of size $1/r$ is negative with probability approximately $1/e$, this stage needs $e(\ln(1/r) - 1) = 1.9\log(1/r) - e$ queries per defective. In a second stage, the $(1 + e)rn$ remaining candidates are tested individually, thus the total number of queries per defective is $1.9\log(1/r) + 1$. □

Now we are ready for the main result. Basically it says that we can approach the entropy lower bound in 4 stages when $d = o(n)$.

Theorem 6. *Group testing with defective rate r can be solved in four stages using $(1 + o(1))\log(1/r)$ queries per defective, where $o(1)$ vanishes for $r \to 0$.*

Proof. We split the elements in disjoint cells of x/r elements, where x is a free parameter. We choose x depending on r such that, $\lim_{r\to 0} x = 0$ but

$\lim_{r\to 0} x\log(1/r) = \infty$, which also implies $\lim_{r\to 0} x/r = \infty$. The expected number of defectives in a cell is x. Below we will use well-known inequalities like $1 - x < e^{-x} < 1 - x + x^2/2$ and $e^x < 1 + x + x^2$ (for small x).

Remember that we are going to prove an asymptotic bound. Since $r \to 0$ but the cell size grows, the number of defectives in a cell follows, in the limit, a Poisson distribution with expectation x. (We omit a detailed technical discussion with error bounds.) In particular, we can assume that a cell has 0, 1, and more than 1 defective with probability e^{-x}, xe^{-x}, and $1 - (1 + x)e^{-x}$, respectively. We call these cells type 0, 1, and 2, respectively.

In stage 1 we simply query each cell, thus we recognize the type-0 cells, using $1/x$ queries per defective. In stage 2 we apply Lemma 3 to tell apart the type-1 and type-2 cells, and to find the unique defective in the type-1 cells. This needs

$$\log(x/r) + (0.5 + o(1)) \log\log(x/r)$$

queries in each type-1 or type-2 cell, and identifies an e^{-x} fraction of the defectives. Here, $o(1)$ denotes a term that vanishes for $x/r \to \infty$. For every type-1 cell there exist on average

$$(1 - (1 + x)e^{-x})/(xe^{-x}) = (e^x - 1 - x)/x$$

type-2 cells, that is, $(e^x - 1)/x < 1 + x$ type-1 *and* type-2 cells per type-1 cell.

Hence we have used fewer than

$$(1 + x)\log(x/r) + (0.5 + o(1)) \log\log(x/r)) + 1/x$$

queries per recognized defective, in stage 1 and 2. For $x \to 0$ and $x/r \to \infty$ this simplifies to

$$(1 + o(1)) \log(x/r) + 1/x < (1 + o(1)) \log(1/r) + 1/x,$$

since log log grows slower than log.

In stage 3 and 4 we merge all type-2 cells and find the remaining defectives using Lemma 4. They make up an $1 - e^{-x}$ fraction of all defectives, and the total size of type-2 cells is $1 - (1 + x)e^{-x}$ times the original number of elements. Hence the defective rate is

$$r' = r(1 - e^{-x})/(1 - (1 + x)e^{-x}).$$

Due to Lemma 4 we need $1.9 \log(1/r') + 1$ queries per defective from type-2 cells, which are

$$(1 - e^{-x})(1.9 \log(1/r') + 1) < 1.9x \log(1/r') + x$$

queries per defective. Furthermore we have

$$1/r' = (1 - (1 + x)e^{-x})/(1 - e^{-x}) \cdot (1/r).$$

This expression is smaller than

$$(1 - (1 + x)(1 - x))/(1 - (1 - x + x^2/2)) \cdot (1/r) = x/(1 - x/2) \cdot (1/r).$$

Note that for $x \to 0$, the upper bound expression for $1/r'$ tends to x/r. Thus we have used

$$1.9x \log(x/r) + x < 1.9x \log(1/r) + x$$

queries per defective in stage 3 and 4.

The total number of queries per defective from all stages is still described by

$$(1 + o(1)) \log(1/r) + 1/x.$$

Since $\lim_{r\to 0} x \log(1/r) = \infty$, clearly $(1/x)/\log(1/r)$ tends to 0, thus the $1/x$ term is redundant. □

Acknowledgments. This work has been supported by the Swedish Research Council (Vetenskapsrådet), grant No. 2010-4661, "Generalized and fast search strategies for parameterized problems". Part of the work was inspired by the seminar "Search Methodologies II" (2010) organized by Ahlswede, R. and Cicalese, F. at the Center for Interdisciplinary Research, University of Bielefeld, Germany. We also thank the reviewers for their encouraging remarks and useful hints.

References

1. Ben-Gal, I.: An Upper Bound on the Weight-Balanced Testing Procedure with Multiple Testers. IIE Trans. 36, 481–493 (2004)
2. Chen, H.B., Hwang, F.K.: Exploring the Missing Link Among d-Separable, $\bar{d}$-Separable and d-Disjunct Matrices. Discr. Appl. Math. 155, 662–664 (2007)
3. Cheng, Y., Du, D.Z.: New Constructions of One- and Two-Stage Pooling Designs. J. Comp. Biol. 15, 195–205 (2008)
4. Clementi, A.E.F., Monti, A., Silvestri, R.: Selective Families, Superimposed Codes, and Broadcasting on Unknown Radio Networks. In: SODA 2001, pp. 709–718. ACM/SIAM (2001)
5. Cormode, G., Muthukrishnan, S.: What's Hot and What's Not: Tracking Most Frequent Items Dynamically. ACM Trans. Database Systems 30, 249–278 (2005)
6. Damaschke, P., Sheikh, M.A.: Competitive Group Testing and Learning Hidden Vertex Covers with Minimum Adaptivity. Discr. Math. Algor. Appl. 2, 291–311 (2010)
7. Damaschke, P., Muhammad, A.S.: Bounds for Nonadaptive Group Tests to Estimate the Amount of Defectives. In: Wu, W., Daescu, O. (eds.) COCOA 2010, Part II. LNCS, vol. 6509, pp. 117–130. Springer, Heidelberg (2010)
8. De Bonis, A., Gasieniec, L., Vaccaro, U.: Optimal Two-Stage Algorithms for Group Testing Problems. SIAM J. Comp. 34, 1253–1270 (2005)
9. De Bonis, A., Vaccaro, U.: Constructions of Generalized Superimposed Codes with Applications to Group Testing and Conflict Resolution in Multiple Access Channels. Theor. Comp. Sc. 306, 223–243 (2003)
10. Dorfman, R.: The Detection of Defective Members of Large Populations. The Annals of Math. Stat. 14, 436–440 (1943)
11. Du, D.Z., Hwang, F.K.: Pooling Designs and Nonadaptive Group Testing. Series on Appl. Math., vol. 18. World Scientific (2006)

12. Eppstein, D., Goodrich, M.T., Hirschberg, D.S.: Improved Combinatorial Group Testing Algorithms for Real-World Problem Sizes. SIAM J. Comp. 36, 1360–1375 (2007)
13. Goodrich, M.T., Hirschberg, D.S.: Improved Adaptive Group Testing Algorithms with Applications to Multiple Access Channels and Dead Sensor Diagnosis. J. Comb. Optim. 15, 95–121 (2008)
14. Hassin, R.: A Dichotomous Search for a Geometric Random Variable. Oper. Res. 32, 423–439 (1984)
15. He, Q.M., Gerchak, Y., Grosfeld-Nir, A.: Optimal Inspection Order When Process Failure Rate is Constant. Int. J. Reliability, Quality and Safety Eng. 3, 25–41 (1996)
16. Huffman, D.A.: A Method for the Construction of Minimum-Redundancy Codes. Proc. IRE 40, 1098–1101 (1952)
17. Iwen, M.A., Tewfik, A.H.: Adaptive Group Testing Strategies for Target Detection and Localization in Noisy Environments. IMA Preprint Series no. 2311. Univ. of Minnesota (2010)
18. Kahng, A.B., Reda, S.: New and Improved BIST Diagnosis Methods from Combinatorial Group Testing Theory. IEEE Trans. CAD of Integr. Circuits and Systems 25, 533–543 (2006)
19. Kainkaryam, R.M., Bruex, A., Gilbert, A.C., Schiefelbein, J., Woolf, P.J.: poolMC: Smart Pooling of mRNA Samples in Microarray Experiments. BMC Bioinf. 11, 299 (2010)
20. Kleinberg, J., Tardos, E.: Algorithm Design. Addison-Wesley, Boston (2006)
21. Kleitman, D.: On a Conjecture of Erdös–Katona on Commensurable Pairs Among Subsets of an n–Set. In: Erdös, P., Katona, G. (eds.) Theory of Graphs, Colloq. Proc., pp. 215–218. Akademiai Kiado, Budapest (1968)
22. Lubell, D.: A Short Proof of Sperner's Lemma. J. Comb. Theory 1, 299 (1966)
23. Mézard, M., Toninelli, C.: Group Testing With Random Pools: Optimal Two-Stage Algorithms. IEEE Trans. Info. Th. 57, 1736–1745 (2011)
24. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge Univ. Press (1995)
25. Schlaghoff, J., Triesch, E.: Improved Results for Competitive Group Testing. Comb. Prob. and Comp. 14, 191–202 (2005)
26. Spencer, J.: Minimal Completely Separating Systems. J. Combin. Theory 8, 446–447 (1970)
27. Sperner, E.: Ein Satz über Untermengen einer endlichen Menge. Math. Zeitschrift 27, 544–548 (1928) (in German)

Complexity of Model Checking for Modal Dependence Logic

Johannes Ebbing* and Peter Lohmann**

Institut für Theoretische Informatik
Leibniz Universität Hannover
Appelstr. 4, 30167 Hannover, Germany
{ebbing,lohmann}@thi.uni-hannover.de

Abstract. Modal dependence logic (MDL) was introduced recently by Väänänen. It enhances the basic modal language by an operator $=(\cdot)$. For propositional variables $p_1, \ldots, p_n$ the atomic formula $=(p_1, \ldots, p_{n-1}, p_n)$ intuitively states that the value of p_n is determined solely by those of $p_1, \ldots, p_{n-1}$.

We show that model checking for MDL formulae over Kripke structures is *NP*-complete and further consider fragments of MDL obtained by restricting the set of allowed propositional and modal connectives. It turns out that several fragments, e.g., the one without modalities or the one without propositional connectives, remain *NP*-complete.

We also consider the restriction of MDL where the length of each single dependence atom is bounded by a number that is fixed for the whole logic. We show that the model checking problem for this bounded MDL is still *NP*-complete. Furthermore we almost completely classifiy the computational complexity of the model checking problem for all restrictions of propositional and modal operators for both unbounded as well as bounded MDL.

An extended version of this article can be found on arXiv.org [3].

ACM Subject Classifiers: F.2.2 Complexity of proof procedures; F.4.1 Modal logic; D.2.4 Model checking.

Keywords: dependence logic, modal logic, model checking, computational complexity.

1 Introduction

Dependence among values of variables occurs everywhere in computer science (databases, software engineering, knowledge representation, AI) but also the social sciences (human history, stock markets, etc.). In his monograph [9] in 2007 Väänänen introduced functional dependence into the language of first-order logic.

* Supported by DAAD grant 50740539a within the PPP programme.
** Supported by the NTH Focused Research School for IT Ecosystems.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 226–237, 2012.

Functional dependence of the value of q from the values of $p_1, \ldots, p_n$ means that there exists a determining function f with $q = f(p_1, \ldots, p_n)$, i.e., the value of q is completely determined by the values of $p_1, \ldots, p_n$ alone. We denote this form of dependence (or determination) by the *dependence atom* $=(p_1, \ldots, p_n, q)$. To examine dependence between situations, plays, worlds, events or observations we consider collections of these, so called *teams*. For example, a database can be interpreted as a team. In this case $=(p_1, \ldots, p_n, q)$ means that in every record the value of the attribute q is determined by the values of the attributes $p_1, \ldots, p_n$.

As was introduced in [10], a team in modal logic is a set of worlds in a Kripke structure. Here $=(p_1, \ldots, p_n, q)$ means that in every world of the team the value of the atomic proposition q is determined by the propositions $p_1, \ldots, p_n$, i.e., there is a fixed Boolean function $f : \{0,1\}^n \to \{0,1\}$ that determines the value of q from the values of $p_1, \ldots, p_n$ for all worlds in the team. In first-order logic $=(x_1, \ldots, x_n, y)$ means the same for a function $f : A^n \to A$ where A is the universe of a first-order structure. *Dependence logic* [9] is then defined by simply adding dependence atoms to usual first-order logic and *modal dependence logic* (MDL) [10,8] is defined by introducing dependence atoms to modal logic.

Besides the inductive semantics (which we will use here) Väänänen also gave two equivalent game-theoretic semantics for MDL [10]. Sevenster showed that for singleton sets of worlds there exists a translation from MDL to plain modal logic [8]. Sevenster also showed that the satisfiability problem for MDL is *NEXPTIME*-complete [8] and Lohmann and Vollmer continued the complexity analysis of the satisfiability problem for MDL by systematically restricting the set of allowed modal and propositional operators and completely classifying the complexity for all fragments of MDL definable in this way [7].

The method of systematically classifying the complexity of logic related problems by restricting the set of operators allowed in formulae goes back to Lewis who used this method for the satisfiability problem of propositional logic [6]. Recently it was, for example, used by Hemaspaandra et al. for the satisfiability problem of modal logic [4,5]. The motivation for this approach is that by systematically examining all fragments of a logic one might find a fragment which allows for efficient algorithms but still has high enough expressivity to be useful in practice. On the other hand, this systematic approach usually leads to insights into the sources of hardness, i.e., the exact components of the logic that make satisfiability or model checking hard.

In this paper we transfer the method from satisfiability [7] to model checking and classify the model checking problem for almost all fragments of MDL definable by restricting the set of allowed modal ($\Box$, $\Diamond$) and propositional ($\wedge$, $\vee$, $\neg$) operators to an arbitrary subset of all operators. Given a formula, a team, and a Kripke structure, the model checking problem asks whether the structure and the team satisfy the formula. For plain modal logic this problem is solvable in P as shown by Clarke et al. [2]. A detailed complexity classification for the model checking problem over fragments of modal logic was shown by Beyersdorff et al. [1] (who investigate the temporal logic CTL which contains plain modal logic as a special case).

Table 1. Classification of complexity for fragments of MDL-MC

Operators					Complexity	Reference
$\Box$	$\Diamond$	$\wedge$	$\vee$	$\neg$	$=$	
$*$	$*$	$+$	$+$	$*$	$+$	*NP*-complete	Theorem 1
$+$	$*$	$*$	$+$	$*$	$+$	*NP*-complete	Theorem 3
$*$	$+$	$*$	$*$	$*$	$+$	*NP*-complete	Theorem 2
$*$	$-$	$*$	$-$	$*$	$*$	in P	Theorem 4
$-$	$-$	$-$	$+$	$*$	$+$	in *NP*	Proposition 1
$*$	$*$	$*$	$*$	$*$	$-$	in P	[2]

$+$: operator present $-$: operator absent $*$: complexity independent of operator

Table 2. Classification of complexity for fragments of MDL_k-MC with $k \geq 1$

Operators					Complexity	Reference
$\Box$	$\Diamond$	$\wedge$	$\vee$	$\neg$	$=$	
$*$	$*$	$+$	$+$	$*$	$+$	*NP*-complete	Theorem 1
$+$	$*$	$*$	$+$	$*$	$+$	*NP*-complete	Theorem 3
$*$	$+$	$+$	$*$	$*$	$+$	*NP*-complete	Theorem 7
$*$	$+$	$*$	$+$	$*$	$+$	*NP*-complete	Theorem 8
$*$	$*$	$-$	$-$	$*$	$*$	in P	Theorem 6
$*$	$-$	$*$	$-$	$*$	$*$	in P	Theorem 4
$-$	$-$	$-$	$*$	$*$	$*$	in P	Theorem 5
$*$	$*$	$*$	$*$	$*$	$-$	in P	[2]

$+$: operator present $-$: operator absent $*$: complexity independent of operator

In the case of MDL it turns out that model checking is *NP*-complete in general and that this still holds for several seemingly quite weak fragments of MDL, e.g., the one without modalities or the one where nothing except dependence atoms and $\Diamond$ is allowed (first and third line in Table 1).

Furthermore it seems natural to not only restrict modal and propositional operators but to also impose restrictions on dependence atoms. One such restriction is to limit the arity of dependence atoms, i.e., the number n of variables $p_1, \ldots, p_n$ by which q has to be determined to satisfy the formula $=(p_1, \ldots, p_n, q)$, to a fixed upper bound $k \geq 0$ (the logic is then denoted by $\mathsf{MDL_k}$). For this restriction model checking remains *NP*-complete in general but, for the fragment with only the $\Diamond$ operator allowed, this does not hold anymore (fifth line in Table 2). In this case either $\wedge$ (third line in Table 2) or $\vee$ (fourth line in Table 2) is needed to still get *NP*-hardness.

We classify the complexity of the model checking problem for fragments of MDL with unbounded as well as bounded arity dependence atoms. We are able to determine the tractability of each fragment except the one where formulae are built from atomic propositions and unbounded dependence atoms only by disjunction and negation (fifth line in Table 1). In each of the other cases we either show *NP*-completeness or show that the model checking problem admits an efficient (polynomial time) solution.

In Table 1 we list our complexity results for the cases with unbounded arity dependence atoms and in Table 2 for the cases with an a priori bound on the arity. In these tables a "+" means that the operator is allowed, a "-" means that the operator is forbidden and a "*" means that the operator does not have any effect on the complexity of the problem.

2 Modal Dependence Logic

We will briefly present the syntax and semantics of MDL. For a more in-depth introduction we refer to Väänänen's definition of MDL [10] and Sevenster's analysis of model-theoretic aspects and the complexity of the satisfiability problem [8] which also contains a self-contained introduction to MDL.

Definition 1 (Syntax of MDL).
Let AP be an arbitrary set of atomic propositions and $p_1, \dots, p_n, q \in AP$. Then MDL *is the set of all formulae built from the following rules:*

$$\varphi ::= \top \mid \bot \mid q \mid \neg q \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \Box\varphi \mid \Diamond\varphi \mid =(p_1, \dots, p_n, q) \mid \neg =(p_1, \dots, p_n, q).$$

Note that negation is only atomic, i.e., it is only defined for atomic propositions and dependence atoms.

We sometimes write $\Box^k$ (resp. $\Diamond^k$) for $\overbrace{\Box\Box\dots\Box}^{k \text{ times}}$ (resp. $\overbrace{\Diamond\Diamond\dots\Diamond}^{k \text{ times}}$). For a dependence atom $=(p_1, \dots, p_n, q)$ we define its *arity* as n, i.e., the arity of a dependence atom is the arity of the determining function whose existence it asserts.

In Section 4 we will investigate the model checking problem for the following logic.

Definition 2 ($\mathsf{MDL_k}$).
$\mathsf{MDL_k}$ *is the subset of* MDL *that contains all formulae which do not contain any dependence atoms whose arity is greater than k.*

We will classify MDL for all fragments defined by sets of operators.

Definition 3 ($\mathsf{MDL}(M)$).
Let $M \subseteq \{\Box, \Diamond, \wedge, \vee, \neg, =\}$. By $\mathsf{MDL}(M)$ *(resp.* $\mathsf{MDL_k}(M)$*) we denote the subset of* MDL *(*$\mathsf{MDL_k}$ *resp.) built from atomic propositions using only operators from M. We sometimes write* $\mathsf{MDL}(op1, op2, \dots)$ *instead of* $\mathsf{MDL}(\{op1, op2, \dots\})$.

Definition 4 (Kripke structure).
*An AP-*Kripke structure *is a tuple $W = (S, R, \pi)$ where S is an arbitrary non-empty set of* worlds*, $R \subseteq S \times S$ is the* accessibility relation *and $\pi : S \to \mathcal{P}(AP)$ is the* labeling function.

MDL formulae are interpreted over Kripke structures but in contrast to common modal logics, truth of a MDL formula is not defined with respect to a single world of a Kripke structure but with respect to a set (or *team*) of worlds.

Definition 5 (Semantics of MDL).
Let AP a set of atomic propositions, $p, p_1, \ldots, p_n \in AP$, and $\boldsymbol{p} = (p_1, \ldots, p_{n-1})$. The truth *of a formula $\varphi \in \mathsf{MDL}$ in a team $T \subseteq S$ of an AP-Kripke structure $W = (S, R, \pi)$ is denoted by $W, T \models \varphi$ and is defined as follows:*

$$
\begin{array}{lll}
W,T \models \top & & \text{always holds} \\
W,T \models \bot & \text{iff} & T = \emptyset \\
W,T \models p & \text{iff} & p \in \pi(s) \text{ for all } s \in T \\
W,T \models \neg p & \text{iff} & p \notin \pi(s) \text{ for all } s \in T \\
W,T \models =(\boldsymbol{p}, p_n) & \text{iff} & \text{for all } s_1, s_2 \in T \text{ it holds that} \\
 & & \pi(s_1) \cap \{\boldsymbol{p}\} \neq \pi(s_2) \cap \{\boldsymbol{p}\} \\
 & & \text{or } \pi(s_1) \cap \{p_n\} = \pi(s_2) \cap \{p_n\} \\
W,T \models \neg =(\boldsymbol{p}, p_n) & \text{iff} & T = \emptyset \\
W,T \models \varphi \wedge \psi & \text{iff} & W,T \models \varphi \text{ and } W,T \models \psi \\
W,T \models \varphi \vee \psi & \text{iff} & \text{there are sets } T_1, T_2 \text{ with } T = T_1 \cup T_2, \\
 & & W,T_1 \models \varphi \text{ and } W,T_2 \models \psi \\
W,T \models \Box \varphi & \text{iff} & W, \{s' \mid \exists s \in T \text{ with } (s,s') \in R\} \models \varphi \\
W,T \models \Diamond \varphi & \text{iff} & \text{there is a set } T' \subseteq S \text{ such that } W,T' \models \varphi \text{ and for} \\
 & & \text{all } s \in T \text{ there is an } s' \in T' \text{ with } (s,s') \in R
\end{array}
$$

Note that this semantics is a conservative extension of plain modal logic semantics, i.e., it coincides with the latter for formulae which do not contain dependence atoms. Rationales for this semantics – especially for the case of the negative dependence atom – were given by Väänänen [9, p. 24].

In Sections 3 and 4 we will classify the complexity of the model checking problem for fragments of MDL and MDL_k.

Definition 6 (MDL**-**MC**).**
Let $M \subseteq \{\Box, \Diamond, \wedge, \vee, \neg, =\}$. Then the model checking problem for $\mathsf{MDL}(M)$ ($\mathsf{MDL}_\mathsf{k}(M)$ resp.) over Kripke structures is defined as the canonical decision problem of the set

$$
\begin{array}{c}\text{MDL-MC}(M) \\ (\text{MDL}_k\text{-MC}(M) \text{ resp.})\end{array} := \left\{ \langle W, T, \varphi \rangle \;\middle|\; \begin{array}{l} W = (S, R, \pi) \text{ a Kripke structure,} \\ T \subseteq S,\ \varphi \in \mathsf{MDL}(M)\ (\mathsf{MDL}_\mathsf{k}(M) \\ \text{resp.) and } W, T \models \varphi \end{array} \right\}.
$$

We write MDL-MC *for* MDL-MC$(\{\Box, \Diamond, \wedge, \vee, \neg, =\})$.

3 Unbounded Arity Fragments

First we will show that the most general of our problems is in *NP* and therefore all model checking problems investigated later are as well.

Proposition 1. *Let M be an arbitrary set of* MDL *operators. Then* MDL-MC(M) *is in NP. And hence also* MDL_k-MC(M) *is in NP for every $k \geq 0$.*

Proof. A straightforward non-deterministic top-down algorithm can check the truth of the formula φ on the Kripke structure W in the evaluation set T in polynomial time.

The algorithm manipulates the team T according to the outermost operator in φ and then proceeds recursively. Non-deterministic steps only occur when evaluating subformulas of type $\Diamond\psi$ (a successor team is guessed) or of type $\psi \vee \theta$ (a partition of the current team is guessed).

Now we will see that the model checking problem is *NP*-hard and that this still holds without modalities.

Theorem 1. *Let $M \supseteq \{\wedge, \vee, =\}$. Then* MDL-MC($M$) *is NP-complete. Furthermore,* MDL_k-MC(M) *is NP-complete for every $k \geq 0$.*

Proof. Membership in *NP* follows from Proposition 1. For the hardness proof we reduce from 3SAT.

For this purpose let $\varphi = C_1 \wedge \ldots \wedge C_m$ be an arbitrary 3CNF formula with variables $x_1, \ldots, x_n$. Let W be the Kripke structure (S, R, π) over the atomic propositions $r_1, \ldots, r_n, p_1, \ldots, p_n$ defined by

$$\begin{aligned} S &:= \{s_1, \ldots, s_m\}, \\ R &:= \emptyset, \\ \pi(s_i) \cap \{r_j, p_j\} &:= \begin{cases} \{r_j, p_j\} & \text{iff } x_j \text{ occurs in } C_i \text{ positively,} \\ \{r_j\} & \text{iff } x_j \text{ occurs in } C_i \text{ negatively,} \\ \emptyset & \text{iff } x_j \text{ does not occur in } C_i. \end{cases} \end{aligned}$$

Let ψ be the $\mathsf{MDL}(\wedge, \vee, =)$ formula

$$\bigvee_{j=1}^{n} r_j \wedge {=}(p_j)$$

and let $T := \{s_1, \ldots, s_m\}$ the evaluation set.

It holds that $\varphi \in$ 3SAT iff $W, T \models \psi$. Hence, 3SAT $\leq_m^p$ MDL_o-MC(M).

Instead of not having modalities at all we can also allow nothing but the $\Diamond$ modality, i.e., we disallow propositional connectives and the $\Box$ modality, and model checking is *NP*-complete as well.

Theorem 2. *Let $M \supseteq \{\Diamond, =\}$. Then* MDL-MC($M$) *is NP-complete.*

Proof. Membership in *NP* follows from Proposition 1 again.

For hardness we again reduce from 3SAT. Let $\varphi = \bigwedge_{i=1}^{m} C_i$ be an arbitrary 3CNF formula built from the variables $x_1, \ldots, x_n$. Let W be the Kripke structure (S, R, π), over the atomic propositions $p_1, \ldots, p_n, q$, shown in Figure 1 and formally defined by

$$\begin{aligned} S &:= \{c_1, \ldots, c_m, s_1^1, \ldots, s_n^1, s_1^0, \ldots, s_n^0\}, \\ R \cap \{(c_i, s_j^1), (c_i, s_j^0)\} &:= \begin{cases} \{(c_i, s_j^1)\} & \text{iff } x_j \text{ occurs in } C_i \text{ positively,} \\ \{(c_i, s_j^0)\} & \text{iff } x_j \text{ occurs in } C_i \text{ negatively,} \\ \emptyset & \text{iff } x_j \text{ does not occur in } C_i, \end{cases} \\ \pi(c_i) &:= \emptyset, \\ \pi(s_j^1) &:= \{p_j, q\}, \\ \pi(s_j^0) &:= \{p_j\}. \end{aligned}$$

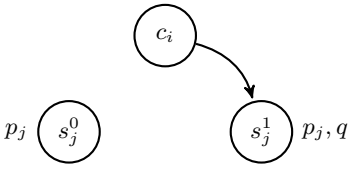


Fig. 1. Kripke structure part corresponding to the 3CNF fragment $\dots \wedge C_i \wedge \dots$ with $C_i = x_j \vee \dots$.

Let $\psi := \Diamond =(p_1, \dots, p_n, q)$ and $T := \{c_1, \dots, c_m\}$.

It now holds that $\varphi \in$ 3SAT iff $W, T \models \psi$. Hence, 3SAT $\leq_{\mathrm{m}}^{\mathrm{p}}$ MDL-MC(M) and MDL-MC(M) is *NP*-hard.

If we disallow $\Diamond$ but allow $\Box$ instead we have to also allow $\vee$ to get *NP*-hardness.

Theorem 3. *Let* $M \supseteq \{\Box, \vee, =\}$. *Then* MDL-MC($M$) *is NP-complete. Also,* MDL$_k$-MC(M) *is NP-complete for every* $k \geq 0$.

Proof. Membership in *NP* follows from Proposition 1 again. To prove hardness, we will once again reduce 3SAT to this problem.

Let $\varphi = \bigwedge_{i=1}^{m} C_i$ be an arbitrary 3CNF formula over the variables $x_1, \dots, x_n$. Let W be the structure (S, R, π), over the atomic propositions $p_1, \dots, p_n$, defined as follows:

$$\begin{aligned} S \quad &:= \{s_i | i \in \{1, \dots, m\}\} \\ &\quad \cup \{r_k^j | k \in \{1, \dots, m\}, j \in \{1, \dots, n\}\} \\ &\quad \cup \{\overline{r}_k^j | k \in \{1, \dots, m\}, j \in \{1, \dots, n\}\} \end{aligned}$$

$$R \cap \bigcup_{j \in \{1,\dots,n\}} \{(s_i, r_i^j), (s_i, \overline{r}_i^j)\} := \begin{cases} \{(s_i, r_i^1)\} & \text{iff } x_1 \text{ occurs in } C_i \text{ (positively or negatively)} \\ \{(s_i, r_i^1), (s_i, \overline{r}_i^1)\} & \text{iff } x_1 \text{ does not occur in } C_i \end{cases}$$

$$R \cap \bigcup_{k \in \{1,\dots,n\}} \{(r_i^j, r_i^k), (r_i^j, \overline{r}_i^k), (\overline{r}_i^j, r_i^k), (\overline{r}_i^j, \overline{r}_i^k)\} := \begin{cases} \{(r_i^j, r_i^{j+1})\} & \text{iff } x_j \text{ and } x_{j+1} \text{ both occur in } C_i \\ \{(r_i^j, r_i^{j+1}), (r_i^j, \overline{r}_i^{j+1})\} & \text{iff } x_j \text{ occurs in } C_i \text{ but } x_{j+1} \text{ does not occur in } C_i \\ \{(r_i^j, r_i^{j+1}), (\overline{r}_i^j, r_i^{j+1})\} & \text{iff } x_j \text{ does not occur in } C_i \text{ but } x_{j+1} \text{ does occur in } C_i \\ \{(r_i^j, r_i^{j+1}), (\overline{r}_i^j, \overline{r}_i^{j+1})\} & \text{iff neither } x_j \text{ nor } x_{j+1} \text{ occur in } C_i \end{cases}$$

$$\begin{aligned} \pi(s_i) &:= \emptyset \\ \pi(r_i^j) &:= \begin{cases} \{p_j\} & \text{iff } x_j \text{ occurs in } C_i \text{ positively or not at all} \\ \emptyset & \text{iff } x_j \text{ occurs in } C_i \text{ negatively} \end{cases} \\ \pi(\overline{r}_i^j) &:= \emptyset \end{aligned}$$

Let ψ be the $\mathsf{MDL}(\Box, \vee, =)$ formula

$$\bigvee_{j=1}^{n} \Box^j =(p_j)$$

and let $T := \{s_1, \ldots, s_m\}$.

Then $\varphi \in$ 3SAT iff $W, T \models \psi$ and therefore $\mathrm{MDL_o}$-MC$(\Box, \vee, =)$ is *NP*-complete. Intuitively, the direction from left to right holds because the disjunction splits the team $\{s_1, \ldots, s_m\}$ of all starting points of chains of length n into n subsets (one for each variable) in the following way: s_i is in the subset that belongs to x_j iff x_j satisfies the clause C_i under the variable valuation that satisfies φ. Then the team that belongs to x_j collectively satisfies the disjunct $\Box^j =(p_j)$ of ψ. For the reverse direction the $\overline{r}_i^j$ states are needed to ensure that a state s_i can only satisfy a disjunct $\Box^j =(p_j)$ if there is a variable x_j that occurs in clause C_i (positively or negatively) and satisfies C_i.

If we disallow both $\Diamond$ and $\vee$ the problem becomes tractable since the non-deterministic steps in the model checking algorithm are no longer used.

Theorem 4. *Let* $M \subseteq \{\Box, \wedge, \neg, =\}$. *Then* MDL-MC$(M)$ *is in P.*

The *NP* model checking algorithm becomes a *P* algorithm if we leave out the only nondeterministic steps for $\vee$ and $\Diamond$.

Note that this deterministic polynomial time algorithm is a top-down algorithm and therefore works in a fundamentally different way than the usual deterministic polynomial time bottom-up algorithm for plain modal logic.

Now we have seen that MDL-MC(M) is tractable if $\vee \notin M$ and $\Diamond \notin M$ since these two operators are the only source of non-determinism. On the other hand, MDL-MC(M) is *NP*-complete if $=(\cdot) \in M$ and either $\Diamond \in M$ (Theorem 2) or $\vee, \Box \in M$ (Theorem 3). The remaining question is what happens if only $\vee$ (but not $\Box$) is allowed. Unfortunately this case remains open for now.

4 Bounded Arity Fragments

We will now show that MDL-MC$(\{\vee, \neg, =\})$ is in *P* if we impose the following constraint on the dependence atoms in formulae given as part of problem instances: there is a constant $k \in \mathbb{N}$ such that in any input formula it holds for all dependence atoms of the form $=(p_1, \ldots, p_j, p)$ that $j \leq k$. To prove this statement we will decompose it into two smaller propositions.

First we show that even the whole $\{\vee, \neg, =\}$ fragment with unrestricted $=(\cdot)$ atoms is in *P* as long as it is guaranteed that in every input formula at least a specific number of dependence atoms – depending on the size of the Kripke structure – occur.

We will first formalize a notion of "many dependence atoms in a formula".

Definition 7. *For $\varphi \in \mathsf{MDL}$ let $\sigma(\varphi)$ be the number of positive dependence atoms in φ (where different occurences of the same atom are counted more than once). Let $\ell : \mathbb{N} \to \mathbb{R}$ an arbitrary function and $\star \in \{<, \leq, >, \geq, =\}$. Then* MDL-MC$_{\star\,\ell(n)}(M)$ *(*MDL$_k$-MC$_{\star\ell(n)}(M)$ *resp.) is the problem* MDL-MC(M) *(*MDL$_k$-MC(M) *resp.) restricted to inputs $\langle W = (S, R, \pi), T, \varphi\rangle$ that satisfy the condition $\sigma(\varphi) \star \ell(|S|)$.*

Note that, e.g., $\sigma(=(p) \vee =(p)) = 2$. The rationale for this is that $=(p) \vee =(p)$ is not equivalent to $=(p)$.

If we only allow $\vee$ and we are guaranteed that there are many dependence atoms in each input formula then model checking becomes trivial – even for the case of unbounded dependence atoms.

Proposition 2. *Let $M \subseteq \{\vee, \neg, =\}$. Then* MDL-MC$_{>\log_2(n)}(M)$ *is trivial, i.e., for all Kripke structures $W = (S, R, \pi)$ and all $\varphi \in \mathsf{MDL}(M)$ such that the number of positive dependence atoms in φ is greater than $\log_2(|S|)$ it holds for all $T \subseteq S$ that $W, T \models \varphi$.*

Note that MDL-MC$_{>\log_2(n)}(M)$ is only trivial, i.e., all instance structures satisfy all instance formulae, if we assume that only valid instances, i.e., where the number of dependence atoms is guaranteed to be large enough, are given as input. However, if we have to verify this number the problem clearly remains in P.

Now we consider the case in which we have very few dependence atoms (which have bounded arity) in each formula. We use the fact that there are only a few dependence atoms by searching through all possible determining functions for the dependence atoms. Note that in this case we do not need to restrict the set of allowed MDL operators as we have done above.

Proposition 3. *Let $k \geq 0$. Then* MDL$_k$-MC$_{\leq\log_2(n)}$ *is in P.*

The idea of the algorithm is to guess for every dependence atom by which determining function it is to be interpreted. Then bottom-up model checking as for plain modal logic is carried out. The guessing can be done deterministically in polynomial time because the number of dependence atoms is only logarithmic and their arity is bounded.

With Proposition 2 and Proposition 3 we have shown the following theorem.

Theorem 5. *Let $M \subseteq \{\vee, \neg, =\}$, $k \geq 0$. Then* MDL$_k$-MC(M) *is in P.*

Proof. Given a Kripke structure $W = (S, R, \pi)$ and a MDL$_k(\vee, \neg, =)$ formula φ the algorithm counts the number m of dependence atoms in φ. If $m > \log_2(|S|)$ the input is accepted (because by Proposition 2 the formula is always fulfilled in this case). Otherwise the algorithm from the proof of Proposition 3 is used.

And there is another case where we can use the exhaustive determining function search.

Theorem 6. *Let* $M \subseteq \{\Box, \Diamond, \neg, =\}$. *Then* MDL_k-MC(M) *is in* P *for every* $k \geq 0$.

Proof. Let $\varphi \in \mathsf{MDL}_k(M)$. Then there can be at most one dependence atom in φ because M only contains unary operators. Therefore we can once again use the algorithm from the proof of Proposition 3.

In Theorem 2 we saw that MDL-MC$(\Diamond, =)$ is *NP*-complete. The previous theorem includes MDL_k-MC$(\Diamond, =) \in P$ as a special case. Hence, the question remains which are the minimal supersets M of $\{\Diamond, =\}$ such that MDL_k-MC(M) is *NP*-complete.

We will now see that in the case of $k \geq 1$ adding either $\wedge$ (Theorem 7) or $\vee$ (Theorem 8) is already enough to get *NP*-completeness again. For the case of $k = 0$ the question remains open.

Theorem 7. *Let* $M \supseteq \{\Diamond, \wedge, =\}$. *Then* MDL_k-MC(M) *is NP-complete for every* $k \geq 1$.

Proof. Membership in *NP* follows from Proposition 1. For hardness we once again reduce 3SAT to our problem.

For this purpose let $\varphi := \bigwedge_{i=1}^{m} C_i$ be an arbitrary 3CNF formula built from the variables $x_1, \ldots, x_n$. Let W be the Kripke structure (S, R, π) formally defined by

$$
\begin{aligned}
S &:= \{c_i \mid i \in \{1, \ldots, m\}\} \cup \{s_{j,j'}, \overline{s}_{j,j'} \mid j, j' \in \{1, \ldots, n\}\} \\
R &:= \{(c_i, s_{1,j}) \mid x_j \in C_i\} \cup \{(c_i, \overline{s}_{1,j}) \mid \overline{x}_j \in C_i\} \\
&\quad \cup \{(s_{k,j}, s_{k+1,j}) \mid j \in \{1, \ldots, n\}, k \in \{1, \ldots, n-1\}\} \\
\pi(c_i) &:= \emptyset \\
\pi(s_{j,j'}) &:= \begin{cases} \{p_j, q\} \text{ iff } j = j' \\ \emptyset \quad \text{else} \end{cases} \\
\pi(\overline{s}_{j,j'}) &:= \begin{cases} \{p_j\} \text{ iff } j = j' \\ \emptyset \quad \text{else} \end{cases} .
\end{aligned}
$$

And let ψ be the $\mathsf{MDL}(\Diamond, \wedge, =)$ formula

$$
\begin{aligned}
&\Diamond \left(\bigwedge_{j=1}^{n} \Diamond^{j-1} {=}(p_j, q) \right) \\
\equiv\ &\Diamond \big({=}(p_1, q) \wedge \Diamond {=}(p_2, q) \wedge \Diamond\Diamond {=}(p_3, q) \wedge \ldots \wedge \Diamond^{n-1} {=}(p_n, q) \big).
\end{aligned}
$$

Then it holds that $\varphi \in$ 3SAT iff $W, \{c_1, \ldots, c_m\} \models \psi$.

Theorem 8. *Let* $M \supseteq \{\Diamond, \vee, =\}$. *Then* MDL_k-MC(M) *is NP-complete for every* $k \geq 1$.

Proof. As above membership in *NP* follows from Proposition 1 and for hardness we reduce 3SAT to our problem.

For this purpose let $\varphi := \bigwedge_{i=1}^{m} C_i$ be an arbitrary 3CNF formula built from the variables $p_1, \ldots, p_n$. Let W be the Kripke structure (S, R, π) formally defined

by

$$
\begin{aligned}
S &:= \{c_{i,j} \mid i \in \{1,\dots,m\}, j \in \{1,\dots,n\}\} \\
&\quad \cup \{x_{j,j'} \mid j, j' \in \{1,\dots,n\}, j' \leq j\} \\
R &:= \{(c_{i,j}, c_{i,j+1}) \mid i \in \{1,\dots,m\}, j \in \{1,\dots,n-1\}\} \\
&\quad \cup \{(x_{j,j'}, x_{j,j'+1}) \mid j \in \{1,\dots,n\}, j' \in \{1,\dots,j-1\}\} \\
\pi(x_{j,j'}) &:= \begin{cases} \{q, p_j\} \text{ iff } j' = j \\ \{q\} \quad \text{iff } j' < j \end{cases} \\
\pi(c_{i,j}) &:= \begin{cases} \{q\} \quad \text{iff } p_j, \neg p_j \notin C_i \\ \{p_j\} \text{ iff } p_j \in C_i \\ \emptyset \quad \text{iff } \neg p_j \in C_i \end{cases} .
\end{aligned}
$$

Let ψ be the MDL formula

$$
\begin{aligned}
&\bigvee_{j=1}^{n} \Diamond^{j-1} {=}(q, p_j) \\
\equiv\ &{=}(q, p_1) \vee \Diamond {=}(q, p_2) \vee \Diamond\Diamond {=}(q, p_3) \vee \dots \vee \Diamond^{n-1} {=}(q, p_n).
\end{aligned}
$$

Once again it can be shown that $\varphi \in$ 3SAT iff $W, \{c_{1,1}, \dots, c_{m,1}, x_{1,1}, x_{2,1}, \dots, x_{n,1}\} \models \psi$.

5 Conclusion

In this paper we showed that MDL-MC is *NP*-complete (Theorem 1). Furthermore we have systematically analyzed the complexity of model checking for fragments of MDL defined by restricting the set of modal and propositional operators. It turned out that there are several fragments which stay *NP*-complete, e.g., the fragment obtained by restricting the set of operators to only $\Box, \vee$ and $=$ (Theorem 3) or only $\Diamond$ and $=$ (Theorem 2). Intuitively, in the former case the *NP*-hardness arises from existentially guessing partitions of teams while evaluating disjunctions and in the latter from existentially guessing successor teams while evaluating $\Diamond$ operators. Consequently, if we allow all operators except $\Diamond$ and $\vee$ the complexity drops to *P* (Theorem 4).

For the fragment only containing $\vee$ and $=$ on the other hand we were not able to determine whether its model checking problem is tractable. Our inability to prove either *NP*-hardness or containment in *P* led us to restrict the arity of the dependence atoms. For the aforementioned fragment the complexity drops to *P* in the case of bounded arity (Theorem 8). Furthermore, some of the cases which are known to be *NP*-complete for the unbounded case drop to *P* in the bounded arity case as well (Theorem 6) while others remain *NP*-complete but require a new proof technique (Theorems 7 and 8). Most noteworthy in this context are probably the results concerning the $\Diamond$ operator. With unbounded dependence atoms this operator alone suffices to get *NP*-completeness whereas with bounded dependence atoms it needs the additional expressiveness of either $\wedge$ or $\vee$ to get *NP*-hardness.

Interestingly, in none of our reductions to show *NP*-hardness the MDL formula depends on anything else but the number of propositional variables of the input

3CNF formula. The structure of the input formula is always encoded by the Kripke structure alone. So it seems that even for a fixed formula the model checking problem could still be hard. This, however, cannot be the case since, by Theorem 3, model checking for a fixed formula is always in P.

Further open questions, apart from the unclassified unbounded arity case, are related to two cases with bounded arity dependence atoms. In Theorems 7 and 8 it was only possible to prove NP-hardness for arity at least one and it is not known what happens in the case where the arity is zero. Additionally, it might be interesting to determine the exact complexity for the cases which are in P since we have not shown any lower bounds in these cases so far.

References

1. Beyersdorff, O., Meier, A., Mundhenk, M., Schneider, T., Thomas, M., Vollmer, H.: Model checking CTL is almost always inherently sequential. Logical Methods in Computer Science (2011), http://arxiv.org/abs/1103.4990v1
2. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. 8(2), 244–263 (1986)
3. Ebbing, J., Lohmann, P.: Complexity of model checking for modal dependence logic. CoRR abs/1104.1034v1 (2011), http://arxiv.org/abs/1104.1034v1
4. Hemaspaandra, E.: The complexity of poor man's logic. CoRR cs.LO/9911014v2 (2005), http://arxiv.org/abs/cs/9911014v2
5. Hemaspaandra, E., Schnoor, H., Schnoor, I.: Generalized modal satisfiability. J. Comput. Syst. Sci. 76(7), 561–578 (2010)
6. Lewis, H.: Satisfiability problems for propositional calculi. Mathematical Systems Theory 13, 45–53 (1979)
7. Lohmann, P., Vollmer, H.: Complexity Results for Modal Dependence Logic. In: Dawar, A., Veith, H. (eds.) CSL 2010. LNCS, vol. 6247, pp. 411–425. Springer, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-15205-4_32
8. Sevenster, M.: Model-theoretic and computational properties of modal dependence logic. Journal of Logic and Computation 19(6), 1157–1173 (2009), http://logcom.oxfordjournals.org/cgi/content/abstract/exn102v1
9. Väänänen, J.: Dependence logic: A new approach to independence friendly logic. London Mathematical Society Student Texts, vol. 70. Cambridge University Press (2007)
10. Väänänen, J.: Modal dependence logic. In: Apt, K.R., van Rooij, R. (eds.) New Perspectives on Games and Interaction, Texts in Logic and Games, vol. 4, pp. 237–254. Amsterdam University Press (2008)

Multitape NFA: Weak Synchronization of the Input Heads

Ömer Eğecioğlu[1], Oscar H. Ibarra[1,⋆], and Nicholas Q. Tran[2]

[1] Department of Computer Science
University of California, Santa Barbara, CA 93106, USA
{omer,ibarra}@cs.ucsb.edu
[2] Department of Mathematics & Computer Science
Santa Clara University, Santa Clara, CA 95053, USA
ntran@math.scu.edu

Abstract. Given an n-tape nondeterministic finite automaton (NFA) M with a one-way read-only head per tape and a right end marker \$ on each tape, and a nonnegative integer k, we say that M is weakly k-synchronized if for every n-tuple $x = (x_1, \ldots, x_n)$ that is accepted, there is a computation on x such that at any time during the computation, no pair of input heads, neither of which is on \$, are more than k cells apart. As usual, an n-tuple $x = (x_1, \ldots, x_n)$ is accepted if M eventually reaches the configuration where all n heads are on \$ in an accepting state. We show decidable and undecidable results concerning questions such as: (1) Given M, is it weakly k-synchronized for some k (resp., for a specified k) and (2) Given M, is there a weakly k-synchronized M' for some k (resp., for a specified k) such that $L(M') = L(M)$? Most of our results are the strongest possible in the sense that slight restrictions on the models make the undecidable problems decidable. A few questions remain open.

Keywords: multitape NFA, weakly synchronized, (un)decidability.

1 Introduction

Motivated by applications to verification problems in string manipulating program (see, e.g., [5, 6, 7] for discussions on the need to validate input strings to avoid security vulnerabilities such as SQL injection attack), we look at the problem of whether the input heads in a multitape nondeterministic finite automaton (NFA) are *weakly k-synchronized*, i.e., for each accepted input there is an accepting computation where no pair of inputs heads, neither of which is on \$, are more than k tape cells apart at any time.

In a recent paper [2], we studied a different notion of head synchronization: an n-tape NFA M is *strongly k-synchronized* if at any time during any computation on *any* input n-tuple $(x_1, \ldots, x_n)$ (accepted or not), no pair of input heads,

⋆ Supported in part by NSF Grants CCF-1143892 and CCF-1117708.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 238–250, 2012.

neither of which is on \$, are more than k tape cells apart. In [2], we showed the following (among other things):

(**) It is decidable to determine, given an n-tape NFA M, whether it is k-synchronized for some k, and if this is the case, the smallest such k can be found.

Strong synchronization (studied in [2]) is a more restrictive requirement than what we investigate in this paper. Obviously, a strongly synchronized machine is also weakly synchronized, but the converse is not true. Consider, e.g., the set $L = \{(a^m\$, b^n\$) \mid m, n > 0\}$. We can construct a 2-tape NFA M, which when given input $(a^m\$, b^n\$)$, nondeterministically executes (1) or (2) below:

1. M reads $a^m\$$ on tape 1 until head 1 reaches \$, and then reads $b^n\$$ on tape 2 until head 2 reaches \$ and then accepts.
2. M reads the symbols on the two tapes simultaneously until one head reaches \$. Then the other head scans the remaining symbols on its tape and accepts.

Then M is not strongly synchronized, because of (1). However, M is weakly synchronized (in fact, weakly 0-synchronized) because every tuple $(a^m\$, b^n\$)$ can be accepted in a computation as described in (2). Thus strongly synchronized implies weakly synchronized, but not conversely.

It turns out that questions concerning weak synchronization are harder to answer than those for strong synchronization. Moreover, these two cases give some contrasting results. For example we show that, unlike (**) above, it is undecidable to determine, given a 2-ambiguous 2-tape NFA, whether it is weakly k-synchronized. However, the problem is decidable if M is 1-ambiguous, i.e., unambiguous. (A machine is k-ambiguous if there are at most k accepting computations for any input. Note that deterministic is a special case of 1-ambiguous.)

Note: Some proofs are omitted due to lack of space. All proofs will be given in a full version of the paper.

2 Preliminaries

An n-tape NFA M is a finite automaton with n tapes where each tape contains a string over input alphabet Σ. Each tape is read-only and has an associated one-way input head. We assume that each tape has a right end marker \$ (not in Σ). On a given n-tuple input $x = (x_1, \ldots, x_n)$, M starts in initial state q_0 with all the heads on the first symbols of their respective tapes. The transition function δ of M with state set Q is a mapping from $Q \times (\Sigma \cup \{\$\})^n \to 2^{Q \times \{0,1\}^n}$. If M is in state q with head H_i on symbol a_i and $(p, d_1, \ldots, d_n)$ is in $\delta(q, a_i, \ldots, a_n)$, then the machine moves H_i in direction d_i which is 1 or 0 (for right move or stationary move), and enters state p. When a head reaches the end marker \$, that head has to remain on the end marker. The input x is accepted if M eventually reaches the configuration where all n heads are on \$ in an accepting state.

Let M be an n-tape NFA and $k \geq 0$. M is *weakly k-synchronized* if for every n-tuple $x = (x_1, \ldots, x_n)$ that is accepted, there is a computation on x such that

at any time during the computation, no pair of input heads, neither of which is on \$, are more than k cells apart. Notice that, since the condition in the definition concerns pairs of heads that are both on symbols in Σ, if one of these two heads is on \$, then we can stipulate that the condition is automatically satisfied, irrespective of the distance between the heads. In particular, if $k = 0$, then all heads move to the right synchronously at the same time (except for heads that reach the right end marker early). M is *weakly synchronized* if it is weakly k-synchronized for some k.

An n-tape NFA that is deterministic is called an n-tape DFA. An n-tape NFA (DFA) can be augmented with a finite number of reversal-bounded counters. At each step, each counter (which is initially set to zero and can never become negative) can be incremented by 1, decremented by 1, or left unchanged and can be tested for zero. The counters are reversal-bounded in the sense that there is a specified r such that during any computation, no counter can change mode from increasing to decreasing and vice-versa more than r times. A counter is 1-reversal if once it decrements, it can no longer increment. Clearly, an r-reversal counter can be simulated by $\lceil (r+1)/2 \rceil$ 1-reversal counters.

Given an n-tuple $(x_1, \ldots, x_n)$, denote by $\langle x_1, \ldots, x_n \rangle$ an n-track string where the symbols of x_i's are left-justified (i.e., the symbols are aligned) and the shorter strings are right-filled with blanks (λ) to make all tracks the same length. For example, $\langle 01, 1111, 101 \rangle$ has $01\lambda\lambda$ on the upper track, 1111 on the middle track, and 101λ on the lower track. Given a set L of n-tuples, define $\langle L \rangle = \{\langle x \rangle \mid x \in L\}$.

Lemma 1. *Let L a set of n-tuples. Then L is accepted by a weakly 0-synchronized n-tape NFA if and only if $\langle L \rangle$ is regular.*

Let $\mathbb{N}$ be the set of nonnegative integers and k be a positive integer. A subset Q of $\mathbb{N}^k$ is a *linear set* if there exist vectors $v_0, v_1, \ldots, v_n$ in $\mathbb{N}^k$ such that $Q = \{v_0 + t_1 v_1 + \cdots + t_n v_n \mid t_1, \ldots, t_n \in \mathbb{N}\}$. The vectors$v_0$ (referred to as the *constant vector*) and $v_1, \ldots, v_n$ (referred to as the *periods*) are called the *generators* of the linear set Q. The set $Q \subseteq N^k$ is *semilinear* if it is a finite union of linear sets. The empty set is a trivial (semi)linear set, where the set of generators is empty. Every finite subset of $\mathbb{N}^k$ is semilinear – it is a finite union of linear sets whose generators are constant vectors. Semilinear sets are closed under (finite) union, complementation and intersection. It is known that the disjointness, containment, and equivalence problems for semilinear sets are decidable [3].

Let $\Sigma = \{a_1, \ldots, a_k\}$. For $w \in \Sigma^*$, let $|w|$ be the number of letters in w, and $|w|_{a_i}$ denote the number of occurrences of a_i in w. The *Parikh image* $P(w)$ of w is the vector $(|w|_{a_1}, \ldots, |w|_{a_k})$; the Parikh image of a language L is defined as $P(L) = \{P(w) \mid w \in L\}$.

We will need the following result from [1]:

Theorem 1. *The emptiness (Is $L(M) = \varnothing$?) and infiniteness (Is $L(M)$ infinite?) problems for 1-tape NFA with reversal-bounded counters are decidable.*

Corollary 1. *The emptiness and infiniteness problems for n-tape NFA with reversal-bounded counters are decidable.*

An instance $I = (u_1, \ldots, u_n); (v_1, \ldots, v_n)$ of the *Post Correspondence Problem* (PCP) is a pair of n-tuples of nonnull strings over an alphabet with at least two symbols. A solution to I is a sequence of indices $i_1, i_2, \ldots, i_m$ such that $u_{i_1} \ldots u_{i_m} = v_{i_1} \ldots v_{i_m}$. It is well known that it is undecidable to determine, given a PCP instance I, whether it has a solution.

Convention: (1) We shall also refer to a set of n-tuples accepted by an n-tape machine as a language. (2) All input n-tuples $(x_1, \ldots, x_n)$ are delimited by a right end marker \$ on each tape, although sometimes the end markers are not explicitly shown. (3) A construction is *effective* if it can be implemented as an algorithm.

3 2-Ambiguous Multitape NFA

In this section, we will show that it is undecidable to determine, given a 2-ambiguous 2-tape NFA M and an integer k, whether M is weakly synchronized, whether M is weakly k-synchronized for a given k, and whether there is a weakly synchronized M' such that $L(M) = L(M')$.

We first prove this result for general 2-tape NFA and then show how to modify the proof so that it applies to the restricted case of 2-ambiguous 2-tape NFA.

Let $I = (u_1, \ldots, u_n); (v_1, \ldots, v_n)$ be an instance of the PCP. Let c and d be new symbols. We construct a 2-tape NFA M to accept the language

$$L = \{(xc^i, yd^j) \mid i, j > 0, x \neq y\} \cup \{(xc^i, xd^j) \mid i, j > 0, j = 2i, x \text{ is a solution of the PCP instance } I\}$$

as follows: M on input (xc^i, yd^j) nondeterministically selects to check (a) or (b) below:

(a) M first checks that $x \neq y$ by moving both heads in sync (0-synchronized) until it finds the first position where x differs from y (note that this also takes care of the case when their lengths are different). When M finds a discrepancy, both heads are moved to the right in sync until one head reaches the end marker and then the other head is moved to the right until it reaches the end marker. Then M accepts. Note that the whole process is deterministic.

(b) M guesses a sequence of indices $i_1, i_2, \ldots$. After guessing an index i_j, M verifies that u_{i_j} is on tape 1, and then v_{i_j} is on tape 2. This process is repeated until both x and y are exhausted. If there is no discrepancy and the heads reach the first c and d on their tapes, the second head moves right twice for every right move of the first head to check that $j = 2i$, and then M accepts.

There are six possible outcomes from a computation of M:

1. (xc^i, yd^j), where $x \neq y$ and (a) is selected: there is a 0-synchronized accepting computation of M on this input.
2. (xc^i, yd^j), where $x \neq y$, (b) is selected, and either the selected indices yield a discrepancy or $j \neq 2i$: this input will be rejected in (b).

3. (xc^i, yd^j), where $x \neq y$, (b) is selected, and the selected indices do not yield a discrepancy and $j = 2i$: this input will be accepted in (b). Note that this input will also be accepted in (a).
4. (xc^i, xd^j), and (a) is selected: this input will be rejected.
5. (xc^i, xd^j), (b) is selected, and the selected indices yield a discrepancy or $j \neq 2i$: this input will be rejected.
6. (xc^i, xd^j), (b) is selected, and the selected indices do not yield a discrepancy and $j = 2i$: this input will be accepted.

We observe the following:

Note 1: (i) From (1) and (3), it is possible that the same input of the form (xc^i, yd^j), where $x \neq y$ can be accepted in both processes (a) or (b). (ii) Another source of ambiguity is in process (b) itself – different sequences of (guessed) indices $i_1, i_2, ...$ may yield no discrepancies when matching x and y, so the same input may be accepted in many ways in (b).

Note 2: An input of the form (xc^i, xd^j), where the selected indices did not yield a discrepancy (i.e., the PCP has a solution) and $j = 2i$, is only accepted in (b) (it is rejected in (a)). Furthermore, since i is arbitrary, the heads will be out of sync unboundedly.

Hence, if the PCP instance I does not have a solution, then (xc^i, yd^j) is accepted iff $x \neq y$, so there is a 0-synchronized accepting computation (type (a)). In other words, M is weakly 0-synchronized and hence $\langle L(M) \rangle$ is regular by Lemma 1.

On the other hand, if PCP instance I has a solution, then $L(M)$ contains tuples of the form (xc^i, xd^{2i}). We claim that $L(M)$ cannot be accepted by *any* weakly
0-synchronized 2-tape NFA. If it is, then by Lemma 1, $\langle L(M) \rangle$ is regular. But then for large enough i, we can pump the string $\langle xc^i, xd^{2i} \rangle$ to get a string $\langle xc^{i+k}, xd^{i+k}d^i \rangle$ for some $k > 0$ to be in $\langle L(M) \rangle$. But $(xc^{i+k}, xd^{i+k}d^i)$ is not in $L(M)$, a contradiction.

To summarize, PCP instance I has a solution iff there is *no* weakly 0-synchronized 2-tape NFA M' such that $L(M) = L(M')$. Furthermore, the construction above shows that either M is weakly 0-synchronized or it is not weakly k-synchronized for any k. Together these results yield

Theorem 2. *The following problems are undecidable, given a 2-tape (and hence multitape) NFA M:*

1. *Is M weakly k-synchronized for a given k?*
2. *Is M weakly k-synchronized for some k?*
3. *Is there a 2-tape (multitape) NFA M' that is weakly 0-synchronized (or weakly k-synchronized for a given k, or weakly k-synchronized for some k) such that $L(M') = L(M)$?*

We now modify the construction of the 2-tape NFA M above to make it 2-ambiguous. The sources of ambiguity are cited in Notes 1 and 2 above. Clearly,

since process (a) is deterministic, if we can make process (b) deterministic, then the 2-NFA will be 2-ambiguous.

We accomplish this as follows. Instead of x, we use x' where x' has two tracks: track 1 contains x and track 2 contains the "encoding" of the indices that are used to match x and y; y remains single-track. Specifically, let $I = (u_1, \ldots, u_n); (v_1, \ldots, v_n)$ be an instance of the PCP. Let $\#, e_1, \ldots, e_n$ be new symbols. For $1 \leq i \leq n$, let the string $E(i) = e_i \#^{|u_i|-1}$. Thus, the length of $E(i)$ is equal to the length of u_i. Let $\Delta = \{\#, e_1, \ldots, e_n\}$, and define the language:

$L = \{(x'c^i, yd^j) \mid i, j > 0, x'$ is a 2-track tape where the first track contains x and the second track is a string in Δ^*, $x \neq y\} \cup \{x'c^i, yd^j) \mid i, j > 0, x'$ is a 2-track tape where the first track contains x and the second track is a string $E(i_1) \cdots E(i_r)$ for some $i_1, \ldots, i_r$, $x = y$, $x = u_{i_1} \cdots u_{i_r}, y = v_{i_1} \cdots v_{i_r}, j = 2i\}$.

One can easily check that processes (a) and (b) described earlier can be made deterministic. However, it is possible that the same input of the form $(x'c^i, yd^j)$, where $x \neq y$ can be accepted in both processes (a) or (b). (x is the first track of x'.) Hence, M is 2-ambiguous. Therefore we have:

Theorem 3. *The following problems are undecidable, given a 2-ambiguous 2-tape (and hence multitape) NFA M:*

1. *Is M weakly k-synchronized for a given k?*
2. *Is M weakly k-synchronized for some k?*
3. *Is there a 2-tape (multitape) NFA M' that is weakly 0-synchronized (or weakly k-synchronized for a given k, or weakly k-synchronized for some k) such that $L(M') = L(M)$?*

4 Unambiguous Multitape NFA

In this section, we show that given an unambiguous multitape NFA M and an integer k, it is decidable to determine whether M is weakly synchronized, whether M is weakly k-synchronized for a given k, and whether $L(M) = L(M')$ for some weak synchronized multitape unambiguous NFA M'. Recall that unambiguous multitape NFA have at most one accepting computation for every input n-tuple. Note that multitape DFAs are a special case. The results in this section are modifications of the corresponding results for synchronized multitape automata in [2], and we omit their proofs.

Theorem 4. *It is decidable to determine, given an unambiguous n-tape NFA M, whether it is weakly k-synchronized for some k.*

Corollary 2. *It is decidable to determine, given an unambiguous n-tape NFA M and an integer $k \geq 0$, whether M is k-synchronized.*

The following follows from the previous two results.

Corollary 3. *It is decidable to determine, given an unambiguous n-tape NFA M, whether it is weakly k-synchronized for some k. Moreover, if it is, we can effectively determine the smallest such k.*

The above results generalize to machines with reversal-bounded counters:

Theorem 5. *It is decidable to determine, given an unambiguous n-tape NFA M augmented with reversal-bounded counters, whether it is weakly k-synchronized for some k. Moreover, if it is, we can effectively determine the smallest such k.*

5 Multitape NFA on ABO-Bounded Inputs

A language is *bounded* if it is a subset of $a_1^* \cdots a_n^*$ for some distinct letters (symbols) $a_1, \ldots, a_n$. A multitape NFA is *unary* if each tape contains a string over a single symbol (letter); *bounded* if each tape contains a string from a bounded language; and *all-but-one-bounded (ABO-bounded)* if all but the first tape contains a string from a bounded language. We also refer to the inputs of such machines as unary, bounded, ABO-bounded, respectively.

In section 3, we showed that it is undecidable to determine, given a 2-tape NFA M and an integer $k \geq 0$, whether M is weakly k-synchronized. Here we show that the problem is decidable for n-tape NFA when the inputs are ABO-bounded. In fact, the result holds for n-tape NFA over $\Sigma^* \times x_{21}^* \cdots x_{2m_2}^* \times \cdots \times x_{n1}^* \cdots x_{nm_n}^*$ for some (not necessarily distinct) nonnull strings x_{ij}.

Note that if L is a set of n-tuples of strings, then $\overline{L}$ (the complement of L) is the set of n-tuples $(x_1, \ldots, x_n)$ such that $(x_1, \ldots, x_n)$ is in $\overline{L}$ if and only if $(x_1, \ldots, x_n)$ is not in L.

An n-tape NFA is *strictly k-synchronized* if in *any accepting* computation, any pair of the heads are within k cells apart (when neither head is on \$). In comparison, this condition must hold for only some accepting computation in weakly k-synchronized machines and for any computation in strongly k-synchronized machines.

Lemma 2. *Let M be an n-tape NFA that is strictly k-synchronized that accepts the language (set of n-tuples) $L = L(M)$. Then:*

1. *We can effectively construct a strictly 0-synchronized n-tape DFA M_1 accepting L.*
2. *We can effectively construct a strictly 0-synchronized n-tape DFA M_2 accepting $\overline{L}$.*

Moreover, (1) holds for n-tape weakly k-synchronized NFA as well.

Proof. For the first part, given M, we construct an ordinary (i.e., 1-tape) NFA A_1 such that $(x_1, \ldots, x_n)$ is accepted by M if and only if (the aligned version) $\langle x_1, \ldots, x_n \rangle$ is accepted by A_1. This is possible as A_1 need only maintain a finite buffer of symbols in its state. A_1 can then be converted to be deterministic (by the usual subset construction), i.e., we can convert A_1 to an equivalent DFA A_2.

Then from A_2, we can trivially construct a strictly 0-synchronized n-tape DFA M_1 accepting L.

For part 2, we can easily construct from the DFA A_2 a DFA A_3 accepting $\langle x_1, \ldots, x_n \rangle$ if and only if A_2 does not accept $\langle x_1, \ldots, x_n \rangle$. Let $L' = \{\langle x_1, \ldots, x_n \rangle \mid x_1, \ldots, x_n$ are strings with no λ's $\}$. Clearly, L' is regular and is accepted by some DFA A_4. Construct a DFA A_5 accepting $L(A_3) \cap L(A_4)$. (The reason for the intersection is to make sure that we only retain the well-formed aligned strings.) From A_5, we can then construct a strictly 0-synchronized 2-tape DFA M_2 accepting $\overline{L}$. □

We are now ready to prove the main result of this section. To illustrate the construction, we consider 3-tape NFA.

Theorem 6. *It is decidable to determine, given a 3-tape NFA M that accepts a subset of $\Sigma^* \times a^* \times b^*$ (where Σ is any alphabet and a, b are symbols) and a nonnegative integer k, whether M is weakly k-synchronized.*

Proof. Construct from M a 3-tape NFA M_1 over $\Sigma^* \times a^* \times b^*$ that is strictly k-synchronized: on input (x_1, x_2, x_3), M_1 simulates the computation of M faithfully and accepts (x_1, x_2, x_3) if M accepts (x_1, x_2, x_3) and during the computation, the heads are always within k cells apart (provided no head has reached \$).

From Lemma 2, part 2, we can effectively construct from M_1 a strictly 0-synchronized 3-tape DFA M_2 accepting $\overline{L(M_1)}$.

Clearly, M is weakly k-synchronized if and only if $L(M) \subseteq L(M_1)$ (in fact $L(M) = L(M_1)$), hence, if and only if $L(M) \cap \overline{L(M_1)} = \varnothing$, i.e., $L(M) \cap L(M_2) = \varnothing$.

To decide the above, we construct from M and M_2, a 3-tape NFA M' with four 1-reversal counters C_1, C_2, D_1, D_2 that works as follows when given input (x, a^r, b^s):

M' reads a^r and b^s and stores r in counters C_1 and D_1 and s in counters C_2 and D_2. Then M' simulates M on (x, a^r, b^s) by using counters C_1 and C_2, i,e., it decreases C_1 (resp., C_2) by 1 every time the second head (resp., third head) of M moves right on a^r (resp., b^s). At the same time, M' also simulates M_2 on (x, a^r, a^s) using counters D_1 and D_2. Note that since M_2 is 0-synchronized, D_1 (resp., D_2) is decreased only when M moves its first head to the right on x. M'accepts if M and M_2 accept.

It follows that M is weakly k-synchronized if and only if $L(M')$ is empty, which is decidable by Corollary 1. □

Corollary 4. *It is decidable to determine, given a 3-tape NFA M over $\Sigma^* \times a_1^* \cdots a_r^* \times b_1^* \cdots b_s^*$ for distinct symbols $a_1, \ldots, a_r$, $b_1, \ldots, b_s$ and a nonnegative integer k, whether M is weakly k-synchronized.*

In fact, we can prove a stronger result:

Corollary 5. *It is decidable to determine, given a 3-tape NFA M over $\Sigma^* \times v_1^* \cdots v_r^* \times w_1^* \cdots w_s^*$ for (not necessarily distinct) nonnull strings $v_1, \cdots, v_r$, $w_1, \ldots, w_s$ and a nonnegative integer k, whether M is weakly k-synchronized.*

The above corollary generalizes to multitape NFA:

Corollary 6. *It is decidable to determine, given an n-tape NFA M over $\Sigma^* \times x_{21}^* \cdots x_{2m_2}^* \times \cdots \times x_{n1}^* \cdots x_{nm_n}^*$ for some (not necessarily distinct) nonnull strings x_{ij}'s and a nonnegative integer k, whether M is weakly k-synchronized.*

6 Multitape NFA on Unary Inputs

In this section, we look at decision problems for multitape NFA on unary inputs.

6.1 Synchronizability

In Theorem 3, we saw that if the inputs of the 2-tape NFA M is unrestricted, it is undecidable to determine if there exists a weakly 0-synchronized 2-tape NFA M' equivalent to M. But what about the case when one tape has bounded input, or when both tapes have bounded inputs? At present, we do not know the answer. However, as the following shows, even for the unary case, there are machines that cannot be converted to be weakly 0-synchronized:

Proposition 1. *$L = \{(a^m, b^n) \mid m > 0, n = 2m\}$ can be accepted by a 2-tape NFA M but cannot be accepted by a weakly 0-synchronized 2-tape NFA.*

6.2 Weakly Synchronized Regular Languages

First we consider the extension of the definition of strongly k-synchronized and weakly k-synchronized to *languages* (instead of *machines* in the original definitions) over a binary alphabet $\Sigma = \{a, b\}$ and show that whether or not a regular language is weakly synchronized is decidable. We do so via a structural characterization of weakly synchronized regular languages, which is of independent interest.

A word w is *strongly k-synchronized* if for any factorization $x = uv$

$$-k \leq |u|_a - |u|_b \leq k. \tag{1}$$

A language L over Σ is *strongly k-synchronized* if all of its words are strongly k-synchronized, and *strongly synchronized* if it is strongly k-synchronized for some k. L is called *weakly k-synchronized* if for every $w \in L$, there is a corresponding $w' \in L$ such that $P(w') = P(w)$ and w' is strongly k-synchronized. L is *weakly synchronized* if it is weakly k-synchronized for some k. Suppose M is a DFA over $\Sigma = \{a, b\}$. Consider an r-cycle C in M given by the sequence of states

$$q_1, q_2, \ldots, q_{r+1} \tag{2}$$

with $r \geq 2$ and $q_1 = q_{r+1}$. The word associated to C is $a_1 a_2 \cdots a_r$ where $\delta(q_i, a_i) = q_{i+1} \quad (i = 1, 2, \ldots, r)$. When there is no ambiguity, we use C to also denote the associated word $a_1 a_2 \ldots a_r$. We call C *balanced* if $|C|_a = |C|_b$. C is *a-heavy* if $|C|_a > |C|_b$, and *b-heavy* if $|C|_a < |C|_b$. C is a *simple* cycle if $q_i \neq q_j$ $(i, j = 1, 2, \ldots, r)$ in (2).

Lemma 3. *Suppose M is a DFA with m states with no unbalanced simple cycles. If C is a cycle in M then the word C is strongly m-synchronized.*

Theorem 7. *L is weakly synchronized iff the minimum state DFA M for L has no unbalanced simple cycles.*

Proof. Suppose M has an unbalanced simple r-cycle C. We will show that L is not weakly synchronized. WLOG C is a-heavy. By the minimality of M, the beginning state q_1 of C is reachable from the initial state of M, and there is a path from $q_{r+1} = q_1$ to a final state of M. It follows that we can pump C: i.e. there exists words x, y such that for every $t \geq 0$, $w_t = xC^t y \in L$. Then $P(w_t) = (c_0 + c_1 t, d_0 + d_1 t)$ for constants $c_0, c_1, d_0, d_1 \geq 0$ with $c_1 > d_1$. Therefore for any word w'_t with $P(w_t) = P(w'_t)$, taking $u = w'_k$, $|u|_a - |u|_b = c_0 - d_0 + (c_1 - d_1)t \to \infty$ as $t \to \infty$, so (1) cannot hold for any fixed k. Therefore L is not weakly synchronized. Conversely, assume that every simple cycle in M is balanced. We will show that L is strongly synchronized, and therefore weakly synchronized. Let m be the number of states of M. We show that L is strongly $2m$-synchronized. Any w accepted by M can be written as $w = x_0 C_1 x_1 C_2 \dots C_t x_t$ where each C_i is a balanced cycle (not necessarily simple) and $|x_0 x_1 \cdots x_t| < m$. By lemma 3, each cycle is strongly m-synchronized. Since the contribution of the part $x_0 x_1 \cdots x_t$ to the difference of the number of occurrences of a's and b's in w is at most m, any prefix u of w satisfies (1) with $k = 2m$. □

Corollary 7. *A regular language L is strongly synchronized iff it is weakly synchronized.*

Corollary 8. *It is decidable whether a regular language L is weakly synchronized.*

Finally, we can state the condition for the weak synchronizability of a regular language L in terms of its Parikh image. Consider a linear set that appears in $P(L)$: $\{(a_0, b_0) + k_1(a_1, b_1) + \cdots + k_s(a_s, b_s) \mid k_1, k_2, \dots, k_s \geq 0\}$. Each of the vectors $(a_i, b_i), i = 1, \dots, s$ are called *periods*. A period (a_i, b_i) is balanced iff $a_i = b_i$. If $a_i > b_i$, then the period is a-heavy, if $b_i > a_i$ then it is b-heavy. These notions are translations of the ones on cycles to the Parikh images where a balanced period corresponds to a balanced simple cycle, etc. Therefore

Corollary 9. *For a regular language L over $\Sigma = \{a, b\}$, the following are equivalent:*

1. *L is strongly synchronized,*
2. *L is weakly synchronized,*
3. *The minimum state DFA for L has no unbalanced simple cycles,*
4. *The Parikh image $P(L)$ has no unbalanced periods.*

6.3 Weakly Synchronized NFA on Unary Inputs

In Section 3, we showed that it is undecidable to determine, given a 2-tape NFA M, whether it is weakly k-synchronized for some k. The problem is also

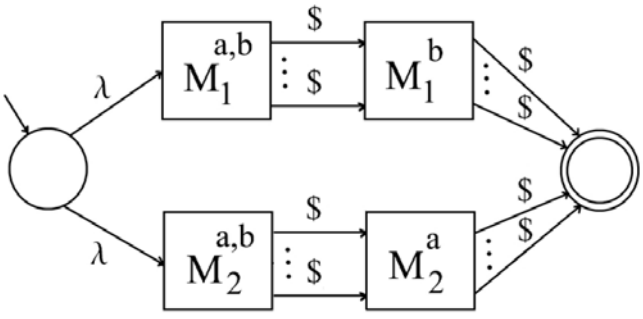


Fig. 1. An ordinary NFA that accepts accepting computations of a 2-tape NFA

undecidable when k is specified, even for $k = 0$. In Section 5, we showed that it is decidable to determine, given a 2-tape NFA M one of whose tapes contains a string over a bounded language and an integer k, whether it is weakly k-synchronized. We currently do not know if this restriction would make determining whether a 2-tape NFA M is weakly k-synchronized for some unspecified k decidable. It appears to be a difficult combinatorial problem. However, for the special case when the two tapes are unary, we can show that the problem is decidable.

Let M be a 2-tape NFA where the inputs are of the form $(a^n\$, b^m\$)$. We assume that exactly one of the heads moves during the computation of M.

The transitions of M can be labeled with letters from $\Sigma = \{a, b, \$\}$. A head movement on the first (second) tape is labeled with a (b). The last move by each head is labeled \$. In this way each computation path of M can be identified with a word over Σ. Each such word contains exactly two occurrences of the symbol \$. We will not consider these when we look at the Parikh vectors of the words accepted by M. Since M is nondeterministic, there may be many accepting computation paths w for an accepted input, but each of these has Parikh vector $P(w) = (a^n, b^m)$.

M can be trivially modified into an ordinary NFA M' that accepts the accepting computations of M as shown in Figure 1.

There are two types of accepting computations:

1. $w = x\$y\$$ with $x \in \{a, b\}^*$ and $y \in \{b\}^*$,
2. $w = x\$y\$$ with $x \in \{a, b\}^*$ and $y \in \{a\}^*$.

The first type comes from accepting computations in which the first head reaches \$ first, and then the computation continues on tape two with the second head moving to the right consuming b's until it reaches \$ on the second tape. The second type is similar, with the roles of the two tapes interchanged. Consequently, whether M is weakly k-synchronized is equivalent to showing that for any given word $w \in L(M')$, there is a strongly k-synchronized word u such that either $ub^j \in L(M')$ and $P(w) = P(u)+(0, j)$, or $ua^i \in L(M')$ and $P(w) = P(u)+(i, 0)$.

Theorem 8. *Suppose M is a unary 2-tape NFA. Then it is decidable whether or not M is weakly synchronized.*

Proof. We can assume that exactly one of the heads moves one cell to the right in each step until the heads reach \$. Let $L = L(M)$. We can assume that the automata that appear in the boxes in Figure 1 are minimum state DFA. The language accepted by the upper part of Figure 1 can be written as the disjoint union of languages accepted by pairs of DFA, corresponding to final states of $M_1^{a,b}$. If $M_1^{a,b}$ has t final states, then this results in t pairs of minimum state DFA $(F_i^{a,b}, N_i^b)$. The corresponding language accepted is the concatenation $L(F_i^{a,b})L(N_i^b)$. There is a similar decomposition for the language accepted by the lower part of Figure 1. For any given i, consider $P_i = P(F_i^{a,b})$. Eliminating each unbalanced period from each of the linear sets in P_i results in a new Parikh image P_i'. This is the Parikh image of the automaton obtained from $F_i^{a,b}$ by eliminating unbalanced cycles as follows.

Consider a simple unbalanced r-cycle C of $F_i^{a,b}$ given by $q_1, q_2, \ldots, q_{r+1}$ with $r \geq 2$ and $q_1 = q_{r+1}$. Let $a_1 \cdots a_r \in \Sigma^*$ be the word associated where $\delta(q_i, a_i) = q_{i+1}$ $(i = 1, 2, \ldots, r)$ Consider the new alphabet $\Sigma' = \Sigma \cup \{x_1, x_2, \ldots, x_r\}$. Alter the labels on C by replacing a_i $(a_i \in \{a, b\})$ by the symbol x_i. Denote by L_{01} the language of all words over Σ' taking the start state of $F_i^{a,b}$ to q_1, and by L_{1f} the language of all words over Σ' taking $q_{r+1} = q_1$ to the final state of $F_i^{a,b}$. If h is the homomorphism defined by $h(x_i) = a_i$, $(i = 1, \ldots, r)$, then the language

$$h\left(L(F_i^{a,b}) \setminus L_{01}x_1x_2\cdots x_rL_{1f}\right)$$

is the language of words in $L(F_i^{a,b})$ which are accepted without traversing the cycle C. Since there are finitely many simple cycles in $F_i^{a,b}$, in finitely many steps we can take away from $L(F_i^{a,b})$ all of the words that traverse an unbalanced cycle of $F_i^{a,b}$. This has the effect of deleting all of the unbalanced periods from the linear sets that appear in the Parikh map.

Let P_i' be the Parikh image of the resulting automaton, and let P^u be the union of the sets $P_i' + P(N_i^b)$ over i. For the lower part of the diagram, a similar construction results in the set P^l. By the characterization of weak synchronization preceding the statement of Theorem 8, M is weakly synchronized iff $P(L) \subseteq P^u \cup P^l$. Each of the Parikh images above can effectively be constructed and the containment required can be effectively checked, since Parikh images of regular languages are semilinear. □

7 Conclusion

We looked at decision questions concerning weak synchronization of the heads of multitape NFA. Most of our results are the strongest possible in the sense that slight restrictions on the models make the undecidable problems decidable.

Some questions remain open. In particular, is it decidable to determine, given a 2-tape NFA whose tapes are over bounded languages, whether it is weakly k-synchronized for some k ? This question seems quite difficult – we have only been able to resolve this question (in the positive) for the bounded unary case.

References

1. Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. J. Assoc. Comput. Mach. 25, 116–133 (1978)
2. Ibarra, O.H., Tran, N.: On synchronized multitape and multihead automata. In: Proc. of the 13th Int. Workshop on Descriptional Complexity of Formal Systems (DCFS 2011), pp. 184–197 (2011)
3. Ginsburg, G., Spanier, E.: Bounded Algol-like languages. Trans. of the Amer. Math. Society 113, 333–368 (1964)
4. Parikh, R.J.: On context-free languages. J. Assoc. Comput. Mach. 13, 570–581 (1966)
5. Yu, F., Bultan, T., Cova, M., Ibarra, O.H.: Symbolic String Verification: An Automata-Based Approach. In: Havelund, K., Majumdar, R. (eds.) SPIN 2008. LNCS, vol. 5156, pp. 306–324. Springer, Heidelberg (2008)
6. Yu, F., Bultan, T., Ibarra, O.H.: Symbolic String Verification: Combining String Analysis and Size Analysis. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 322–336. Springer, Heidelberg (2009)
7. Yu, F., Bultan, T., Ibarra, O.H.: Relational String Verification Using Multi-track Automata. In: Domaratzki, M., Salomaa, K. (eds.) CIAA 2010. LNCS, vol. 6482, pp. 290–299. Springer, Heidelberg (2011)

Visibly Pushdown Transducers with Look-Ahead*

Emmanuel Filiot[1] and Frédéric Servais[2]

[1] Université Libre de Bruxelles
[2] Hasselt University and Transnational University of Limburg

Abstract. Visibly Pushdown Transducers (VPT) form a subclass of pushdown transducers. In this paper, we investigate the extension of VPT with visibly pushdown look-ahead ($\mathsf{VPT}_{\mathsf{la}}$). Their transitions are guarded by visibly pushdown automata that can check whether the well-nested subword starting at the current position belongs to the language they define. First, we show that $\mathsf{VPT}_{\mathsf{la}}$ are not more expressive than VPT, but are exponentially more succinct. Second, we show that the class of deterministic $\mathsf{VPT}_{\mathsf{la}}$ corresponds exactly to the class of functional VPT, yielding a simple characterization of functional VPT. Finally, we show that while $\mathsf{VPT}_{\mathsf{la}}$ are exponentially more succinct than VPT, checking equivalence of functional $\mathsf{VPT}_{\mathsf{la}}$ is, as for VPT, EXPT-C. As a consequence, we show that any functional VPT is equivalent to an unambiguous one.

1 Introduction

Visibly pushdown transducers (VPT) [17,9] form an interesting subclass of pushdown transducers (PT). Several problems that are undecidable for PT are decidable for VPT, noticeably: functionality is decidable in PTIME, k-valuedness in NPTIME and equivalence of functional VPT is EXPT-C [9].

Visibly pushdown machines [1], automata (VPA) or transducers, are pushdown machines such that the behavior of the stack, i.e. whether it pushes or pops, is visible in the input word. Technically, the input alphabet is partitioned into call, return and internal symbols. When reading a call the machine must push a symbol on the stack, when reading a return symbol it must pop and when reading an internal symbol it cannot touch the stack. The partitioning of the input alphabet induces a nesting structure of the input words [2]. A call symbol delimits an additional level of nesting, while a return symbol is a position in the word that ends a level of nesting. A word is well-nested if each call, respectively each return, has a matching return, respectively a matching call. Visibly pushdown transductions are transductions that can be defined by VPT.

Unranked trees in their linear form (such as XML documents) can be viewed as well-nested words. VPT are therefore a suitable formalism for unranked tree transformations. In particular, they can express operations such as node deletion, renaming and

* This research was supported by the projects: Gasics: "Games for Analysis and Synthesis of Interactive Computational Systems", http://www.ulb.ac.be/di/gasics/, and Moves: "Fundamental Issues in Modelling, Verification and Evolution of Software", http://moves.ulb.ac.be, a PAI program funded by the Federal Belgian Government. Partially funded by the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, No. FP7-ICT-233599.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 251–263, 2012.

insertion. Furthermore, over well-nested words, a simple and expressive subclass of VPT, the class of well-nested VPT [9], is closed under composition and has a decidable type checking problem. In the setting of XML documents, VPA, as they read the tree in a left-to-right depth-first traversal manner, are well-suited for streaming validation [11,15] or streaming XML queries [10]. In the same way well-nested VPT are amenable to define streaming transformations.

In this paper, one of our motivations is to give a simple characterization of functional VPT that can be checked easily. Deterministic VPT are not expressive enough to capture all functional VPT, as for instance swapping the first and last letters of a word cannot be done deterministically. Instead of non-determinism, we show that some limited inspection of the longest well-nested subword starting at the current position (called the *current well-nested prefix*) is required to capture (non-deterministic) functional VPT. More precisely, we show that functional VPT-transductions are captured by deterministic VPT extended with visibly pushdown look-aheads that inspect the current well-nested prefix. Moreover, inspecting the current well-nested prefix is somehow the minimal necessary information to capture all functional VPT.

In this paper, we therefore introduce and investigate the class of VPT with visibly pushdown look-ahead. A VPT with visibly pushdown look-ahead ($\mathsf{VPT}_{\mathsf{la}}$) is a VPT such that call transitions are guarded with visibly pushdown automata (VPA). When reading a call at position i, a $\mathsf{VPT}_{\mathsf{la}}$ can apply a call transition provided the longest well-nested word starting at position i is included in the language of the VPA of the transition. In the same way one can define VPA with look-ahead ($\mathsf{VPA}_{\mathsf{la}}$). Our main contributions are the following:

1. $\mathsf{VPT}_{\mathsf{la}}$ *(resp.* $\mathsf{VPA}_{\mathsf{la}}$*) are as expressive as* VPT *(resp.* VPA*), but exponentially more succinct.*

For this we present an exponential construction that shows how a VPT can simulate look-aheads. Moreover we show this exponential blow-up is unavoidable.

2. *Deterministic* $\mathsf{VPT}_{\mathsf{la}}$ *and functional* VPT *are equally expressive*.

This equivalence is obtained by a construction (which is also exponential) that replaces the non-determinism of the functional VPT with deterministic look-ahead. This also yields a simple characterization of functional VPT.

3. *Functional* VPT *and unambiguous* VPT *are equally expressive*.

As an application of look-aheads, we show that a nice consequence of the constructions involved in contributions 1 and 2 is that functional VPT are effectively characterized by unambiguous VPT. This result was already known for finite-state transducers [4,14,5] and here we extend it to VPT with rather simple constructions based on the concept of look-aheads. This characterization of functional finite-state transducers has been generalized to k-valued and k-ambiguous finite-state transducers [18] and recently with a better upper-bound [13] based on lexicographic decomposition of transducers.

4. *Equivalence of functional* $\mathsf{VPT}_{\mathsf{la}}$ *(resp* $\mathsf{VPA}_{\mathsf{la}}$*) is, as for* VPT *(resp* VPA*),* EXPT-C.

Therefore even though $\mathsf{VPT}_{\mathsf{la}}$ are exponentially more succinct than VPT, testing equivalence of functional $\mathsf{VPT}_{\mathsf{la}}$ is not harder than for functional VPT. This is done in two steps. First one checks equivalence of the domains. Then one checks that the union of the two transducers is still functional. We show that testing functionality is EXPT-C for $\mathsf{VPT}_{\mathsf{la}}$: get rid of the look-aheads with an exponential blow-up and test in PTIME

Table 1. Decision Problems for VPA,$\mathsf{VPA}_{\mathsf{la}}$,VPT,$\mathsf{VPT}_{\mathsf{la}}$

	VPA [1]	$\mathsf{VPA}_{\mathsf{la}}$	VPT [9]	$\mathsf{VPT}_{\mathsf{la}}$
Emptiness	PTIME	EXPT-C	PTIME	EXPT-C
Universality	EXPT-C	EXPT-C	*NA*	*NA*
Inclusion	EXPT-C	EXPT-C	EXPT-C	EXPT-C
Equivalence	EXPT-C	EXPT-C	EXPT-C (for fVPT)	EXPT-C (for fVPT)
Functionality	*NA*	*NA*	PTIME	EXPT-C

the functionality of the constructed VPT. To verify that the domains are equivalent, the naive technique (removing the look-aheads and then verifying the mutual inclusion of the domains) yields a doubly exponential algorithm. Instead, we show that the domains of $\mathsf{VPT}_{\mathsf{la}}$ are linearly reducible to alternating top-down tree automata. Testing the equivalence of such automata can be done in EXPT [3].

Table 1 summarizes the complexity of decision problems for $\mathsf{VPA}_{\mathsf{la}}$ and $\mathsf{VPT}_{\mathsf{la}}$.

Variants of look-ahead. We discuss in [16] some variants of look-ahead. The closure by look-ahead (Theorem 1) and the equivalence between deterministic $\mathsf{VPT}_{\mathsf{la}}$ and functional VPT (Theorem 2) still hold when the look-ahead can inspect the whole suffix and can also be triggered on return transitions. However, when the look-ahead can inspect only the current well-nested prefix of the form cwr (corresponding to the first subtree of the current hedge in a tree), it is not sufficient to express all functional VPT with determinism.

Related Works. Regular look-aheads have been mainly considered for classes of tree transducers, where a transition can be fired provided the current subtree belongs to some regular tree language. For instance, regular look-aheads have been added to *top-down (ranked) tree transducers* in order to obtain a robust class of tree transducers that enjoys good closure properties wrt composition [6], or to *macro tree transducers* (MTT) [8]. For top-down tree transducers, adding regular look-ahead strictly increases their expressive power while MTT are closed by regular look-ahead [8]. Another strong result shows that every functional top-down tree transduction can be defined by a *deterministic* top-down tree transducer with look-ahead [7].

Trees over an alphabet Σ can be linearized as well-nested words over the structured alphabet $\Sigma_c = \{c_a \mid a \in \Sigma\}$, $\Sigma_r = \{r_a \mid a \in \Sigma\}$. It is well-known that unranked trees can be represented by binary trees via the classical first-child next-sibling encoding (fcns). Top-down (ranked) tree transducers can thus be used as unranked tree transducers on fcns encodings of unranked trees. Inspecting a subtree in the fcns encoding corresponds to inspecting the first subtree and its next-sibling subtrees in an unranked tree, which in turn corresponds to inspecting the current longest well-nested prefix in their linearization. However top-down tree transducers and VPT are incomparable: top-down tree transducers can copy subtrees while VPT cannot, and VPT support concatenation of tree sequences while top-down tree transducers cannot. For example, the transformation that removes the g node in unranked trees of the form $f(g(a,\dots,a),b,b,\dots b)$ produces trees of the form $f(a,a,\dots,a,b,\dots b)$. This transformation can easily be defined by a VPT, but not by a top-down ranked tree transducers with the fcns encoding [12,9]. Indeed, in the fcns encoding, this transformation maps any tree of the form $f(g(t_a,t_b),\perp)$ to $f(t_a.t_b,\perp)$, where $t_a, t_b, t_a.t_b$ are the binary encodings of the hedges

$(a,\dots,a)$, $(b,\dots,b)$, $(a,\dots,a,b,\dots,b)$ respectively:

$$t_a = a(\bot, a(\bot,\dots a(\bot,\bot)\dots)) \quad t_b = b(\bot, b(\bot,\dots b(\bot,\bot)\dots))$$
$$t_a.t_b = a(\bot, a(\bot,\dots a(\bot, b(\bot, b(\bot,\dots b(\bot,\bot)\dots)))))$$

Therefore, this transformation requires to move the subtree t_b (whose size may be unbounded) as a leaf of the subtree t_a (whose size may also be unbounded). This cannot be done by a top-down tree transducer, but can be defined by some MTT thanks to parameters (some parameter will store the entire subtree t_b while evaluating t_a). A detailed comparison of VPT and tree transducers can be found in [16].

Modulo the former encodings, MTT subsume VPT [9] and as we said before, there is a correspondence between the two notions of look-aheads, for VPT and MTT respectively. However it is not clear how to derive our results on closure by look-aheads from the same result on MTT, as the latter highly relies on parameters and it would require back-and-forth encodings between the two models. The direct construction we give in this paper is self-contained and allows one to derive the characterization of functional VPT as unambiguous VPT by a careful analysis of the construction.

An extended version of the paper with all proofs can be found in [16].

2 Visibly Pushdown Languages and Transductions

All over this paper, Σ denotes a finite alphabet partitioned into two disjoint sets Σ_c, Σ_r, denoting respectively the *call* and *return* alphabets. We denote by Σ^* the set of (finite) words over Σ and by ϵ the empty word. The length of a word u is denoted by $|u|$. The set of *well-nested* words Σ^*_{wn} is the smallest subset of Σ^* such that $\epsilon \in \Sigma^*_{\mathsf{wn}}$ and for all $c \in \Sigma_c$, all $r \in \Sigma_r$, all $u, v \in \Sigma^*_{\mathsf{wn}}$, $cur \in \Sigma^*_{\mathsf{wn}}$ and $uv \in \Sigma^*_{\mathsf{wn}}$.

A *visibly pushdown automaton* (VPA) [1] on finite words over Σ is a tuple $A = (Q, I, F, \Gamma, \delta)$ where Q is a finite set of states, $I \subseteq Q$ the set of initial states, $F \subseteq Q$ the set of final states, Γ the (finite) stack alphabet, and $\delta = \delta_c \uplus \delta_r$ where $\delta_c \subseteq Q \times \Sigma_c \times \Gamma \times Q$ are the *call transitions*, $\delta_r \subseteq Q \times \Sigma_r \times \Gamma \times Q$ are the *return transitions*[1].

On a call transition $(q, a, \gamma, q') \in \delta_c$, γ is pushed onto the stack and the control goes from q to q'. On a return transition $(q, a, \gamma, q') \in \delta_r$, γ is popped from the stack.

A *configuration* of a VPA is a pair $(q, \sigma) \in Q \times \Gamma^*$. A *run* of T on a word $u = a_1 \dots a_l \in \Sigma^*$ from a configuration (q, σ) to a configuration (q', σ') is a finite sequence $\rho = \{(q_k, \sigma_k)\}_{0 \le k \le l}$ such that $q_0 = q$, $\sigma_0 = \sigma$, $q_l = q'$, $\sigma_l = \sigma'$ and for each $1 \le k \le l$, there exists $\gamma_k \in \Gamma$ such that either $(q_{k-1}, a_k, \gamma_k, q_k) \in \delta_c$ and $\sigma_k = \sigma_{k-1}\gamma_k$ or $(q_{k-1}, a_k, \gamma_k, q_k) \in \delta_r$ and $\sigma_{k-1} = \sigma_k\gamma_k$. The run ρ is *accepting* if $q_0 \in I$, $q_l \in F$ and $\sigma_0 = \sigma_l = \bot$ A word w is *accepted* by A if there exists an accepting run of A over w. $L(A)$, the *language* of A, is the set of words accepted by A. A language L over Σ is a *visibly pushdown language* if there is a VPA A over Σ such that $L(A) = L$.

As finite-state transducers extend finite-state automata with outputs, visibly pushdown transducers extend visibly pushdown automata with outputs [9]. To simplify notations, we suppose that the output alphabet is Σ, but our results still hold for an arbitrary

[1] In contrast to [1], we do not consider *internal* symbols i, as they can be simulated by a (unique) call c_i followed by a (unique) return r_i. We make this assumption to simplify proofs and notations. Moreover, we do not allow return transition on $\bot$ and we require the final stack to be empty. This implies that all accepted words are well-nested. All our results extend easily to alphabets with internal symbols and to VPT that accept by final state only.

output alphabet. Informally, the stack behavior of a VPT is similar to the stack behavior of visibly pushdown automata (VPA). On a call symbol, the VPT pushes a symbol on the stack and produces some output word (possibly empty and not necessarily well-nested), on a return symbol, it must pop the top symbol of the stack and produce some output word (possibly empty) and on an internal symbol, the stack remains unchanged and it produces some output word.

Definition 1. A *visibly pushdown transducer* (VPT) on finite words over Σ is a tuple $T = (Q, I, F, \Gamma, \delta)$ where Q is a finite set of states, $I \subseteq Q$ is the set of initial states, $F \subseteq Q$ the set of final states, Γ is the stack alphabet, $\delta = \delta_c \uplus \delta_r$ the (finite) transition relation, with $\delta_c \subseteq Q \times \Sigma_c \times \Sigma^* \times \Gamma \times Q$, $\delta_r \subseteq Q \times \Sigma_r \times \Sigma^* \times \Gamma \times Q$.

Configurations and runs are defined similarly as VPA. Given a word $u = a_1 \dots a_l \in \Sigma^*$ and a word $v \in \Sigma^*$, v is an *output* of u by T if there exists an accepting run $\rho = \{(q_k, \sigma_k)\}_{0 \leq k \leq l}$ on u and l words $v_1, \dots, v_l$ such that $v = v_1 \dots v_l$ and for all $0 \leq k < l$, there is a transition of T from (q_k, σ_k) to (q_{k+1}, σ_{k+1}) that produces the output v_{k+1} on input letter a_{k+1}. We write $(q, \sigma) \xrightarrow{u/v} (q', \sigma')$ when there exists a run on u from (q, σ) to (q', σ') producing v as output. A transducer T defines the binary word relation $[\![T]\!] = \{(u, v) \mid \exists q \in I, q' \in F,\ (q, \bot) \xrightarrow{u/v} (q', \bot)\}$.

A *transduction* is a binary relation $R \subseteq \Sigma^* \times \Sigma^*$. We say that a transduction R is a VPT-transduction if there exists a VPT T such that $R = [\![T]\!]$. A transduction R is *functional* if for all $u \in \Sigma^*$, there exists at most one $v \in \Sigma^*$ such that $(u, v) \in R$. A VPT T is *functional* if $[\![T]\!]$ is functional, and we denote by fVPT the class of functional VPT. Two transducers T_1, T_2 are *equivalent* if $[\![T_1]\!] = [\![T_2]\!]$. It is known [9] that functionality is decidable in PTIME for VPT and equivalence of functional VPT is EXPT-C. Finally, a VPT is *unambiguous* if there is at most one accepting run per input word. In particular, any unambiguous VPT is functional. Unambiguity can be checked in PTIME [9].

For any input word $u \in \Sigma^*$, we denote by $R(u)$ the set $\{v \mid (u, v) \in R\}$. Similarly, for a VPT T, we denote by $T(u)$ the set $[\![T]\!](u)$. If R is functional, we confound $R(u)$ (which is at most of cardinality 1) and the unique image of u if it exists. The *domain* of T (denoted by *Dom*(T)) is the domain of $[\![T]\!]$. Note that the domain of T contains only well-nested words, which is not necessarily the case of the codomain.

Example 1. Let $\Sigma_c = \{c, a\}, \Sigma_r = \{r\}$ be the call and return symbols of the alphabet. The following VPT T transforms a word as follows: (i) a and r are mapped to a and r respectively; (ii) c is mapped either to c if no a appears in the longest well-nested word starting at c, and to a if an a appears. E.g. $ccrrarcr$ is mapped to $acrrarcr$, and $cccrrcrcarrr$ to $aacrraraarrr$.

The VPT $T = (Q, I, F, \Gamma, \delta)$ is defined by $Q = \{q, q_a, q_{\neg a}\}$, $I = \{q\}$, $F = Q$, $\Gamma = \{\gamma, \gamma_a, \gamma_{\neg a}\}$ and δ contains the following transitions:

$$
\begin{array}{lll}
q \text{ or } q_a \xrightarrow{c/a,\gamma} q_a & q \text{ or } q_a \xrightarrow{c/a,\gamma_a} q & q \xrightarrow{c/c,\gamma_{\neg a}} q_{\neg a} \\
q \text{ or } q_a \xrightarrow{a/a,\gamma} q & q_{\neg a} \xrightarrow{c/c,\gamma_{\neg a}} q_{\neg a} & \\
q \text{ or } q_{\neg a} \xrightarrow{r/r,\gamma_a} q_a & q \text{ or } q_{\neg a} \xrightarrow{r/r,\gamma} q & q_{\neg a} \xrightarrow{r/r,\gamma_{\neg a}} q_{\neg a}
\end{array}
$$

The state q_a, resp. $q_{\neg a}$, means that there is, resp. is not, an a in the longest well-nested word that starts at the current position. The state q indicates that there is no

constraints on the appearance of a. If T is in state q and reads a c, there are two cases: it outputs an a or a c. If it chooses to output an a, then it must check that an a occurs later. There are again two cases: either T guesses there is an a in the well-nested word that starts just after c and takes the transitions $q \xrightarrow{c/a,\gamma} q_a$, or it guesses an a appears in the well-nested word that starts after the matching return of c, in that latter case it takes the transition $q \xrightarrow{c/a,\gamma_a} q$ and uses the stack symbol γ_a to carry over this information. If on c it chooses to output c, it must check that there is no a later by using the transition $q \xrightarrow{c/a,\gamma_{\neg a}} q_{\neg a}$. Other cases are similar.

3 VPT with Visibly Pushdown Look-Ahead

Given a word w over Σ we denote by $\text{pref}_{\text{wn}}(w)$ the longest well-nested prefix of w. E.g. $\text{pref}_{\text{wn}}(ccrcr) = \epsilon$ and $\text{pref}_{\text{wn}}(crc) = cr$. We define a VPT T with visibly pushdown look-ahead (simply called look-ahead in the sequel) informally as follows. The look-ahead is given by a VPA A without initial state. On a call symbol c, T can trigger the look-ahead from a state p of the VPA (which depends on the call transition). The look-ahead tests membership of the longest well-nested prefix of the current suffix (that starts by the letter c) to $L(A, p)$, where (A, p) is the VPA A with initial state p. If the prefix is in $L(A, p)$ then the transition of T can be fired. When we consider nested words that encode trees, look-aheads correspond to inspecting the subtree rooted at the current node and all right sibling subtrees (in other words, the current hedge). Formally:

Definition 2. *A* VPT *with look-ahead (*$\mathsf{VPT_{la}}$*) is a pair* $T_{la} = (T, A)$ *where* A *is a* VPA $A = (Q^{la}, F^{la}, \Gamma^{la}, \delta^{la})$ *without initial state and* T *is a tuple* $T = (Q, q_0, F, \Gamma, \delta)$ *such that* Q *is a finite set of states,* $q_0 \in Q$ *is an initial state,* $F \subseteq Q$ *is a set of final states,* Γ *is a stack alphabet, and* $\delta = \delta_c \uplus \delta_r$ *is a transition relation such that* $\delta_c \subseteq Q \times \Sigma_c \times \Sigma^* \times Q^{la} \times \Gamma \times Q$ *and* $\delta_r \subseteq Q \times \Sigma_r \times \Sigma^* \times \Gamma \times Q$.

A VPA *with look-ahead (*$\mathsf{VPA_{la}}$*) is defined similarly.*

Let $u \in \Sigma^*$. A run of T_{la} on $u = a_1 \dots a_l$ is a sequence of configurations $\rho = \{(q_k, \sigma_k)\}_{0 \leq k \leq l}$ such that, for all $k < l$, there exist $\gamma \in \Gamma$ and $v_{k+1} \in \Sigma^*$ such that (i) if $a_{k+1} \in \Sigma_r$, then $\sigma_{k+1}\gamma = \sigma_k$ and $(q_k, a_{k+1}, v_{k+1}, \gamma, q_{k+1}) \in \delta_r$; (ii) if $a_{k+1} \in \Sigma_c$, then $\sigma_{k+1} = \sigma_k\gamma$, and there exists $p \in Q^{la}$ such that $(q_k, a_{k+1}, v_{k+1}, p, \gamma, q_{k+1}) \in \delta_c$ and $\text{pref}_{\text{wn}}(a_{k+1} \dots a_l) \in L(A, p)$. The run ρ is accepting if $\sigma_0 = \sigma_l = \bot$ and $q_l \in F$. The word $v_1 \dots v_l$ is an output of u.

The $\mathsf{VPT_{la}}$ T_{la} is *deterministic* if for all transitions $(q, c, v_1, p_1, \gamma_1, q_1) \in \delta_c$ and $(q, c, v_2, p_2, \gamma_2, q_2) \in \delta_c$, if $v_1 \neq v_2$ or $\gamma_1 \neq \gamma_2$ or $q_1 \neq q_2$ or $p_1 \neq p_2$, then $L(A, p_1) \cap L(A, p_2) = \varnothing$; and for all transitions $(q, r, v_1, \gamma_1, q_1) \in \delta_r$ and $(q, r, v_2, \gamma_2, q_2) \in \delta_r$ we have $v_1 = v_2$, $\gamma_1 = \gamma_2$ and $q_1 = q_2$. Note that deciding whether some $\mathsf{VPT_{la}}$ is deterministic can be done in PTIME. One has to check that for each state q and each call symbol c, the VPL guarding transitions from state q and reading c are *pairwise* disjoint. The number of states of a $\mathsf{VPT_{la}}$ is the number of states of the transducer plus the number of states of the look-ahead.

Example 2. A $\mathsf{VPT_{la}}$ is represented in Figure 1. The look-ahead automaton is depicted on the right, while the transducer in itself is on the left. It defines the transduction

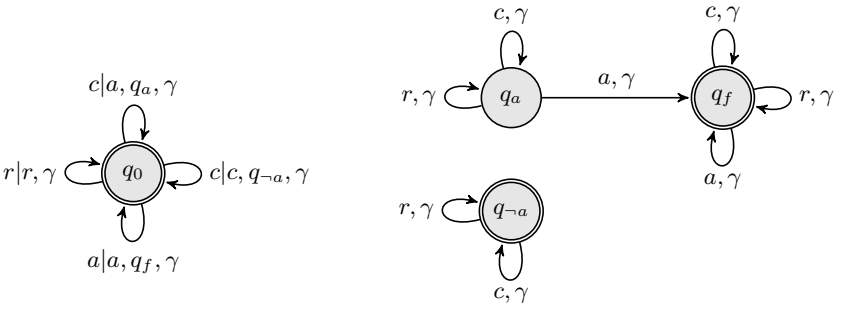


Fig. 1. A $\mathsf{VPT}_{\mathsf{la}}$ (left) and its look-ahead (right) on $\Sigma_c = \{c, a\}$ and $\Sigma_r = \{r\}$

of Example 1. When starting in state q_a, respectively $q_{\neg a}$, the look-ahead automaton accepts well-nested words that contains an a, respectively does not contain any a. When starting in state q_f it accepts any well-nested word. The transducer rewrites c symbols into a if the well-nested word starting at c contains an a (transition on the top), otherwise it just copy a c (transition on the right). This is achieved using the q_a and $q_{\neg a}$ states of the look-ahead automaton. Other input symbols, i.e. a and r, are just copied to the output (left and bottom transitions).

The next theorem states that adding look-aheads to VPT does not add expressiveness. The main difficulty is to simulate an unbounded number of look-aheads at the same time. Indeed, a look-ahead is triggered at each call and is alive until the end of the well-nested subword starting at this call. To handle the simulation of the look-aheads that started at a *strictly less deeper* nesting level we use the notion of summaries. Recall that summaries were introduced in the context of the determinization of VPA ([1]), they are pairs of states. More precisely, for a given VPA, a pair (p, q) is a summary if there exists a well-nested word w such that the configuration $(q, \bot)$ is accessible from $(p, \bot)$ by reading w. We use a classical subset construction for the look-aheads that started at the *same* nesting level.

Theorem 1. *For any* $\mathsf{VPT}_{\mathsf{la}}$*, resp.* $\mathsf{VPA}_{\mathsf{la}}$*,* T_{la} *with* n *states, one can construct an equivalent* VPT*, resp.* VPA*,* T' *with* $O(n2^{n^2+1})$ *states. Moreover, if* T_{la} *is deterministic, then* T' *is unambiguous.*

Proof. We prove the result for $\mathsf{VPT}_{\mathsf{la}}$ only, this trivially implies the result for $\mathsf{VPA}_{\mathsf{la}}$. Let $T_{la} = (T, A)$ with $T = (Q, q_0, F, \Gamma, \delta)$ and $A = (Q^{la}, F^{la}, \Gamma^{la}, \delta^{la})$. We construct $T' = (Q', q'_0, F', \Gamma', \delta')$ as follows (where $Id_{Q^{la}}$ denotes the identity relation on Q^{la}): $Q' = Q \times 2^{Q^{la} \times Q^{la}} \times 2^{Q^{la}}$, $q'_0 = (q_0, Id_{Q^{la}}, \varnothing)$, $F' = \{(q, R, L) \in Q' \mid q \in F, L \subseteq F^{la}\}$, $\Gamma' = \Gamma \times 2^{Q^{la} \times Q^{la}} \times 2^{Q^{la}} \times \Sigma_c$.

The transducer T' simulates T and its running look-aheads. A state of T' is a triple (q, R, L). The first component is the state of T. The second and third components are used to simulate the running look-aheads. When taking a call c, T' non-deterministically chooses a new look-ahead triggered by T. This look-ahead is added to all running look-aheads that started at the same nesting level. T' ensures that the run will fail if the

longest well-nested prefix starting at c is not in the language of the triggered look-ahead. The L component contains the states of all running look-aheads triggered at the current nesting level. The R component is the summary necessary to update the L-component. When reading a call the L component is put on the stack. When reading a return, T' must check that all look-ahead states in L are final, i.e. T' ensures that the chosen look-aheads are successful.

After reading a well-nested word w if T' is in state (q, R, L), with $q \in Q$, $R \subseteq Q^{la} \times Q^{la}$ and $L \subseteq Q^{la}$, we have the following properties. The pair $(p, p') \in R$ iff there exists a run of A from p to p' on w. If some p'' is in L, there exists a run of a look-ahead that started when reading a call symbol of w at depth 0 which is now in state p''. Conversely, for all look-aheads that started when reading a call symbol of w at depth 0, there exists a state $p'' \in L$ and a run of this look-ahead that is in state p''.

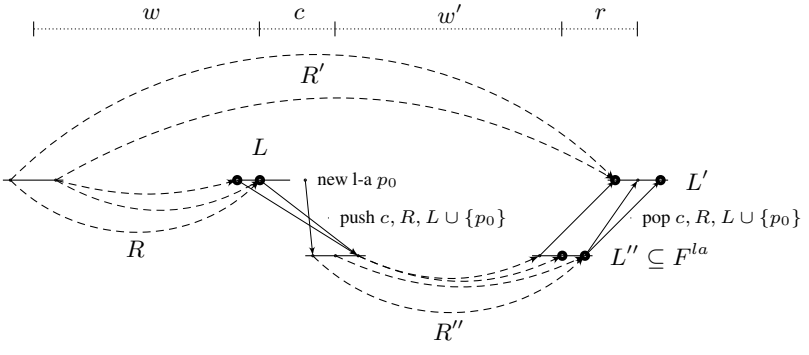


Fig. 2.

Let us consider a word $wcw'r$ for some well-nested words w, w' (depicted on Fig. 2). Assume that T' is in state (q, R, L) after reading w (on the figure, the relation R is represented by dashed arrows and the set L by big points, and other states by small points). We do not represent the T-component of the states on the figure but rather focus on R and L. The information that we push on the stack when reading c is the necessary information to compute a state (q', R', L') of T' reached after reading $wcw'r$. After reading the call symbol c, we go in state $(q', Id_{Q^{la}}, \varnothing)$ and produce the output v for some q', v such that $q \xrightarrow{c|v,p_0,\gamma} q' \in \delta_c$, where $p_0 \in Q^{la}$ is the starting state of a new look-ahead. Note that determinism of T is preserved. On the stack we put the tuple $(\gamma, R, L \cup \{p_0\}, c)$ where γ, R, L, p_0, c have been defined before.

Now, suppose that after reading wcw' the transducer T' is in state (q'', R'', L''). It means that T is in state q'' after reading wcw', and $(p, p') \in R''$ iff there exists a run of A from p to p' on w', and L'' is some set of states reached by the look-aheads that started at the same depth as w'. Therefore we first impose that any transition from (q'', R'', L'') reading r must satisfy $L'' \subseteq F^{la}$. Clearly, R' can be constructed from c, R and R''. Finally, L' is a set which satisfies for all $p \in L \cup \{p_0\}$, there exists $p' \in L'$ such that there exists a run of A from p to p' on $cw'r$. If such an L' does not exist, there is no transition on r. The set L' can be constructed from $L \cup \{p_0\}$ and R''.

We now define the transitions formally. First, for all q, R, L, c, γ, we have:

$$(q, R, L) \xrightarrow{c|u,(\gamma,R,L\cup\{p_0\},c)} (q', Id_{Q^{la}}, \varnothing) \in \delta'_c \text{ whenever } q \xrightarrow{c|u,p_0,\gamma} q' \in \delta_c$$

Then, for all $R, L, r, \gamma, q'', R'', L'', q', R', L'$ we have:

$$(q'', R'', L'') \xrightarrow{r|u,(\gamma,R,L,c)} (q', R', L') \in \delta'_r \text{ if the following conditions hold:}$$

(i) $q'' \xrightarrow{r|u,\gamma} q' \in \delta_r$, (ii) $L'' \subseteq F^{la}$
(iii) $R' = \{(p, p') \mid \exists s \xrightarrow{c,\gamma} s' \in \delta_c^{la} \cdot \exists (s', s'') \in R'' \cdot (p, s) \in R \text{ and } s'' \xrightarrow{r,\gamma} p' \in \delta_r^{la}\}$
(iv) for all $p \in L$, there exist $p' \in L'$, $\gamma \in \Gamma$, $s, s' \in Q^{la}$ such that $(s, s') \in R''$, $p \xrightarrow{c,\gamma} s \in \delta_c^{la}$, $s' \xrightarrow{r,\gamma} p' \in \delta_r^{la}$.

If T is deterministic, then T' is unambigous. Indeed, it is deterministic on return transitions. If there are two possible transitions $q \xrightarrow{c|u_1,p_1,\gamma_1} q_1$ and $q \xrightarrow{c|u_2,p_2,\gamma_2} q_2$ on a call symbol c, as T is deterministic, we know that either the look-ahead starting in p_1 or the look-ahead starting in p_2 will fail. In T', there will be two transitions that will simulate both look-aheads respectively, and therefore at least one continuation of the two transitions will fail as well. Therefore there is at most one accepting computation per input word in T. □

Succinctness. The exponential blow-up in the construction of Theorem 1 is unavoidable. Indeed, it is obviously already the case for finite state automata with regular look-ahead. These finite state automata can be easily simulated by VPA on flat words (in $(\Sigma_c\Sigma_r)^*$) (in that case the stack is useless). For example, consider for all n the language $L_n = \{vuv \mid |v| = n\}$. One can construct a finite state automaton with regular look-ahead with $O(n)$ states that recognizes L_n. It is done by using look-aheads that check for all $a \in \Sigma$ and $i \leq n$ that the $m - (n - i)$-th letter is equal to a, where m is the length of the word. Without a regular look-ahead, any automaton has to store the n-th first letters of w in its states, then it guesses the $m - n$-th position and checks that the prefix of size n is equal to the suffix of size n. A simple pumping argument shows that the automaton needs at least $|\Sigma|^n$ states.

4 Functional VPT and $\mathsf{VPT}_{\mathsf{la}}$

While there is no known syntactic restriction on VPT that captures all functional VPT, we show that the class of deterministic $\mathsf{VPT}_{\mathsf{la}}$ captures all functional VPT. As there may be an unbounded number of accepting runs, the equivalent $\mathsf{VPT}_{\mathsf{la}}$ has to choose only one of them by using look-aheads. This is done by ordering the states and extending this order to runs. Similar ideas have been used in [7] to show the same result for top-down tree transducers. The main new difficulty with VPT is to cope with nesting. Indeed, when the transducer enters an additional level of nesting, its look-ahead cannot inspect the entire suffix but is limited to the current nesting level. When reading a call, choosing (thanks to some look-ahead) the smallest run on the current well-nested prefix is not correct because it may not be possible to extend this run to an accepting run on the entire word. Therefore the transducer has to pass some information from one to the

next level of nesting about the chosen global run, while for top-down tree transducers, as the evaluation is top-down, the transformation of the current subtree is independent of the transition choices that have been made at upper levels.

Theorem 2. *For all* VPT T*, one can construct a deterministic* $\mathsf{VPT}_{\mathsf{la}}$ T_{la} *with at most exponentially many more states such that* $[\![T_{la}]\!] \subseteq [\![T]\!]$ *and* $Dom(T_{la}) = Dom(T)$*. If* T *is functional, then* $[\![T_{la}]\!] = [\![T]\!]$*.*

Proof. We order the states of T and use look-aheads to choose the smallest runs wrt to an order on runs that depends on the structure of the word. Let $T = (Q, q_0, F, \Gamma, \delta)$ be a VPT. Wlog we assume that for all $q, q' \in Q$, all $\alpha \in \Sigma$, there is at most one $u \in \Sigma^*$ and one $\gamma \in \Gamma$ such that $(q, \alpha, u, \gamma, q') \in \delta$. A transducer satisfying this property can be obtained by duplicating the states with transitions, i.e. by taking the set of states $Q \times \Delta$.

We construct a deterministic $\mathsf{VPT}_{\mathsf{la}}$ $T_{la} = (T', A)$ such that $[\![T_{la}]\!] \subseteq [\![T]\!]$ and $Dom(T_{la}) = Dom(T)$ and where $T' = (Q', q_0, F', \Gamma', \delta')$ with $Q' = \{q_0\} \cup Q^2$, $F' = F \times Q$ if $q_0 \notin F$ otherwise $F' = (F \times Q) \cup \{q_0\}$. The look-ahead A is defined later. Before defining δ' formally, let us explain it informally. There might be several accepting runs on an input word w, T_{la} has to choose exactly one. Furthermore, to ensure determinism, when reading a symbol, T_{la} has to choose exactly one transition. The idea is to order the states by a total order $<_Q$ and to extend this order to runs. The look-ahead will be used to choose the next transition of T that has to be fired, so that the choice will ensure that T follows the smallest accepting run on w. However the look-ahead can only visit the current longest well-nested prefix, and not the entire word. Therefore the "parent" of the call c has to pass some information about the global run to its child c. In particular, when T' is in state (q, q') for some state q', it means that T is in state q and the state reached after reading the last return symbol of the longest-well nested current prefix must be q'.

Consider a word of the form $w = c_1 w_1 r_1 w_2 c_3 w_3 r_3$ where w_i are well-nested, depicted on Fig. 3. Suppose that before evaluating w, T' is in state (q_1, q_3). It means that the last transition T has to fire when reading r_3 has a target state q_3. When reading the call symbol c_1, T' uses a look-ahead to determine the smallest triple of states (q'_1, q'_2, q_2) such that there exists a run on w that starts in q_1 and such that after reading c_1 it is in state q'_1, before reading r_1 it is in state q'_2, after reading r_1 it is in state q_2 and after reading r_3 it is in state q_3. Then, T' fires the call transition on c_1 that with source and target states q_1 and q'_1 respectively (it is unique by hypothesis), put on the stack the states (q_2, q_3) and passes to w_1 (in the state) the information that the chosen run on w_1 terminates by the state q'_2, i.e. it goes to the state (q'_1, q'_2). (see Fig. 3). On the figure, we do not explicit all the states and anonymous components are denoted by $\_$. When reading r_1, T' pops from the stack the tuple (γ, q_2, q_3) and therefore knows that the transition to apply on r_1 has target state q_2 and the transition to apply on r_3 has target state q_3. Then it passes q_3 to the current state.

When the computation starts in q_0, we do not know yet what return transition has to be fired at the end of the hedge. This case can be easily treated separately by a look-ahead on the first call symbol that determine the smallest 4-tuple of states (q_1, q'_2, q_2, q_3) which satisfies the conditions described before, but to simplify the proof, we assume that the VPT accepts only words of the form cwr, where w is well-nested, so that one only needs to consider triples of states.

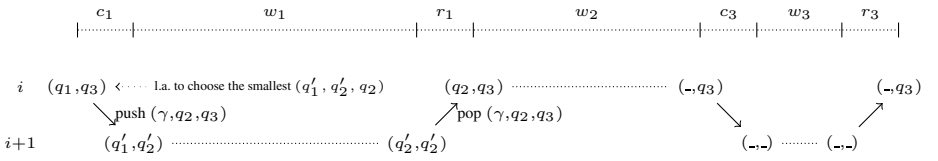


Fig. 3.

We now define the transition relation formally. Let $<$ be a total order on states, extended lexicographically to tuples. For all states $q_1, q_1', q_2', q_2, q_3 \in Q$, it is easy to define a VPA $A_{q_1,q_1',q_2',q_2,q_3}$ whose size is polynomial in the size of T that accepts a word w iff it is of the form $c_1 w_1 r_1 w_3$ where w_1, w_3 are well-nested and there exists a run of T on w that starts in state q_1 and is state q_1' after reading c_1, in state q_2' before reading r_1, in state q_2 after reading r_1 and in state q_3 after reading w_3. Note that if $w_3 = \epsilon$ then if $q_3 \neq q_2$, then $w \notin L(A_{q_1,q_1',q_2',q_2,q_3})$. We denote by $\overline{A_{q_1,q_1',q_2',q_2,q_3}}$ the complement of $A_{q_1,q_1',q_2',q_2,q_3}$.

We let $B_{q_1,q_1',q_2',q_2,q_3}$ a VPA with initial state $p_{q_1,q_1',q_2',q_2,q_3}$ that defines the language:

$$L(B_{q_1,q_1',q_2',q_2,q_3}) = L(A_{q_1,q_1',q_2',q_2,q_3}) \cap \bigcap_{\substack{(s_1,s_2',s_2) \in Q^3 \\ (s_1,s_2',s_2) < (q_1,q_2',q_2)}} L(\overline{A_{q_1,s_1,s_2',s_2,q_3}})$$

Such a VPA exists as VPA are closed by intersection and complement. Its size however may be exponential in $|Q|$. We define the look-ahead VPA as the union of all those VPA, $A_{la} = \biguplus B_{q_1,q_1',q_2',q_2,q_3}$. We now define the call and return transitions of T' as follows, for all $c \in \Sigma_c, r \in \Sigma_r, \gamma \in \Gamma, q_1, q_1', q_2', q_3, q \in Q, u \in \Sigma^*$:

$$\begin{array}{rcl}
(q_1,q_3) & \xrightarrow{c|u,\ (\gamma,q_2,q_3),\ p_{q_1,q_1',q_2',q_2,q_3}} & (q_1',q_2') \text{ if } (q_1 \xrightarrow{c|u,\gamma} q_1') \in \delta_c \\
q_0 & \xrightarrow{c|u,\ (\gamma,q_3,q_3),\ p_{q_0,q_1',q_2',q_3,q_3}} & (q_1',q_2') \text{ if } (q_0 \xrightarrow{c|u,\gamma} q_1') \in \delta_c \\
(q_2',q) & \xrightarrow{r|u,(\gamma,q_2,q_3)} & (q_2,q_3) \text{ if } (q_2' \xrightarrow{r|u,\gamma} q_2) \in \delta_r
\end{array}$$

It can be shown that T' is deterministic [16]. Clearly, if T is functional then T_{la} is equivalent. □

This construction, followed by the construction of Theorem 1 that removes the look-aheads, yields a nice characterization of functional VPT:

Theorem 3. *For all functional* VPT T*, one can effectively construct an equivalent unambiguous* VPT T'*.*

5 Decision Problems

In this section, we study the problems of functionality of $\mathsf{VPT}_{\mathsf{la}}$ and equivalence of functional $\mathsf{VPT}_{\mathsf{la}}$. In particular, we prove that while being exponentially more succinct than VPT, the equivalence of functional $\mathsf{VPT}_{\mathsf{la}}$ remains decidable in ExpT, as equivalence of functional VPT.

Theorem 4. *Functionality of* $\mathsf{VPT}_{\mathsf{la}}$ *is* EXPT-C*, even for deterministic look-aheads.*

Proof. For the EXPT upper-bound, we first apply Theorem 1 to remove the look-aheads. This results in a VPT possibly exponentially bigger. Then functionality can be tested in PTIME [9]. For the lower-bound, we reduce the problem of deciding emptiness of the intersection of n deterministic top-down tree automata, which is known to be EXPT-C when n is part of the input [3]. □

We know that the equivalence of two functional VPT is EXPT-C [9]. For equivalence of functional $\mathsf{VPT}_{\mathsf{la}}$, one can first remove the look-aheads, modulo an exponential blow-up, and use the procedure for VPT. This would yield a 2-EXPT procedure for the equivalence of functional $\mathsf{VPT}_{\mathsf{la}}$. However, it is possible to decide it in EXPT:

Theorem 5. *Emptiness of* $\mathsf{VPT}_{\mathsf{la}}$*, resp. of* $\mathsf{VPA}_{\mathsf{la}}$*, equivalence and inclusion of functional* $\mathsf{VPT}_{\mathsf{la}}$*, resp. of* $\mathsf{VPA}_{\mathsf{la}}$*, is* EXPT-C*, even if the transducers, resp. automata, and the look-aheads are deterministic.*

Proof. The lower bounds are obtained, as for functionality, by reduction of the emptiness of n (deterministic) tree automata.
Emptiness of $\mathsf{VPA}_{\mathsf{la}}$ can be checked by first removing the look-aheads (modulo an exponential blow-up) and then check emptiness of the equivalent VPA (in PTIME). Checking emptiness of a $\mathsf{VPT}_{\mathsf{la}}$ amounts to check emptiness of its domain, which is a $\mathsf{VPA}_{\mathsf{la}}$.
To show that equivalence and inclusion of two $\mathsf{VPA}_{\mathsf{la}}$ is in EXPT, we construct two alternating (ranked) tree automata equivalent to the VPA modulo the first-child next-sibling encoding in PTIME. Look-aheads are encoding as universal transitions. Equivalence and inclusion of alternating tree automata is in EXPT [3].
Then, let us show how to check the equivalence, resp. inclusion, of two $\mathsf{VPT}_{\mathsf{la}}$: transform each $\mathsf{VPT}_{\mathsf{la}}$ into an equivalent VPT with at most an exponential blow-up, take the union and verify (in PTIME) that the resulting VPT is still functional. Then check that their domains (which are $\mathsf{VPA}_{\mathsf{la}}$ obtained by ignoring the output of the two $\mathsf{VPT}_{\mathsf{la}}$) are equivalent, resp. included. □

Acknowledgments. We are very grateful to Sebastian Maneth for suggesting us to extend VPT with look-aheads, and to Pierre-Alain Reynier for simplifying the proof of Theorem 1.

References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC, pp. 202–211 (2004)
2. Alur, R., Madhusudan, P.: Adding nesting structure to words. JACM 56(3), 1–43 (2009)
3. Comon, H., Dauchet, M., Gilleron, R., Löding, C., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree automata techniques and applications (2007)
4. Eilenberg, S.: Automata, Languages, and Machines. Academic Press, Inc. (1974)
5. Elgot, C.C., Mezei, J.E.: On relations defined by generalized finite automata. IBM Journal of Research and Development 9, 47–68 (1965)
6. Engelfriet, J.: Top-down tree transducers with regular look-ahead. Mathematical Systems Theory 10, 289–303 (1977)
7. Engelfriet, J.: On tree transducers for partial functions. Inf. Process. Lett. 7(4), 170–172 (1978)

8. Engelfriet, J., Vogler, H.: Macro tree transducers. JCSS 31(1), 71–146 (1985)
9. Filiot, E., Raskin, J.-F., Reynier, P.-A., Servais, F., Talbot, J.-M.: Properties of Visibly Pushdown Transducers. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 355–367. Springer, Heidelberg (2010)
10. Gauwin, O., Niehren, J., Tison, S.: Queries on XML streams with bounded delay and concurrency. Inf. Comput. 209(3), 409–442 (2011)
11. Kumar, V., Madhusudan, P., Viswanathan, M.: Visibly pushdown automata for streaming XML. In: WWW, pp. 1053–1062 (2007)
12. Perst, T., Seidl, H.: Macro forest transducers. IPL 89(3), 141–149 (2004)
13. Sakarovitch, J., de Souza, R.: Lexicographic decomposition of k-valued transducers. TCS 47(3), 758–785 (2010)
14. Schützenberger, M.P.: Sur les relations rationnelles entre monoides libres. TCS 3(2), 243–259 (1976)
15. Segoufin, L., Vianu, V.: Validating streaming XML documents. In: PODS, pp. 53–64 (2002)
16. Servais, F.: Visibly Pushdown Transducers. PhD thesis, Université Libre de Bruxelles (2011)
17. Staworko, S., Laurence, G., Lemay, A., Niehren, J.: Equivalence of Deterministic Nested Word to Word Transducers. In: Kutyłowski, M., Charatonik, W., Gębala, M. (eds.) FCT 2009. LNCS, vol. 5699, pp. 310–322. Springer, Heidelberg (2009)
18. Weber, A.: Decomposing finite-valued transducers and deciding their equivalence. SIAM Journal on Computing 22(1), 175–202 (1993)

A Generalization of Spira's Theorem and Circuits with Small Segregators or Separators

Anna Gál* and Jing-Tang Jang**

Dept. of Computer Science, University of Texas at Austin,
Austin, TX 78712-1188, USA
{panni,keith}@cs.utexas.edu

Abstract. Spira [28] showed that any Boolean formula of size s can be simulated in depth $O(\log s)$. We generalize Spira's theorem and show that any Boolean *circuit* of size s with segregators of size $f(s)$ can be simulated in depth $O(f(s)\log s)$. If the segregator size is at least s^{ε} for some constant $\varepsilon > 0$, then we can obtain a simulation of depth $O(f(s))$. This improves and generalizes a simulation of polynomial-size Boolean circuits of constant treewidth k in depth $O(k^2 \log n)$ by Jansen and Sarma [17]. Since the existence of small balanced separators in a directed acyclic graph implies that the graph also has small segregators, our results also apply to circuits with small separators. Our results imply that the class of languages computed by non-uniform families of polynomial-size circuits that have constant size segregators equals non-uniform NC^1.

Considering space bounded Turing machines to generate the circuits, for $f(s)\log^2 s$-space uniform families of Boolean circuits our small-depth simulations are also $f(s)\log^2 s$-space uniform. As a corollary, we show that the Boolean Circuit Value problem for circuits with constant size segregators (or separators) is in deterministic $SPACE(\log^2 n)$. Our results also imply that the Planar Circuit Value problem, which is known to be P-Complete [16], can be solved in deterministic $SPACE(\sqrt{n}\log n)$.

1 Introduction

Spira [28] proved the following theorem.

Theorem A. *[28] Let F be any Boolean formula of size s. Then F can be simulated by an equivalent formula of depth $O(\log s)$.*

There are several results improving or extending Spira's theorem. Bonet and Buss [3] improved the constants in the depth bounds and the size of the simulation for Boolean formulas, Wegener [30] proved the statement for monotone Boolean formulas, and Brent [5], Bshouty et. al. [6] extended it for arithmetic formulas. All these results study formulas, i.e. tree-like circuits with fan-out 1.

* Supported in part by NSF Grant CCF-1018060.
** Supported in part by MCD fellowship from Dept. of Computer Science, University of Texas at Austin, and NSF Grant CCF-1018060.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 264–276, 2012.

Valiant, Skyum, Berkowitz and Rackoff [29] showed that arithmetic circuits of size s and degree d can be simulated in size $O((sd)^{O(1)})$ and $O(\log s \log d)$ depth. This implies that polynomial-size and polynomial-degree arithmetic circuits can be simulated in NC^2. However, very little is known for size vs. depth for general Boolean circuits. The strongest results so far for general Boolean circuits by Paterson and Valiant [23], and Dymond and Tompa [12] give a simulation of arbitrary Boolean circuits of size s in depth $O(s/\log s)$.

In this paper, we generalize Spira's technique to circuits with small segregators or small separators. Informally, the separator of a graph is a subset of the nodes whose removal yields two subgraphs of comparable sizes. (See the following section for a formal definition.) Graphs with small separators include trees, planar graphs [20], graphs with bounded genus [15], graphs with excluded minors [1], as well as graphs with bounded treewidth [25].

Segregators are a relaxed version of separators of directed acyclic graphs. Paul et al. [24], and Santhanam [26] used segregators to study the computation graph of Turing machines. Directed acyclic graphs with small separators also have small segregators, but the reverse may not necessarily hold.

Jansen and Sarma [17] studied the question of simulating Boolean circuits with bounded treewidth by small-depth circuits. They showed that polynomial-size circuits with constant treewidth k can be simulated in depth $O(k^2 \log n)$, and thus the class of languages with non-uniform polynomial-size bounded treewidth circuits equals non-uniform NC^1.

We extend this result to arbitrary circuits with small segregators and show that any Boolean circuit of size s with segregators (or separators) of size $f(s)$ can be simulated in depth $O(f(s) \log s)$. For circuits with segregators of size k, thus also for graphs with treewidth k, this gives a simulation in depth $k \log s$, improving the bound in [17]. If the segregator size is at least s^ε for some constant $\varepsilon > 0$, then we can obtain a simulation of depth $O(f(s))$. Our results imply that the class of languages computed by non-uniform families of polynomial-size circuits that have constant-size segregators equals non-uniform NC^1.

In [14] we observed that the two-person pebble game of Dymond and Tompa can be used to simulate circuits with small separator size in small depth, giving essentially the same dependence of the depth on the separator size as in the current paper. The approach in [14] based on the two person pebble game can also be extended to graphs with small segregators. However, the simulation based on the two person pebble game is non-uniform, and it seems that the resulting circuits cannot be produced efficiently using this approach. Jansen and Sarma's [17] simulation of bounded treewidth circuits is also non-uniform.

For circuits with constant-size segregators or separators, the simulating circuits we obtain in this paper can be generated in space $O(\log^2 s)$. We also note that our simulation works for any circuit, and if the circuit has a segregator of size $f(s)$, we obtain a simulating circuit of depth at most $O(f(s) \log s)$, the value $f(s)$ does not have to be provided in advance. In contrast, the simulation in [17] assumes that the treewidth k is known in advance, and a tree decomposition is available along with the description of the circuit to be simulated.

It would be desirable to generate the simulating circuits even more efficiently with respect to space or circuit depth, especially in the case of polynomial-size circuits with constant-size segregators or separators, since in that case, as in the case of formulas in Spira's theorem, the resulting circuits are NC^1 circuits. Note however, that even in the case of formulas (tree-like circuits) Spira's theorem is non-uniform. It is not known if the restructuring procedure for formulas in Spira's theorem producing the simulating $O(\log s)$ depth circuits can be directly implemented in less than $O(\log^2 s)$ space, or less than $O(\log^2 s)$ depth [8,9].

The question of finding a uniform version of Spira's theorem has direct relevance for the complexity of the Boolean Formula Value problem. While a logspace uniform version of Spira's restructuring algorithm is still not known, it was proved (by a different approach), that for Boolean formulas presented as parenthesized expressions the Boolean Formula Value problem is in $SPACE(\log n)$ [21], and in $DLOGTIME$-uniform NC^1 [8,9].

Our generalization of Spira's theorem allows us to bound the space complexity of the Circuit Value Problem (CVP) for circuits with small separators and segregators. Ladner [18] showed that the Circuit Value Problem is P-complete. The space complexity of the CVP is not known to be $o(n/\log n)$ for general Boolean circuits. It is a straightforward consequence of Borodin's theorem [4] (see Theorem C) that the CVP for logspace uniform depth d circuits is in $SPACE(d)$ for $d \geq \log n$. It is also easy to see that the CVP for small-width circuits can be solved in small space. Barrington, Lu, Miltersen and Skyum [2] showed that the Monotone Planar Circuit Value Problem is in $LOGDCFL$, and thus in $SPACE(\log^2 n)$. See [10,19] for recent results on variants of the Monotone Planar Circuit Value Problem. As far as we know, the only other variant that was previously shown to be computable in small (polylog) space is the Boolean Formula Value Problem, that is the Circuit Value Problem for tree-like circuits [8,9,21]. We show that the Boolean Circuit Value Problem for circuits with constant-size segregators (or separators) is in deterministic $SPACE(\log^2 n)$. Our results also imply that the Planar Circuit Value problem, which is known to be P-Complete [16], can be solved in deterministic $SPACE(\sqrt{n}\log n)$.

2 Preliminaries

2.1 Space Bounded Turing Machines

For the space complexity of Turing machines, we follow the convention of considering Turing machines with a separate *read-only* input tape, and additional work tapes. If the machine has to produce an output string (instead of just accepting or rejecting its input), then we also assume a separate *write-only* output tape. The space used by a Turing machine on a given input is defined as the number of work tape cells visited during the computation over all work tapes. The input tape and the output tape do not contribute to the space bound of the computation. This allows us to consider computations with sublinear space.

$SPACE(s(n))$ denotes the class of languages decidable by deterministic Turing machines with a separate read-only input tape and a separate write-only output tape using $O(s(n))$ space on the work tapes.

In the following, whenever we talk about space bounds of Turing Machines, it is assumed that the input tape is read-only, the output tape is write-only and the space bound refers to the space used on the work tapes. See Papadimitriou [22] for more details on space bounded Turing machines.

2.2 The Circuit Model

A Boolean circuit is a labeled directed acyclic graph (DAG), where every node is labeled by either a variable from $\{x_1, \ldots, x_n\}$, or an operation from $\{\wedge, \vee, \neg\}$. The inputs of a Boolean circuit are the nodes with in-degree (fan-in) zero, and the outputs of a Boolean circuit are the nodes with out-degree (fan-out) zero. We refer to the nodes (including the inputs) as *gates*. A formula (or tree-like circuit) is a circuit whose fan-out is one for every gate except the output. The size of a Boolean circuit is the number of its gates. We will consider Boolean circuits with gates of fan-in at most 2 from the basis $\{\wedge, \vee, \neg\}$. The *depth of a gate* g is the length of the longest path from any input to g. The *depth of a circuit* C is the depth of the output gate. See [31] for more on Boolean circuits.

Definition 1. *A family of Boolean circuits* $\{C_n\}$ *is called* $h(n)$*-space uniform, if there exists a deterministic Turing machine* M *that on input* 1^n*, outputs the standard description of* C_n *using space* $O(h(n))$ *for all* n*. In particular,* $\{C_n\}$ *is logspace uniform if* $h(n) = \log n$*.*

2.3 Separators and Segregators

Informally, a node separator of a graph G is a set of nodes whose removal yields two disjoint subgraphs of G. In this paper we only consider *balanced* separators, that yield subgraphs that are comparable in size. In the next definition each of the two subDAGs could consist of several weakly connected components.

Definition 2. *A* separator of size k *of a DAG* $G = (V, E)$ *is a set of* k *nodes* $S \subseteq V$ *such that* $G \setminus S$ *is not weakly connected (i.e. the underlying undirected graph is not connected); and the removal of* S *partitions* $G \setminus S$ *into two subDAGs,* $G_1 = (V_1, E_1)$ *and* $G_2 = (V_2, E_2)$*, such that* $|V_i| \leq \frac{2}{3}|V|$ *for* $i = 1, 2$*, and there are no edges either from* G_1 *to* G_2*, or from* G_2 *to* G_1 *in* $G \setminus S$*.*

Segregators are a relaxation of separators in directed acyclic graphs [24,26].

Definition 3. *A* segregator of size k *of a DAG* $G = (V, E)$ *is a set of* k *nodes* $S \subseteq V$ *such that every node in* $G \setminus S$ *has at most* $\frac{2}{3}|V|$ *predecessors in* $G \setminus S$*.*

The following lemma follows directly from the definitions.

Lemma 1. *Any DAG with a separator of size* k *has a segregator of size* k*.*

Notice that the reverse is not true in general, since a node in a DAG may have much smaller number of predecessors than the size of the component that contains the node in the underlying undirected graph.

3 Boolean Circuits with Small Segregators or Separators

Definition 4. *We say that a Boolean circuit C has separators of size $f()$ if the underlying DAG of every subcircuit of C with s gates has a separator of size at most $f(s)$.*

We say that a Boolean circuit C has segregators of size $f()$ if the underlying DAG of every subcircuit of C with s gates has a segregator of size at most $f(s)$.

The above definition is reasonable, since we typically consider classes of circuits based on properties of their underlying DAGs that are closed with respect to subDAGs, for example planar circuits, circuits with small treewidth, etc.

We talk about constant-size separators (resp. segregators), if the size of the separator (resp. segregator) is bounded by a fixed constant that does not depend on the size of the circuit.

By Lemma 1, if the circuit has separators of size $f()$, then it must also have segregators of size $f()$. Therefore in the following we will focus on circuits with small segregators. We prove the following generalization of Spira's theorem.

Theorem 2. *Any Boolean circuit of size s with segregators of size $f()$ can be simulated in depth $O(f(s))$ if $f(s) = \Omega(s^{\varepsilon})$ for some constant $\varepsilon > 0$, and in depth $O(f(s) \log s)$ otherwise.*

Proof. The construction is defined recursively. Let $U = \{u_1, \ldots, u_p\}$ be the segregator of C with size $p \leq f(s)$. Let $C_1, \ldots, C_p$ be the subcircuits of C corresponding to the nodes of the segregator, that is the node u_j is the output of the subcircuit C_j, for $j = 1, \ldots, p$. Let g_j be the Boolean function computed by C_j. Let v be the output node of the circuit C, and let $\hat{C}$ be the circuit with output node v, obtained from C by replacing the nodes in U by new variables $y_1, \ldots, y_p$. Thus, if the original circuit C has n variables, then $\hat{C}$ may have up to $p + n$ variables. It is possible that $\hat{C}$ has less than $p + n$ variables, if some of the original inputs get disconnected from the output v after removing the nodes of the segregator from the circuit.

We enumerate all Boolean vectors $c \in \{0,1\}^p$. Let $c_i = \langle c_{i,1}, c_{i,2}, \ldots, c_{i,p} \rangle$ be the ith Boolean vector of length p, for $i = 1, \ldots, 2^p$, according to some fixed ordering. Let $\hat{C}_i$ be the circuit obtained from $\hat{C}$ by fixing the values of the variables $y_1, \ldots, y_p$ to the bits $c_{i,1}, \ldots, c_{i,p}$, respectively. Let h_i be the Boolean function computed by the circuit $\hat{C}_i$.

Then, the Boolean function computed by the circuit C can be represented using the following expression:

$$\bigvee_{i=1}^{2^p} \left(h_i \wedge \bigwedge_{j=1}^{p} ((g_j \wedge c_{i,j}) \vee (\neg g_j \wedge \neg c_{i,j})) \right) \tag{1}$$

Next we will represent the functions h_i for $i = 1, \ldots, 2^p$ and g_j for $j = 1, \ldots, p$. We could proceed with a straightforward recursion, if we could claim that each

subcircuit $C_1, \ldots, C_p$ and each circuit $\hat{C}_i$ for $i = 1, \ldots, 2^p$ has size at most $2s/3$. In fact, we do know that every subDAG of the underlying DAG of C with the nodes of U removed has size at most $2s/3$. However, the output node of the subcircuit C_j is u_j, and u_j is a member of the segregator U. Note that the underlying DAGs of the circuits $\hat{C}_i$ are identical (they only differ from each other in the substituted constants), and their output node v is the output node of the "original" circuit C. The node v may or may not participate in the segregator. If the node v participates in the segregator, then the functions h_i are constants and the recursion stops.

We can compute the function g_j (computed at gate u_j) by an additional gate if we compute the functions computed at the two children of the gate u_j. If none of the children participates in the segregator, then we know that their subcircuits must have size at most $2s/3$. However, it is possible that children of segregator nodes are also included in the segregator. Let S_j be the set of nodes in the segregator, that are predecessors of u_j, such that there is a path from each of them to u_j that consists only of segregator nodes. We also include u_j in S_j. That is, S_j forms a subcircuit with output u_j that consists of segregator nodes. Let B_j be the "boundary" of S_j formed by nodes that are not in the segregator, that is, B_j contains the children of the nodes in S_j that are not included in the segregator. Then we can compute the function g_j from the functions computed at the nodes in B_j (these can be computed by subcircuits of size at most $2s/3$) with an additional set of gates corresponding to the segregator nodes in S_j. Since $|S_j| \leq p$, this takes additional depth at most p.

To summarize, we can compute the functions h_i and g_j, by first computing in parallel the functions corresponding to all subcircuits after removing the nodes of the segregator. We know that each such subcircuit has size at most $2s/3$, and we can use our construction recursively on these smaller size circuits. Then we finish computing every function h_i and g_j we need, by adding the gates corresponding to the nodes participating in the segregator. This will take at most an additional $p \leq f(s)$ depth. Then we can compute the function computed by C by expression (1). This takes at most an additional $p + \lceil \log(p+1) \rceil + 3 = O(f(s))$ depth. Thus, in each iteration, we increase the depth by at most $O(f(s))$. Since the size is reduced by a constant factor in each iteration, we are done after $O(\log s)$ steps. More precisely, the depth of the final circuit is $O\left(\sum_{i=0}^{\lceil \log_{3/2} s \rceil} f\left((2/3)^i s\right)\right)$. Thus the depth of the final circuit is $O(f(s))$ if $f(s) = s^\epsilon$ for some constant $\epsilon > 0$, or $O(f(s) \log s)$ otherwise. □

Theorem 3. *The class of languages decided by non-uniform families of polynomial-size circuits with constant-size segregators equals non-uniform* NC^1.

Proof. Immediately follows from Theorem 2. □

Robertson and Seymour [25] showed that if a graph has treewidth k, then the graph also has separator size $O(k)$. Together with Lemma 1 and Theorem 3, a polynomial-size circuit with treewidth k can be simulated in depth $O(k \log n)$. This improves a result in [17], which showed that Boolean circuits of size $n^{O(1)}$

and treewidth k can be simulated in non-uniform depth $O(k^2 \log n)$. We refer interested readers to [11] and [13] for more background on treewidth.

4 Finding Minimum Size Segregators in Small Space

4.1 Segregators of Directed Acyclic Graphs

In this section, we give a space-efficient algorithm to find a minimum size segregator in arbitrary directed acyclic graphs.

We will use the following space-efficient algorithm for reachability in directed graphs by Savitch [27], to count the number of predecessors of a given node.

Theorem B. *[27] Given a directed graph G on s nodes and two nodes $u, v \in G$, there exists a deterministic Turing machine that decides if there is a path from u to v in G using space $O(\log^2 s)$.*

Lemma 4. *Let G be a DAG with s nodes. There exists a deterministic Turing machine M such that, on input G, if G has a segregator of size $f(s)$, then M outputs a segregator of G of size at most $f(s)$ using space $O(f(s) \log s + \log^2 s)$.*

Proof. We first define a Turing machine M_1 that takes G and a node $v \in G$ as input, and computes the number of predecessors of v in G, i.e. the number of nodes u such that there exists a directed path from u to v in G. In the beginning M_1 initializes a counter to 1. Then M_1 uses Theorem B to check, one-by-one, for each node $u \in G \setminus \{v\}$ if there is a directed path from u to v in G. For each node $u \in G \setminus \{v\}$ such that v is reachable from u, the counter is incremented. The space used to check the reachability of v from u is reused when checking for reachability from the next node in $G \setminus \{v\}$. Thus M_1 uses $O(\log^2 s)$ space and computes the size of the subDAG with v as the root.

We now define M in Lemma 4 as follows. First M enumerates integers k such that $1 \leq k \leq s$ in increasing order. For a fixed k, M enumerates subsets W of size k of the nodes in G in lexicographic order. For a given W, for every node $u \in G \setminus W$, let $G(u)$ denote the set of predecessors of u in $G \setminus W$. That is, $G(u)$ is the subDAG in $G \setminus W$ with u as its root. M uses M_1 to compute $|G(u)|$. If there exists one node $u \in G \setminus W$ such that $|G(u)| > \frac{2}{3}s$, then M continues to the next W, or the next k if every W of the current size has been already checked. Also, every time before continuing to the next W or the next k, M clears unnecessary information from the work tape.

We now argue that M will find a segregator of the smallest size. Observe that the set of nodes of G is a segregator of size s, so M is guaranteed to find a segregator. Since we try every k in increasing order, and we check for every subset W of size k whether or not it is a segregator, it is guaranteed that we will find a segregator of the smallest possible size in G.

We now argue that M only uses $O(f(s) \log s + \log^2 s)$ space. The description of G can be read using a counter of size $O(\log s)$. The enumeration and the storing of W both take $O(k \log s) = O(f(s) \log s)$ space. The computation of $|G(u)|$ takes $O(\log^2 s)$ space since M_1 uses $O(\log^2)$ space. Thus the space complexity to find a segregator of smallest size is $O(f(s) \log s + \log^2 s)$. □

Note that in the proof for Lemma 4, the input of M consists of only the description of the graph. M does not know the value of $f(s)$ in advance. Also, by Lemma 1, for graphs with separators of size k, the algorithm in Lemma 4 will also find a segregator of size at most k.

4.2 Segregators of Uniform Circuits

Intuitively, Lemma 4 seems to apply directly to circuits since circuits are also DAGs. However, the input of the Turing machine that has to generate the circuit C_n for a uniform family of circuits, is the unary representation of n (1^n), so the graph of the circuit C_n is not available directly. Since we want to generate the segregator using small space, we cannot store the description of C_n on the work tapes. As it is standard in such situations, we will generate the description of C_n as needed for the machine in the proof of Lemma 4, but never store the complete description. We then have the following lemma.

Lemma 5. *Let $\mathcal{C}$ be a $h(n)$-space uniform family of circuits. Let $C_n \in \mathcal{C}$ be the Boolean circuit in the family with n inputs, and assume that C_n has size $s = s(n)$ and a segregator of size $f(s)$. Then there exists a deterministic Turing machine $\hat{M}$ that on input 1^n, outputs a segregator of C_n of size at most $f(s)$ using space $O(h(n) + f(s)\log s + \log^2 s)$.*

As in the case for directed graphs, for circuits with separators of size $f(s)$, the algorithm in Lemma 5 will also find a segregator of size at most $f(s)$.

5 Generating the Simulating Circuits in Small Space

Let v be any node and Z be any set of nodes in the underlying graph of a circuit C_n. We denote by $C_{v,Z}$ the circuit obtained from the subcircuit C_v of C_n with output v by replacing every node in Z that participates in C_v by a new input variable.

Lemma 6. *Let $\mathcal{C}$ be a $h(n)$-space uniform family of circuits. Let $C_n \in \mathcal{C}$ be the circuit with n inputs in the family, and assume that C_n has size $s = s(n)$. Let v be any node and Z be any set of nodes in the underlying graph of C_n. Then there exists a Turing machine M_2 such that on input 1^n, v and Z, M_2 outputs the standard description of the circuit $C_{v,Z}$. Furthermore, M_2 runs in space $O(h(n) + \log^2 s)$.*

Note that if $Z = \emptyset$, or if Z does not contain any predecessors of v then $C_{v,Z}$ is simply the subcircuit C_v. Similarly to the circuit $\hat{C}$ in the proof of Theorem 2, if the size of Z is r, and C_v depends on n' input variables, then $C_{v,Z}$ may have up to $n' + r$ variables. If $v \in Z$, then $C_{v,Z}$ is simply a new variable. The proof of this lemma is standard, and we leave it for the full version.

Lemma 7. *Let $\mathcal{C}$ be a $h(n)$-space uniform family of circuits. Let $C_n \in \mathcal{C}$ be the circuit with n inputs in the family, and assume that C_n has size $s = s(n)$. Let v be any node and Z be any set of nodes in the underlying graph of C_n. Also assume that C_n has segregators of size $f()$. Then there exists a Turing machine M_3 such that on input 1^n, v and Z, M_3 outputs a minimum size segregator of $C_{v,Z}$ using space $O(h(n) + f(s)\log s + \log^2 s)$.*

Proof. Let M_2 be the Turing machine in Lemma 6 that generates the description of $C_{v,Z}$ in space $O(h(n)+\log^2 s)$. Let M be the Turing machine in the statement of Lemma 4, that takes a directed graph G as input, and outputs a minimum size segregator of G. The machine M_3 will simulate M on the underlying directed graph of $C_{v,Z}$. However, as before, the full description of the graph will never be stored. Instead, whenever M_3 needs some information about the graph, it lets M_2 run, (without recording its output), until the required information is generated. The size of the subcircuit $C_{v,Z}$ is $s' \leq s$. Since C_n has segregators of size $f()$, we know that $C_{v,Z}$ has a segregator of size $f(s')$. Recall that M always finds a minimum size segregator, thus it will find a segregator of size $f(s') \leq f(s)$. Since M runs in space $O(f(s)\log s + \log^2 s)$, the total space used will be $O(h(n) + f(s)\log s + \log^2 s)$. □

Now we are ready to prove a uniform version of Theorem 2.

Theorem 8. *Let $\mathcal{C}$ be an $h(n)$-space uniform family of Boolean circuits. Let $C_n \in \mathcal{C}$ be the Boolean circuit on n inputs with size $s = s(n)$. Suppose that C_n has segregators of size $f()$. Let $g(s) = f(s)$ if $f(s) = \Omega(s^c)$ for some constant $c > 0$ and $f(s)\log s$ otherwise. Then $\mathcal{C}$ can be simulated by a $O(h(n)+g(s)\log s)$-space uniform family of Boolean circuits of depth $O(g(s))$.*

Proof. We show that the construction in the proof of Theorem 2 can be generated by a machine M^* within the appropriate space bounds. M^* on input 1^n will output the description of the depth $O(g(s))$ circuit simulating the circuit $C_n \in \mathcal{C}$.

M^* generates the simulating circuit essentially as described in the proof of Theorem 2. In each step of the recursion, M^* has to do the following:

1. Find a segregator S of the current subcircuit, and store the list of nodes of S in workspace.
2. Find and store the list of nodes that participate in $B = \cup_{j=1}^{|S|} B_j$. Note that a given node may belong to B_j for more than one j, but $|\cup_{j=1}^{|S|} B_j| \leq 2|S|$, since B_j contains only children of segregator nodes. Thus, if $|S| = p$, it takes $O(p\log s)$ space to store the list of nodes in B. We can generate this list using $\hat{M}_1$, where $\hat{M}_1$ is the Turing machine that on input 1^n generates the description of C_n using space $O(h(n))$. We will run $\hat{M}_1$ several times, reusing space, and never store the full description of the circuit, as discussed before. For finding the set B_j, we have to find the set S_j and store it until we are finished generating B_j. For each j this takes $O(p\log s)$ workspace. We reuse this space when we move on to the next j. For each node of B_j that we find, we check if we have already added it to the list, so the full list B takes at most $O(p\log s)$ workspace to store.

3. Output the description of the part of the circuit that corresponds to the current subcircuit. This is based on the expression (1), and the sets B_j and S_j. We produce the description of the part of the circuit to compute g_j, while we have B_j and S_j stored in memory. We reuse space when we move on to the next j. Recall that the output is not part of the space bound. (We do keep S and the full list B until the end of processing the subcircuit, and maybe longer as we see below.)

The recursion will continue to process the subcircuits $\hat{C}_i$ (functions h_i) defined in the proof of Theorem 2, and the subcircuits of the nodes in B. Recall that each of these subcircuits has size at most $2/3$ of the last subcircuit. The recursion stops when a subcircuit is either constant or an input variable. We need a counter of size p to enumerate the Boolean vectors substituted, and to enumerate the functions h_i, for $i = 1, \ldots, 2^p$.

We reuse space as we proceed to the next recursive step. However, to be able to proceed with the recursion, we need to retain some information about the segregators S, the sets B and list of values substituted for segregator nodes from previous recursive steps to be able to generate and process the current subcircuits. We process the subcircuits similarly to a depth first search in the recursion tree, starting with the subcircuits corresponding to the set B and leaving the subcircuit for the functions h_i for last. Recall that there is only one subcircuit to consider for the functions h_i, they just differ in the values of constants substituted.

We keep S, B and list of values substituted for nodes in S from previous steps along the current path in the recursion tree. Since there are $\log s$ stages of the recursion, at any point we keep at most $\log s$ segregators with their corresponding set B and list of values. This takes $O(\sum_{i=1}^{\log s} f(s/2^i) \log s) = O(g(s) \log s)$ space.

At the first iteration, we simply use the machine $\hat{M}$ from Lemma 5 to find a segregator. Now we describe how to find a segregator of the current subcircuit during the recursion. To find a segregator for the subcircuits with outputs in the sets B described above, we use M_3 with input 1^n, u where u is the output of the subcircuit, and $Z = \emptyset$. (For processing the subcircuits corresponding to nodes in the sets B we do not need to worry about the segregators that we stored from previous levels of the recursion.) For the subcircuits $\hat{C}_i$ (functions h_i) we use M_3 with input 1^n, v, where v is the output node of the subcircuits $\hat{C}_i$ (recall that they have the same output node, they only differ in the constants substituted), and Z where Z is the union of all the segregators currently stored.

In each step of the recursion, M_3 finds the current segregator in at most $h(n) + O(\log^2 s + f(s) \log s)$ space by Lemma 7. Note that after each invocation of Lemma 7, its workspace can be reused.

Thus on input 1^n, the space used to construct the new circuit is at most $O(h(n) + \log^2 s + g(s) \log s) = O(h(n) + g(s) \log s)$ since $g(s) = \Omega(\log s)$. □

6 Circuit Value Problem

The Boolean Circuit Value problem is defined as follows: given the standard description of a circuit C and an assignment x to the variables of C as the input, compute the value of the output of the circuit C evaluated on the assignment x. As an application of Theorem 8, we obtain a bound on the space complexity of the problem for Boolean circuits with small segregators (or separators). We need the following theorem of Borodin [4].

Theorem C. *[4] Any language decided by a $h(n)$-space uniform circuit family of depth $h(n) \geq \log n$, can be decided by a Turing machine in space $O(h(n))$.*

Theorem 9. *The Boolean Circuit Value problem for circuits that have size s and segregators (or separators) of size $f(s)$ is in deterministic $SPACE(f(s)\log s)$ if $f(s) = \Omega(s^{\varepsilon})$ for some constant $\varepsilon > 0$, and $SPACE(f(s)\log^2 s)$ otherwise.*

Proof. Let $g(s) = f(s)$ if $f(s) = \Omega(s^{\varepsilon})$ for some constant $\varepsilon > 0$, and $g(s) = f(s)\log s$ otherwise. Since the description of C is given in the input, by the proof of Theorem 8, using $O(g(s)\log s)$ space, we can generate a circuit C' of depth $O(g(s))$ that simulates C. Then we can evaluate C' in the given assignment using the argument of Theorem C using space $O(g(s))$.

Theorem 9 immediately implies the following theorem.

Theorem 10. *The Boolean Circuit Value problem for circuits with constant-size segregators (or separators) is in deterministic $SPACE(\log^2 n)$.*

Lipton and Tarjan [20] gave the following "planar separator theorem".

Theorem D. *[20] Any planar graph of size s has a separator of size $O(\sqrt{s})$.*

We use this to obtain our result about the space complexity of the Circuit Value Problem for planar graphs.

Theorem 11. *The Planar Circuit Value Problem can be decided in deterministic $SPACE(\sqrt{n}\log n)$.*

Proof. Immediately follows from Theorem D and Theorem 9.

References

1. Alon, A., Seymour, P., Thomas, R.: A Separator Theorem for Graphs with an Excluded Minor and its Applications. In: Proceedings of STOC, pp. 293–299 (1990)
2. Barrington, D., Lu, C., Miltersen, P.B., Skyum, S.: On monotone planar circuits. In: Proceedings of IEEE Conference on Computational Complexity, pp. 24–33 (1999)
3. Bonet, M., Buss, S.R.: Size-depth tradeoffs for Boolean formulae. Information Processing Letters 49(3), 151–155 (1994)

4. Borodin, A.: On Relating Time and Space to Size and Depth. SIAM Journal on Computing 6(4), 733–744 (1977)
5. Brent, R.P.: The Parallel Evaluation of General Arithmetic Expressions. Journal of the ACM 21(2), 201–206 (1974)
6. Bshouty, N., Cleve, R., Eberly, W.: Size-Depth Tradeoffs for Algebraic Formulas. SIAM Journal on Computing 24(4), 682–705 (1995)
7. Bürgisser, P., Clausen, M., Shokrollahi, M.A.: Algebraic complexity theory. Springer, Heidelberg (1997)
8. Buss, S.R.: The Boolean formula value problem is in ALOGTIME. In: Proceedings of STOC, pp. 123–131 (1987)
9. Buss, S., Cook, S., Gupta, A., Ramachandran, V.: An Optimal Parallel Algorithm for Formula Evaluation. SIAM Journal on Computing 21(4), 755–780 (1992)
10. Chakraborty, T., Datta, S.: One-Input-Face MPCVP Is Hard for L, But in LogDCFL. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 57–68. Springer, Heidelberg (2006)
11. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
12. Dymond, P., Tompa, M.: Speedups of Deterministic Machines by Synchronous Parallel Machines. J. Comp. and Sys. Sci. 30(2), 149–161 (1985)
13. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
14. Gál, A., Jang, J.: The Size and Depth of Layered Boolean Circuits. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 372–383. Springer, Heidelberg (2010)
15. Gilbert, J.R., Hutchinson, J.P., Tarjan, R.E.: A Separator Theorem for Graphs of Bounded Genus. Journal of Algorithms 5(3), 391–407 (1984)
16. Goldschlager, L.: The monotone and planar circuit value problem is complete for P. SIGACT News, 25–27 (1977)
17. Jansen, M., Sarma, J.: Balancing Bounded Treewidth Circuits. In: Ablayev, F., Mayr, E.W. (eds.) CSR 2010. LNCS, vol. 6072, pp. 228–239. Springer, Heidelberg (2010)
18. Ladner, R.E.: The circuit value problem is log-space complete for P. SIGACT News 6(2), 18–20 (1975)
19. Limaye, N., Mahajan, M., Sarma, J.: Upper bounds for monotone planar circuit value and variants. Computational Complexity 18, 377–412 (2009)
20. Lipton, R., Tarjan, R.E.: A Separator Theorem for Planar Graphs. SIAM J. Appl. Math. 36, 177–189 (1979)
21. Lynch, N.A.: Log space recognition and translation of parenthesis languages. J. Assoc. Comput. Mach. 24, 583–590 (1977)
22. Papadimitriou, C.H.: Computational complexity. Addison-Wesley (1994)
23. Paterson, M.S., Valiant, L.G.: Circuit Size is Nonlinear in Depth. Theoretical Computer Science 2(3), 397–400 (1976)
24. Paul, W.J., Pippenger, N., Szemerédi, E., Trotter, W.T.: On determinism versus non-determinism and related problems. In: Proceedings of FOCS, pp. 429–438 (1983)
25. Robertson, N., Seymour, P.D.: Graph Minors II. Algorithmic aspects of tree width. Journal of Algorithms 7, 309–322 (1986)
26. Santhanam, R.: On separators, segregators and time versus space. In: Proceedings of the Sixteenth Annual Conference on Computational Complexity, pp. 286–294 (2000)
27. Savitch, W.J.: Relationships Between Nondeterministic and Deterministic Tape Complexities. Journal of Computer and System Sciences 4(2), 177–192 (1970)

28. Spira, P.M.: On time-hardware complexity tradeoffs for Boolean functions. In: Proc. 4th Hawaii Symp. on System Sciences, pp. 525–527 (1971)
29. Valiant, L., Skyum, S., Berkowitz, S., Rackoff, C.: Fast Parallel Computation of Polynomials Using Few Processors. SIAM J. Comp. 12(4), 641–644 (1983)
30. Wegener, I.: Relating monotone formula size and monotone depth of Boolean functions. Information Processing Letters 16(1), 41–42 (1983)
31. Wegener, I.: The Complexity of Boolean Functions (1987)

Consistent Consequence for Boolean Equation Systems

Maciej W. Gazda and Tim A.C. Willemse

Eindhoven University of Technology, Eindhoven, The Netherlands

Abstract. Inspired by the concept of a consistent correlation for Boolean equation systems, we introduce and study a novel relation, called *consistent consequence*. We show that it can be used as an approximation of the solution to an equation system. For the closed, simple and recursive fragment of equation systems we prove that it coincides with *direct simulation* for parity games. In addition, we show that deciding both consistent consequence and consistent correlations are coNP-complete problems, and we provide a sound and complete proof system for consistent consequence. As an application, we define a novel abstraction mechanism for parameterised Boolean equation systems and we establish its correctness using our theory.

1 Introduction

Boolean equation systems [9] have been studied extensively in the context of software and hardware verification, in which the equation systems appear naturally as the end result of encodings of model checking problems, including the modal μ-calculus problem [9] and data and real-time model checking problems [15]; but also as a result of process equivalence checking problems [11]. The encoded verification problems can be answered by solving the associated equation system, which is known to be in NP $\cap$ coNP.

The Boolean equation systems originating from verification problems tend to be rather large. However, so called *parameterised Boolean equation systems* [10,6] can be used to concisely describe the Boolean equation systems, using a combination of data, parameterised recursion and first-order quantification. Subsequently generating the Boolean equation systems from such parameterised Boolean equation systems, however, leads to problems akin to the infamous state-space explosion. In an attempt to side-step the latter phenomenon, transformations that simplify parameterised Boolean equation systems have been studied, see *e.g.* [13]; such transformations are reasonably cheap, but can be extremely effective at reducing the size of the underlying Boolean equation system.

Establishing the soundness of a given transformation has long been a rather tiresome affair; only recently, a proof methodology has been devised that avoids the need for lengthy proofs. The method relies on the identification of a suitable *consistent correlation* [14] between two (parameterised) Boolean equation systems. While the decidability of such consistent correlations was addressed for a fragment of Boolean equation systems, the decidability and the computational complexity for the general setting of Boolean equation systems remained open.

An inconvenience of consistent correlations is that they only serve to reason about transformation that are solution preserving and reflecting. The technique falls short in

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 277–288, 2012.

situations in which one wishes to devise more aggressive transformations that allow us to under- or over-approximate the solution to a given (parameterised) Boolean equation system. Compared to solution preserving transformations, such transformations offer the prospect of even more powerful reductions. To accommodate the development of such transformations, we define and study a *preorder* that is tightly related to the concept of consistent correlation. The resulting preorder is called *consistent consequence*. In addition, we study its computational complexity and its relation to concepts known in the setting of Boolean equation systems and in related settings. Our theoretical contributions can be summarised as follows:

- We establish the relation between consistent consequence, consistent correlation and the concept of a solution to an equation system;
- We link consistent consequence for the fragment of closed Boolean equation systems in simple and recursive form to *direct simulation* for *parity games* [12], providing a graph-based, operational view on the concept;
- We prove that deciding consistent consequence and consistent correlation are both coNP-complete problems, answering the open problem raised in [14];
- We give a sound and complete proof system for consistent consequence and claim the soundness and completeness of the proof system for consistent correlation, conjectured in [14].

Given the notationally more involved setting of parameterised Boolean equation systems, we have opted to study the concept of consistent consequence in the more accessible framework of Boolean equation systems. Lifting the concepts to the richer setting, following [14], is interesting but routine.

We conclude our paper by defining a novel abstraction mechanism for (parameterised) Boolean equation systems, the soundness of which is easily established by an appeal to our new theory. The abstraction mechanism allows one to solve parameterised Boolean equation systems that could not be solved before.

Outline. We give a cursory overview of the Boolean equation system framework in Section 2. In Section 3, we define and study our notion of consistent consequence, and we briefly analyse its computational complexity. The correspondence to direct simulation for parity games is addressed in Section 4. In Section 5, we present our proof system for consistent consequence and state the soundness and completeness of the proof system for consistent correlation, conjectured in [14]. We apply our theory by defining a novel manipulation on parameterised Boolean equation systems, and prove its correctness in Section 6. We wrap up with some ideas for future work in Section 7.

2 Preliminaries

Boolean equation systems are finite sequences of fixed point equations, in which the right-hand sides of the equations are proposition formulae. We assume that the reader has some familiarity with fixed point theory and Boolean equation systems; for an excellent, in-depth account on the latter, we refer to [9].

Definition 1. *A* Boolean equation system (BES) $\mathcal{E}$ *is defined by the following grammar:*

$$\begin{aligned}\mathcal{E} &::= \epsilon \mid (\nu X = f)\,\mathcal{E} \mid (\mu X = f)\,\mathcal{E}\\ f, g &::= \top \mid \bot \mid X \mid f \wedge g \mid f \vee g\end{aligned}$$

The empty BES is denoted ϵ*;* X *is a proposition variable taken from a countable set* $\mathcal{X}$ *of proposition variables;* f, g *are proposition formulae.*

From hereon, whenever we write σ, we mean an arbitrary fixed point sign, *i.e.*, either μ or ν. As a notational convention, we write f_X when we refer to the right-hand side proposition formula in the equation for X: we have $\sigma X = f_X$.

Let $\mathcal{E}$ be an arbitrary equation system. We denote the set of *bound* proposition variables of $\mathcal{E}$ by $\mathsf{bnd}(\mathcal{E})$; that is, $\mathsf{bnd}(\mathcal{E})$ contains those variables at the left-hand side of the equations in $\mathcal{E}$. We only consider equation systems in which all equations have unique left-hand side variables. Proposition variables occurring in a proposition formula f are collected in the set $\mathsf{occ}(f)$; by extension, $\mathsf{occ}(\mathcal{E})$ contains variables occurring in the right-hand side of any of the equations in $\mathcal{E}$.

Whenever all occurring variables are bound in an equation system, it is said to be a *closed* equation system. An equation system is in *simple form* [1] if none of the right-hand sides of the equations that occur in the equation system contain both $\wedge$- and $\vee$-operators. If none of an equation system's right-hand side formulae contain the constants $\top$ and $\bot$, the system is in *recursive form*.

To each *bound* variable X of $\mathcal{E}$, we associate a *rank* $\mathsf{rank}_{\mathcal{E}}(X)$, which is defined as $\mathsf{rank}_{\mathcal{E}}(X) = \mathsf{block}_{\nu,X}(\mathcal{E})$:

$$\mathsf{block}_{\sigma,X}((\sigma' Y = f_Y)\mathcal{E}) = \begin{cases} 0 & \text{if } \sigma = \sigma' \text{ and } X = Y \\ \mathsf{block}_{\sigma,X}(\mathcal{E}) & \text{if } \sigma = \sigma' \text{ and } X \neq Y \\ 1 + \mathsf{block}_{\sigma',X}((\sigma' Y = f_Y)\mathcal{E}) & \text{if } \sigma \neq \sigma' \end{cases}$$

Informally, the rank of a variable X is the i-th block of like-signed equations, containing X's defining equation, counting from left-to-right and starting at 0 if the first block consists of greatest fixed point signs, and 1 otherwise.

Semantics. Let $\eta{:}\mathcal{X} \to \mathbb{B}$ be a *proposition environment*, assigning Boolean values to proposition variables. The semantics of a proposition formula f is given in the context of an environment η (notation $[\![f]\!]\eta$) and is defined as the standard extension of η to formulae. We write $\eta[X := b]$ for the environment η in which the proposition variable X has Boolean value b and all other proposition variables X' have value $\eta(X')$. The ordering $\sqsubseteq$ on environments is defined as $\eta \sqsubseteq \eta'$ if and only if $\eta(X)$ implies $\eta'(X)$ for all X. For reading ease, we do not formally distinguish between a semantic Boolean value and its representation by $\top$ and $\bot$; likewise, for the operands $\wedge$ and $\vee$.

Definition 2. *The* solution of a BES*, given an environment* $\eta{:}\mathcal{X} \to \mathbb{B}$*, is inductively defined as follows:*

$$\begin{aligned}[\![\epsilon]\!]\eta &= \eta\\ [\![(\sigma X = f)\,\mathcal{E}]\!]\eta &= \begin{cases} [\![\mathcal{E}]\!](\eta[X := [\![f]\!]([\![\mathcal{E}]\!]\eta[X := \bot])]) & \text{if } \sigma = \mu \\ [\![\mathcal{E}]\!](\eta[X := [\![f]\!]([\![\mathcal{E}]\!]\eta[X := \top])]) & \text{if } \sigma = \nu \end{cases}\end{aligned}$$

It is not hard to verify that the order of equations impacts the solution: the equation system $(\mu X = Y)(\nu Y = X)$ has $\bot$ as the solution to both X and Y, whereas the equation system $(\nu Y = X)(\mu X = Y)$ will yield the solution $\top$ for both X and Y.

3 Consistent Consequence

Let R be an arbitrary relation on proposition variables $\mathcal{X}$. We write $X \ R \ X'$ iff $(X, X') \in R$. Let θ be an arbitrary environment; θ is *consistent* with R if for all $X \ R \ X'$, $\theta(X) \Rightarrow \theta(X')$. We denote the set of all environments consistent with R by Θ_R.

Definition 3. *Let $\mathcal{E}$ be an equation system. Let R be a relation on proposition variables; R is a* consistent consequence *on $\mathcal{E}$ if for all equations $\sigma X = f_X$ and $\sigma' X' = f_{X'}$ in $\mathcal{E}$ such that $X \ R \ X'$, we have:*

1. $\mathsf{rank}_{\mathcal{E}}(X) = \mathsf{rank}_{\mathcal{E}}(X')$
2. *for all $\theta \in \Theta_R$, we have $[\![f_X]\!]\theta \Rightarrow [\![f_{X'}]\!]\theta$.*

A proposition variable $X' \in \mathsf{bnd}(\mathcal{E})$ is said to be a consistent consequence *of proposition $X \in \mathsf{bnd}(\mathcal{E})$, denoted $X \lessdot X'$ iff there is some relation R that is a consistent consequence on $\mathcal{E}$, such that $X \ R \ X'$.*

Property 1. For any pair of consistent consequences R, S, their union $R \cup S$ is also a consistent consequence.

As a consequence of the above property, we find that there must be a *largest* consistent consequence relation.

Proposition 1. *The relation $\lessdot$ is the largest consistent consequence. Moreover, $\lessdot$ is a preorder.* □

The theorem below is the main result of this section; it establishes a link between the notion of a consistent consequence and the concept of a solution to an equation system.

Theorem 1. *Let $\mathcal{E}$ be an equation system. Let R be a consistent consequence on $\mathcal{E}$. Then for all $\theta \in \Theta_R$ we have $[\![\mathcal{E}]\!]\theta \in \Theta_R$.*

Proof. The proof proceeds using a combination of induction on the length of the equation system $\mathcal{E}$ and approximation of simultaneous fixed points, relying on Bekič principle to convert the nested fixed points to such simultaneous fixed points. □

The notion of consistent consequence is closely related to the notion of a *consistent correlation*, see [14]. For the sake of completeness, we here recall its definition.

Definition 4. *Let $\mathcal{E}$ be an equation system. Let R be a symmetric relation on proposition variables; R is a* consistent correlation *on $\mathcal{E}$ if for all equations $\sigma X = f_X$ and $\sigma' X' = f_{X'}$ in $\mathcal{E}$ such that $X \ R \ X'$, we have:*

1. $\mathsf{rank}_{\mathcal{E}}(X) = \mathsf{rank}_{\mathcal{E}}(X')$
2. *for all $\theta \in \Theta_R$, we have $[\![f_X]\!]\theta = [\![f_{X'}]\!]\theta$.*

Variables $X, X' \in \mathsf{bnd}(\mathcal{E})$ consistently correlate*, denoted* $X \doteqdot X'$*, iff there is some consistent correlation* R *such that* $X \ R \ X'$*.*

The results below relate consistent consequence and consistent correlation: consistent consequence relates to consistent correlation in the same way as simulation preorder relates to bisimulation in the setting of labelled transition systems.

Proposition 2. *Let* $\mathcal{E}$ *be an arbitrary equation system.*

1. *The largest consistent correlation on* $\mathcal{E}$*, denoted* $\doteqdot$*, is the largest symmetric consistent consequence on* $\mathcal{E}$*;*
2. *The largest consistent correlation on* $\mathcal{E}$*,* $\doteqdot$ *is contained in* $\lessdot$ *and in* $\lessdot^{-1}$*.* □

We finish this section with an example that illustrates that consistent correlation $\doteqdot$ is, as can be expected, actually finer than $\lessdot \cap \lessdot^{-1}$.

Example 1. Consider the equation system $\mathcal{E}$ given below:

$$(\mu X_0 = X_4)\ (\mu X_1 = X_0 \vee X_2)\ (\mu X_2 = X_4 \vee X_5)\ (\mu X_3 = X_2)\ (\mu X_4 = X_4)\ (\nu X_5 = X_5)$$

Observe that we have, among others, $X_0 \lessdot X_2$, $X_1 \lessdot X_3$, $X_3 \lessdot X_1$, $X_4 \lessdot X_0$; in particular, this means that X_1 is a consistent consequence of X_3 and *vice versa*. Note also that $X_1 \not\doteqdot X_3$: take, for instance $\theta(X_0) = \top$ and $\theta(X_2) = \bot$, which would be allowed if not $X_0 \doteqdot X_2$. But this follows from the fact that $X_4 \not\doteqdot X_5$, as a result of their ranks. Finally, observe that neither $\lessdot$, nor $\doteqdot$ coincide with the partitioning induced by the solution to the equation system: the solution to both X_0 and X_4 is $\bot$, whereas the solution to X_1, X_2, X_3 and, in particular, X_5 is $\top$; note that X_5 cannot be related to, *e.g.*, X_1, due to a difference in ranks. □

Complexity. In [14], it was shown that deciding $\doteqdot$ for the class of equation systems in simple form requires $O(n \log n)$ time, where n is the number of bound variables in an equation system. The complexity for deciding $\doteqdot$ for arbitrary equation systems so far remained an open problem; both problems are coNP decision problems.

Proposition 3. *Let* $\mathcal{E}$ *be an arbitrary equation system. The decision problems* $X \lessdot Y$ *and* $X \doteqdot Y$*, for* $X, Y \in \mathsf{bnd}(\mathcal{E})$ *are both in coNP.* □

In fact, both decision problems are coNP-complete problems.

Theorem 2. *Deciding* $\lessdot$ *and* $\doteqdot$ *is coNP-complete.*

Proof. The problem can be reduced to the logical consequence problem of [3]. □

The coNP-completeness proof relies on an equation system consisting of two blocks. For equation system consisting of only one block, deciding $\doteqdot$ remains coNP-complete, but $\lessdot$ can be decided in linear time.

Remark 1. For equation systems with either all right-hand sides in *Conjunctive Normal Form* or all in *Disjunctive Normal Form*, both decision problems remain polynomial.

4 Consistent Consequence Generalises Direct Simulation on Parity Games

Equation systems generalise *Parity Games*: two-player, graph-based games [12] with ω-winning conditions. We show that for the fragment of equation systems in simple form, consistent consequence on equation systems coincides with *direct simulation*[1] on parity games. Note that despite the fact that parity games and equation systems are mutually reducible to one another, the correspondence that we establish in this section is non-trivial, given the contrasts between the denotational framework of equation systems and the highly operational characteristics of parity games.

Definition 5. *A* parity game *is a game graph* $\mathcal{G} = \langle V, \to, \Omega, (V_{Even}, V_{Odd})\rangle$*, where* V *is a finite set of vertices partitioned in sets* V_{Even} *and* V_{Odd}*,* $\to \subseteq V \times V$ *is a total set of edges and* $\Omega{:}V \to \mathbb{N}$ *is the priority function. Sets* V_{Even} *and* V_{Odd} *consist of vertices owned by player* even *and* odd*, respectively.*

We forego a formal exposition of the concept of winning vertices in a parity game; for details, we refer to [12]. For our purpose in this section, it suffices to be aware that the set of vertices of a parity game can be partitioned in unique sets W_{Even} and W_{Odd}, representing those vertices in V won by player *even* and those won by player *odd*.[2]

The direct simulation preorder on vertices of a parity game is defined through a game played on an auxiliary graph, called the *simulation game-graph* (*i.e.*, *not* a parity game graph). Given a parity game $\mathcal{G} = (V, \to, \Omega, (V_{\text{Even}}, V_{\text{Odd}}))$, the simulation game is a turn-based game played by players *Duplicator* (D) and *Spoiler* (S) on a game-graph $(V \times V, \to \times \to)$ according to moves adhering to the rules in Table 1. An infinite play $(v_0, w_0)(v_1, w_1) \cdots \in (V \times V)^\omega$ is won by Duplicator if all priorities match along the play, *i.e.*, $\Omega(v_n) = \Omega(w_n)$ for all n; otherwise it is won by Spoiler.

Table 1. Admissible moves in a simulation game-graph

$(v, w) \in$	1st player	plays on	2nd player	plays on
$V_{\text{Even}} \times V_{\text{Even}}$	S	v	D	w
$V_{\text{Even}} \times V_{\text{Odd}}$	S	v	S	w
$V_{\text{Odd}} \times V_{\text{Even}}$	D	w	D	v
$V_{\text{Odd}} \times V_{\text{Odd}}$	S	w	D	v

Definition 6. *Let* $\mathcal{G} = (V, \to, \Omega, (V_{Even}, V_{Odd}))$ *be a parity game. We say that a vertex* v *is* directly simulated *by vertex* w*, denoted* $v \sqsubseteq_{dir} w$ *if player* Duplicator *(D) has a winning strategy for the simulation game starting in* (v, w)*.*

We have the following relation between vertices won by players *even* and *odd*, and the direct simulation relation, as suggested in [5], and as a consequence of our Theorem 3.

[1] It appears that this notion was never formally defined for parity games, but for a variant in the setting of Büchi automata, see [4], and based on suggestions in [5], it can be reconstructed.

[2] The sets W_{Even} and W_{Odd} should not be confused with V_{Even} and V_{Odd}; these are not related.

$$(\mu X_0 = X_4)$$
$$(\mu X_1 = X_0 \vee X_2)$$
$$(\mu X_2 = X_4 \vee X_5)$$
$$(\mu X_3 = X_2)$$
$$(\mu X_4 = X_4)$$
$$(\nu X_5 = X_5)$$

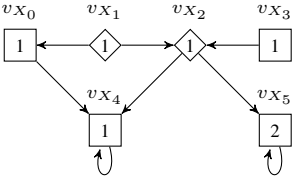


Fig. 1. The equation system $\mathcal{E}$ from Example 1, and its associated parity game

Proposition 4. *Let* $\mathcal{G} = (V, \rightarrow, \Omega, (V_{Even}, V_{Odd}))$ *be a parity game. Let* $v, w \in V$. *If* $v \sqsubseteq_{dir} w$ *and vertex* v *is won by player* even, *then so is vertex* w. □

Parity games and equation systems in simple, recursive form are known to coincide: see, *e.g.*, the linear reductions in both directions in [7]. These reductions are such that a vertex in the game is won by player *even* iff the corresponding equation in the equation system has *true* as its solution. Based on these reductions, we have the following result.

Theorem 3. *The preorder* $\sqsubseteq_{dir}$ *on parity games coincides with the preorder* $\lessdot$ *on closed equation systems in simple and recursive form.* □

Direct simulation can be computed in polynomial time using the framework for games with Büchi winning conditions, see [4]. Clearly, the same technique applies to computing consistent consequence on closed equation systems in simple recursive form; the extension of this technique to *open* equation systems in simple form is standard.

Example 2. Reconsider the equation system $\mathcal{E}$ from Example 1. Its associated parity game is given below; vertices owned by player *even* are diamond-shaped, whereas vertices owned by player *odd* box-shaped. The priorities are written inside the vertices. One can check that vertex v_{X_1} simulates v_{X_3}, corroborating our earlier claim that $X_1 \lessdot X_3$, and *vice versa*. In a similar vein, we have both $v_{X_0} \sqsubseteq_{dir} v_{X_2}$ and $v_{X_4} \sqsubseteq_{dir} v_{X_0}$. □

5 The Proof System $\vdash_c$

The definition of a consistent consequence is phrased entirely in terms of the semantics of the artefacts of an equation system. In [14], it was conjectured that the notion of a consistent correlation $\div$ can be characterised by a proof system that adds only a single coinductive rule to the standard axiomatisation of logical equivalence for negation-free propositional logic. Such an elegant, syntax-based proof system offers an accessible alternative to the semantic definition; this ultimately provides a better understanding of the concept. We vindicate the conjecture in [14], and provide a similar-spirited proof system for consistent consequence.

We write $f \subset g$ to denote that g is a logical consequence of f. Let $\vdash_P$ denote the proof system for logical consequence for negation-free propositional logic, given by the rules in Table 2, save the rules CC and CNT.

Lemma 1. *The proof system* $\vdash_P$ *is sound and complete for logical consequence.* □

Table 2. Proof system for negation-free logical consequence and consistent consequence; α, β, γ represent arbitrary proposition formulae; X, Y are proposition variables; and Γ is a context. Furthermore, ς is a substitution mapping proposition variables to proposition formulae; $f\varsigma$ denotes the natural extension of mapping ς from variables to terms. The rules axiomatise associativity (AS), distributivity (DS), absorption (AB), idempotence (ID), supremum (SUP) and infimum (INF) and top (TOP) and bottom (BOT).

Axioms for negation-free propositional logic

rules of the form $\dfrac{}{\Gamma \vdash A}$, where A ranges over the following laws:

AS1	$\alpha \wedge (\beta \wedge \gamma) \subset (\alpha \wedge \beta) \wedge \gamma$	DS1	$\alpha \vee (\beta \wedge \gamma) \subset (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$
AS2	$(\alpha \wedge \beta) \wedge \gamma \subset \alpha \wedge (\beta \wedge \gamma)$	DS2	$(\alpha \vee \beta) \wedge (\alpha \vee \gamma) \subset \alpha \vee (\beta \wedge \gamma)$
AS3	$\alpha \vee (\beta \vee \gamma) \subset (\alpha \vee \beta) \vee \gamma$	DS3	$\alpha \wedge (\beta \vee \gamma) \subset (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$
AS4	$(\alpha \vee \beta) \vee \gamma \subset \alpha \vee (\beta \vee \gamma)$	DS4	$(\alpha \wedge \beta) \vee (\alpha \wedge \gamma) \subset \alpha \wedge (\beta \vee \gamma)$
COM1	$\alpha \wedge \beta \subset \beta \wedge \alpha$	AB1	$\alpha \vee (\alpha \wedge \beta) \subset \alpha$
COM2	$\alpha \vee \beta \subset \beta \vee \alpha$	AB2	$\alpha \subset \alpha \wedge (\alpha \vee \beta)$
ID1	$\alpha \subset \alpha \wedge \alpha$	ID2	$\alpha \vee \alpha \subset \alpha$
SUP	$\alpha \subset \alpha \vee \beta$	INF	$\alpha \wedge \beta \subset \alpha$
TOP	$\alpha \subset \alpha \wedge \top$	BOT	$\alpha \vee \bot \subset \alpha$

Inequality logic rules

$$\text{SUB}\quad \frac{\Gamma \vdash_c \alpha \subset \beta}{\Gamma \vdash_c \alpha\varsigma \subset \beta\varsigma} \qquad \text{CTX}\quad \frac{\Gamma \vdash_c \alpha \subset \beta}{\Gamma \vdash_c \gamma[X := \alpha] \subset \gamma[X := \beta]}$$

$$\text{TRA}\quad \frac{\Gamma \vdash_c \alpha \subset \beta \quad \beta \subset \gamma}{\Gamma \vdash_c \alpha \subset \gamma} \qquad \text{REF}\quad \frac{}{\Gamma \vdash_c \alpha \subset \alpha}$$

Consistent consequence rules

$$\text{CC}\quad \frac{\Gamma, X \subset Y \vdash_c f_X \subset f_Y \qquad \mathsf{rank}(X) = \mathsf{rank}(Y)}{\Gamma \vdash_c X \subset Y}$$

$$\text{CNT}\quad \frac{}{\Gamma \vdash_c X \subset Y}\ (X \subset Y) \in \Gamma$$

Before we address the soundness and completeness of the proof system for consistent consequence, we parameterise the definition of $\lessdot$ to facilitate reasoning about a context Γ, which can be thought of as a relation on propositional variables.

Definition 7. *Let R, Γ be relations on proposition variables. Let $\mathcal{E}$ be an equation system. Then R is a consistent consequence* relative to *Γ if, whenever $X\ R\ X'$, for $X, X' \in \mathsf{bnd}(\mathcal{E})$, we have:*

1. $\mathsf{rank}_{\mathcal{E}}(X) = \mathsf{rank}_{\mathcal{E}}(X')$;
2. *for all $\theta \in \Theta_{R \cup \Gamma}$, we have $[\![f_X]\!]\theta \Rightarrow [\![f_{X'}]\!]\theta$.*

Proposition X' is said to be a consistent consequence of X, relative to context Γ, denoted $X \lessdot_\Gamma X'$ iff there is a consistent consequence relation R relative to Γ, such that $X(R \cup \Gamma)X'$.

We next generalise the notion of a consistent consequence relative to a context to apply to arbitrary proposition formulae; the generalisation is standard.

Definition 8. *Let f, g be proposition formulae. We say that g is a consistent consequence of f, relative to context Γ, denoted $f \lessdot_\Gamma g$ iff for all $\theta \in \Theta_{R \cup \Gamma}$, we have $[\![f]\!]\theta \Rightarrow [\![g]\!]\theta$.*

From hereon, we tacitly assume a fixed equation system $\mathcal{E}$. Henceforth, $\subset$ denotes the relation induced by all rules found in the proof system in Table 2. We write $\Gamma \vdash_c f \subset g$ to denote that the rules in Table 2 enables us to derive that $f \subset g$ in the context Γ.

Theorem 4. *The proof system $\vdash_c$ is sound for $\lessdot$.* □

Essential to our completeness result, we first claim that for all $X \lessdot Y$, we have $\vdash_c X \subset Y$; in other words, there is always a finite proof tree with root $X \subset Y$.

Lemma 2. *For all $\Gamma \subseteq \lessdot$ satisfying $f_1 \lessdot_\Gamma f_2$, the below rule is derivable in $\vdash_c$:*

$$\frac{H}{\Gamma \vdash_c f_1 \subset f_2} \quad \text{(R2)}$$

where either $H = \emptyset$ or nodes in H are of the form $\Gamma' \vdash_c f \subset g$ for some γ' satisfying $\#\Gamma' > \#\Gamma$.

Theorem 5. *Proof system $\vdash_c$ is complete for $\lessdot$.*

Proof. Take any X, Y such that $X \lessdot Y$. We can construct a finite proof tree by first applying the rule R2 from Lemma 2. We thus obtain:

$$\frac{H_1}{\emptyset \vdash_c X \subset Y}$$

We continue to apply rule R2 to the premises from H_1 and to thus obtained new premises as long as necessary. On each level H_n we have $\#\Gamma = n$ and since, as a result of the finite size of an equation system, the number of possible pairs in the context Γ is finite, the proof tree is well-founded. □

Theorem 6. *The proof system conjectured in [14] is sound and complete for $\doteqdot$.* □

Example 3. As an illustration of the use of our proof system, we will prove the claimed $X_1 \lessdot X_3$ from Example 1.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{X_0 \subset X_2, X_1 \subset X_3 \vdash_c X_4 \subset X_4 \vee X_5}\,\text{(SUP)}}{X_1 \subset X_3 \vdash_c X_0 \subset X_2}\,\text{(CC)}}{X_1 \subset X_3 \vdash_c X_0 \vee X_2 \subset X_2 \vee X_2}\,\text{(CTX)} \qquad \dfrac{}{X_2 \vee X_2 \subset X_2}\,\text{(ID2)}}{X_1 \subset X_3 \vdash_c X_0 \vee X_2 \subset X_2}\,\text{(TRA)}}{X_1 \subset X_3}\,\text{(CC)}$$

6 Application

We next use the concept of consistent consequence to establish the correctness of a novel abstraction mechanism for *parameterised* Boolean equation systems. Our mechanism is inspired by abstraction techniques for behavioural systems [2].

Due to the imposed page limits, we limit ourselves to describing the syntax of parameterised Boolean equation systems; for a detailed, formal exposition of the framework, we refer to [6]. For the purpose of this section, parameterised Boolean equation systems can be thought of as describing (possibly infinite sized) blocks of Boolean equation systems with first-order right-hand side formulae. We assume some familiarity with model checking and the μ-calculus.

Definition 9. *A* parameterised Boolean equation system (PBES) $\mathcal{E}$ *is defined by the following grammar:*

$$\begin{array}{ll} \mathcal{E} & ::= \epsilon \mid (\nu X(\boldsymbol{d_X}{:}\boldsymbol{D_X}) = \phi) \mid (\mu X(\boldsymbol{d_X}{:}\boldsymbol{D_X}) = \phi) \\ \phi, \psi & ::= b \mid \phi \wedge \psi \mid \phi \vee \psi \mid \forall d{:}D.\ \phi \mid \exists d{:}D.\ \phi \mid X(\boldsymbol{e}) \end{array}$$

The expressions ϕ, ψ are predicate formulae; b is a basic Boolean expression, possibly containing data variables $d \in \mathcal{D}$. Variable X, taken from some countable set $\mathcal{P}$, is a (sorted) predicate variable; $\boldsymbol{e}$ is a vector of data expressions. We write $\phi(\boldsymbol{d})$ to indicate that the variables in $\boldsymbol{d}$ occur freely in ϕ.

PBESs allow one to encode various verification problems, such as model checking (first order) modal μ-calculus formulae and verifying various well-known process equivalences. Despite the undecidability of solving PBESs, syntactic manipulations enable one to compute the solution in many practical situations. Most manipulations preserve underlying consistent correlations. In contrast, we next present an abstraction technique that is inspired by consistent consequence.

Definition 10. *Let, for each X, $h_X{:}\boldsymbol{D_X} \to \widehat{\boldsymbol{D}}_{\boldsymbol{X}}$ be a surjection, mapping a* concrete domain $\boldsymbol{D_X}$ *to an* abstract domain $\widehat{\boldsymbol{D}}_{\boldsymbol{X}}$*. We denote the union of all mappings h_X by h. The* abstraction $\mathcal{F}^m(\phi)$ *of ϕ, for $m \in \{\sqcup, \sqcap\}$, in the context of h is defined inductively:*

$$\begin{array}{lll} \mathcal{F}^m(b(\boldsymbol{d})) & = \begin{cases} \exists \boldsymbol{d}.\ h(\boldsymbol{d}) = \widehat{\boldsymbol{d}} \wedge b \text{ if } m = \sqcup \\ \forall \boldsymbol{d}.\ h(\boldsymbol{d}) \neq \widehat{\boldsymbol{d}} \vee b \text{ otherwise} \end{cases} & \\ \mathcal{F}^m((Y(e))(\boldsymbol{d})) & = \begin{cases} \exists \boldsymbol{d}.\ h(\boldsymbol{d}) = \widehat{\boldsymbol{d}} \wedge \widehat{Y}(h(e)) \text{ if } m = \sqcup \\ \forall \boldsymbol{d}.\ h(\boldsymbol{d}) \neq \widehat{\boldsymbol{d}} \vee \widehat{Y}(h(e)) \text{ otherwise} \end{cases} & \\ \mathcal{F}^m((\phi \oplus \psi)(\boldsymbol{d})) & = \mathcal{F}^m(\phi(\boldsymbol{d})) \oplus \mathcal{F}^m(\psi(\boldsymbol{d})) & \text{for } \oplus \in \{\wedge, \vee\} \\ \mathcal{F}^m((\mathsf{Q}\, d'{:}D.\ \phi)(\boldsymbol{d})) & = \mathsf{Q}\, \widehat{d}'{:}\widehat{D}.\ \mathcal{F}^m(\phi(d', \boldsymbol{d})) & \text{for } \mathsf{Q} \in \{\forall, \exists\} \end{array}$$

The abstraction function $\mathcal{F}$ easily extends to PBESs. The following theorem states that $\mathcal{F}^m(\phi)$ can be used to under-approximate, resp. over-approximate the solution to a PBES. The correctness of this theorem can be established by proving the mapping h induces a consistent consequence on the predicate variables involved in a PBES; lifting the theory of consistent consequence to PBESs can be done along the lines of [14].

Theorem 7. *Let $\mathcal{E}$ be an arbitrary closed, non-empty PBES. Let h be a mapping from concrete data domains to abstract data domains. We have, for all environments η, all $X \in \mathsf{bnd}(\mathcal{E})$ and all v:*

$$[\![\mathcal{F}^{\sqcap}(\mathcal{E})]\!]\eta(\widehat{X}(h(v))) \Rightarrow [\![\mathcal{E}]\!]\eta(X(v)) \Rightarrow [\![\mathcal{F}^{\sqcup}(\mathcal{E})]\!]\eta(\widehat{X}(h(v)))$$

We illustrate the practical implications of the above theorem through an (academic) model checking problem on an infinite state space.

Example 4. Consider the infinite labelled transition system $M = \langle \mathbb{N}, \{a, b, c\}, \rightarrow \rangle$, where $\rightarrow \subseteq \mathbb{N} \times \{a, b, c\} \times \mathbb{N}$ is defined as the least set satisfying:

- for all $n \in \mathbb{N}$ satisfying $n < 10$ we have $n \xrightarrow{a} n + 1$.
- for all $m, n \in \mathbb{N}$ satisfying $n \geq 10$ we have $n \xrightarrow{b} m$.
- for all $n \in \mathbb{N}$ satisfying $n \leq 5$ we have $n \xrightarrow{c} n + 1$.

Note that such infinite labelled transition systems can be specified concisely using (process algebraic) languages such as mCRL2 or LOTOS. Consider the μ-calculus formula ϕ: $\nu X.\mu Y.\langle b \rangle X \vee \langle a \rangle Y$, asserting that there is an a, b-path along which action b is executed infinitely often. Following the (automated) translation described in, *e.g.* [14], we obtain the PBES $\mathcal{E}$ given below; we have, for all $v \in \mathbb{N}$, $X(v)$ is true *iff* $M, v \models \phi$.

$$\big(\nu X(n{:}\mathbb{N}) = Y(n)\big)\big(\mu Y(n{:}\mathbb{N}) = (n \geq 10 \wedge \exists m{:}\mathbb{N}.Y(m)) \vee (n < 10 \wedge X(n+1))\big)$$

Note that there is currently no technique to solve the PBES directly. Let $h{:}\mathbb{N} \rightarrow \mathbb{B}$ be defined as $h(n) = \top$ iff $n < 10$. The under-approximation $\mathcal{F}^{\sqcap}(\mathcal{E})$ is given below:

$$\begin{aligned}
(\nu \widehat{X}(\widehat{n}{:}\mathbb{B}) &= \forall n{:}\mathbb{N}.\ h(n) \neq \widehat{n} \vee \widehat{Y}(h(n))) \\
(\mu \widehat{Y}(\widehat{n}{:}\mathbb{B}) &= ((\forall n{:}\mathbb{N}.\ h(n) \neq \widehat{n} \vee n \geq 10) \wedge \\
&\qquad (\exists \widehat{m}{:}\mathbb{B}.\ (\forall n, m{:}\mathbb{N}.\ h(n) \neq \widehat{n} \vee h(m) \neq \widehat{m} \vee \widehat{Y}(h(m))))) \vee \\
&\qquad ((\forall n{:}\mathbb{N}.\ h(n) \neq \widehat{n} \vee n < 10) \wedge (\forall n{:}\mathbb{N}.\ h(n) \neq \widehat{n} \vee \widehat{X}(h(n+1)))))
\end{aligned}$$

Instantiating $\mathcal{F}^{\sqcap}(\mathcal{E})$ into a Boolean equation system yields the following result:

$$(\nu \widehat{X}_{\top} = \widehat{Y}_{\top})\ (\nu \widehat{X}_{\bot} = \widehat{Y}_{\bot})\ (\mu \widehat{Y}_{\top} = \widehat{X}_{\top} \wedge \widehat{X}_{\bot})\ (\mu \widehat{Y}_{\bot} = \widehat{Y}_{\top} \vee \widehat{Y}_{\bot})$$

The solution to this equation system is $\top$ for all equations. By the instantiation, $\widehat{X}_b = \widehat{X}(b)$ for $b \in \mathbb{B}$; by Theorem 7, we find that the solution to $\widehat{X}(h(v))$ in $\mathcal{F}^{\sqcap}(\mathcal{E})$ implies the solution to $X(v)$ in $\mathcal{E}$ for all $v \in \mathbb{N}$. By the correspondence of the solution of $\mathcal{E}$ and the problem $M, v \models \phi$, we find that $M, v \models \phi$ for every state $v \in \mathbb{N}$ of M. □

It is noteworthy that, irrespective of the problems encoded by the PBESs, our abstraction mechanism is sound; *e.g.*, it is suited for model checking problems stemming from the *full* μ-calculus. This starkly contrasts most works employing simulation relations on processes, as (unless one imposes stricter requirements on the abstractions, or moves from labelled transition systems to *modal* labelled transition systems) only the universal, resp. existential, modal μ-calculus fragments are preserved, resp. reflected [8].

To illustrate this, a slightly finer-grained abstraction function h than the one used in the above example, suffices to over-approximate the PBES resulting from the encoding of the problem $M, v \models \mu Y.\ \langle c \rangle \nu X.([a]X \wedge [b]Y)$, still giving a negative answer to the model checking problem for all states $v \in \mathbb{N}$ of M. The latter modal μ-calculus formula is neither part of the universal fragment of the μ-calculus, nor its existential fragment.

7 Future Work

The notion of a consistent consequence, together with the theory we developed in this paper, open up new avenues for research on manipulations for BESs and PBESs. We plan to investigate specialisations of the abstraction mechanisms we introduced in Section 6, such as the use of Galois connections rather than functions. Moreover, we believe it would be worthwhile to investigate the use of CEGAR techniques for automating the construction of proper abstractions.

Acknowledgements. We would like to thank Bas Luttik (TU/e) and Herman Geuvers (Radboud University Nijmegen) for their useful comments and suggestions.

References

1. Arnold, A., Crubille, P.: A linear algorithm to solve fixed-point equations on transition systems. Information Processing Letters 20(1), 57–66 (1988)
2. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. 16(5), 1512–1542 (1994)
3. Delgrande, J.P., Gupta, A.: Two results in negation-free logic. Applied Mathematics Letters 6(6), 79–83 (1993)
4. Etessami, K., Wilke, T., Schuller, R.A.: Fair simulation relations, parity games, and state space reduction for büchi automata. SIAM J. Comput. 34(5), 1159–1175 (2005)
5. Fritz, C., Wilke, T.: Simulation Relations for Alternating Parity Automata and Parity Games. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 59–70. Springer, Heidelberg (2006)
6. Groote, J.F., Willemse, T.A.C.: Parameterised boolean equation systems. Theor. Comput. Sci. 343(3), 332–369 (2005)
7. Keinänen, M.: Techniques for Solving Boolean Equation Systems. PhD thesis, Helsinki University of Technology (2006)
8. Loiseaux, C., Graf, S., Sifakis, J., Bouajjani, A., Bensalem, S.: Property preserving abstractions for the verification of concurrent systems. Formal Methods in System Design 6(1), 11–44 (1995)
9. Mader, A.: Verification of Modal Properties Using Boolean Equation Systems. PhD thesis, Technische Universität München (1997)
10. Mateescu, R.: Vérification des propriétés temporelles des programmes parallèles. PhD thesis, Institut National Polytechnique de Grenoble (1998)
11. Mateescu, R.: A Generic On-the-Fly Solver for Alternation-Free Boolean Equation Systems. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 81–96. Springer, Heidelberg (2003)
12. McNaughton, R.: Infinite games played on finite graphs. Annals of Pure and Applied Logic 65(2), 149–184 (1993)
13. Orzan, S., Wesselink, J.W., Willemse, T.A.C.: Static Analysis Techniques for Parameterised Boolean Equation Systems. In: Kowalewski, S., Philippou, A. (eds.) TACAS 2009. LNCS, vol. 5505, pp. 230–245. Springer, Heidelberg (2009)
14. Willemse, T.A.C.: Consistent Correlations for Parameterised Boolean Equation Systems with Applications in Correctness Proofs for Manipulations. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 584–598. Springer, Heidelberg (2010)
15. Zhang, D., Cleaveland, R.: Fast Generic Model-Checking for Data-Based Systems. In: Wang, F. (ed.) FORTE 2005. LNCS, vol. 3731, pp. 83–97. Springer, Heidelberg (2005)

4-Coloring H-Free Graphs When H Is Small*

Petr A. Golovach, Daniël Paulusma, and Jian Song

School of Engineering and Computing Sciences, Durham University,
Science Laboratories, South Road, Durham DH1 3LE, United Kingdom
{petr.golovach,daniel.paulusma,jian.song}@durham.ac.uk

Abstract. The k-Coloring problem is to test whether a graph can be colored with at most k colors such that no two adjacent vertices receive the same color. If a graph G does not contain a graph H as an induced subgraph, then G is called H-free. For any fixed graph H on at most 6 vertices, it is known that 3-Coloring is polynomial-time solvable on H-free graphs whenever H is a linear forest and NP-complete otherwise. By solving the missing case $P_2 + P_3$, we prove the same result for 4-Coloring provided that H is a fixed graph on at most 5 vertices.

1 Introduction

Graph coloring involves the labeling of the vertices of some given graph by integers called colors such that no two adjacent vertices receive the same color. The corresponding k-Coloring problem is to decide whether a graph can be colored with at most k colors. Due to the fact that k-Coloring is NP-complete for any fixed $k \geq 3$, there has been considerable interest in studying its complexity when restricted to certain graph classes. One of the most well-known results in this respect is due to Grötschel, Lovász, and Schrijver [11] who show that k-Coloring is polynomial-time solvable for perfect graphs. More information on this classic result and on the general motivation, background and related work on coloring problems restricted to special graph classes can be found in several surveys [21,24] on this topic.

We continue the study of the computational complexity of the k-Coloring problem restricted to graph classes defined by one or more forbidden induced subgraphs. This problem has been studied in many papers by different groups of researchers [3,4,5,6,7,8,10,12,14,15,16,17,19,20,25]. Before we summarize these results and explain our new results, we first state the necessary terminology and notations.

Terminology. We only consider finite undirected graphs G with no loops and no multiple edges. We refer to the textbook by Bondy and Murty [2] for any undefined graph terminology. We write $G[U]$ to denote the subgraph of G *induced by* the vertices in U, i.e., the subgraph of G with vertex set U and an edge between two vertices $u, v \in U$ whenever $uv \in E$.

* This work has been supported by EPSRC (EP/G043434/1).

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 289–300, 2012.

The graphs P_n and C_n denote the path and cycle on n vertices, respectively. The disjoint union of two graphs G and H is denoted $G + H$, and the disjoint union of r copies of G is denoted rG. A *linear forest* is the disjoint union of a collection of paths. Let $\{H_1, \ldots, H_p\}$ be a set of graphs. We say that a graph G is $(H_1, \ldots, H_p)$*-free* if G has no induced subgraph isomorphic to a graph in $\{H_1, \ldots, H_p\}$; if $p = 1$, we sometimes write H_1-free instead of (H_1)-free.

A *(vertex) coloring* of a graph $G = (V, E)$ is a mapping $c : V \to \{1, 2, \ldots\}$ such that $c(u) \neq c(v)$ whenever $uv \in E$. Here, $c(u)$ is referred to as the *color* of u. A k*-coloring* of G is a coloring c of G with $c(V) \subseteq \{1, \ldots, k\}$. Here, we used the notation $c(U) = \{c(u) \mid u \in U\}$ for $U \subseteq V$. If G has a k-coloring, then G is called k*-colorable*. Recall that the problem k-Coloring is to decide whether a given graph admits a k-coloring. Here, k is *fixed*, i.e., not part of the input. If k is part of the input then we denote the problem as Coloring. The optimization version of this problem is to determine the *chromatic number* of a graph G, i.e., the smallest k such that G has a k-coloring.

Related work. Král', Kratochvíl, Tuza and Woeginger [16] completely determined the computational complexity of Coloring for graph classes characterized by one forbidden induced subgraph. They achieved the following dichotomy.

Theorem 1 ([16]). *Let H be a fixed graph. If H is a (not necessarily proper) induced subgraph of P_4 or of $P_1 + P_3$, then* Coloring *can be solved in polynomial time for H-free graphs; otherwise it is* NP*-complete for H-free graphs.*

The computational complexity of Coloring for $\mathcal{H}$-free graphs where $\mathcal{H}$ is a family of two graphs is still open, although several partial results are known, e.g., Král' et al. [16] also showed that Coloring is NP-complete for (C_3, H)-free graphs whenever H is a fixed graph containing at least one cycle. This work has been extended by Schindl [22]. Maffray and Preissmann [19] showed that Coloring is NP-complete for $(C_3, K_{1,5})$-free graphs, where $K_{1,5}$ is the 6-vertex star. Broersma et al. [5] showed that Coloring is polynomial-time solvable for $(C_3, 2P_3)$-free graphs, hereby completing a study of Dabrowski et al. [8] who considered the Coloring problem restricted to (C_3, H)-free graphs for graphs H on at most six vertices.

We focus on the computational complexity of the k-Coloring problem for H-free graphs. Kamiński and Lozin [14] showed that for any $k \geq 3$, the k-Coloring problem is NP-complete for the class of graphs of girth (the length of a shortest induced cycle) at least p for any fixed $p \geq 3$. Their result has the following immediate consequence.

Theorem 2. *For any $k \geq 3$, the k-*Coloring *problem is* NP*-complete for the class of H-free graphs whenever H contains a cycle.*

Holyer [13] showed that 3-Coloring is NP-complete on line graphs. Later, Leven and Galil [18] extended this result by showing that k-Coloring is also NP-complete on line graphs for $k \geq 4$. Because line graphs are claw-free, i.e., they have no induced $K_{1,3}$, these two results together have the following consequence.

Theorem 3. *For any $k \geq 3$, the k-Coloring problem is NP-complete for the class of H-free graphs whenever H is a forest with a vertex of degree at least 3.*

Due to Theorems 2 and 3, only the case in which H is a linear forest remains. We first consider the case when H is a path. Hoàng et al. [12] showed that for any $k \geq 1$, the k-Coloring problem can be solved in polynomial time for P_5-free graphs. Randerath and Schiermeyer [20] showed that 3-Coloring can be solved in polynomial time for P_6-free graphs. It is also known that 4-Coloring is NP-complete for P_8-free graphs [4] and that 6-Coloring is NP-complete for P_7-free graphs [3].

We now discuss the case when H is a linear forest that is the disjoint union of two or more paths. Combining a result from Balas and Yu [1] on the maximal number of independent sets in an sP_2-free graph and a result from Tsukiyama et al. [23] on the enumeration of such sets leads to the known result that k-Coloring is polynomial-time solvable on sP_2-free graphs for any two integers k and s. Broersma et al. [4] extended the aforementioned result of Randerath and Schiermeyer [20] by showing that 3-Coloring is polynomial-time solvable for H-free graphs if H is a linear forest with $|V_H| \leq 6$ or $H = sP_3$ for any integer s. They also observed that 3-Coloring is polynomial-time solvable for $(P_1 + H)$-free graphs whenever this problem is polynomial-time solvable for H-free graphs. Couturier et al. [7] extended the aforementioned result of Hoàng et al. [12] by proving that for any fixed integers k and r, the k-Coloring problem can be solved in polynomial time for $(rP_1 + P_5)$-free graphs. All these positive results are summarized in Theorems 4 and 5 if we keep in mind that k-Coloring is polynomial-time solvable on H'-free graphs whenever it is so on H-free graphs for some graph H containing H' as an induced subgraph.

Theorem 4. *The 3-Coloring problem can be solved in polynomial time for H-free graphs if*

- $H = rP_1 + P_2 + P_4$ *for any* $r \geq 0$
- $H = rP_1 + P_6$ *for any* $r \geq 0$
- $H = sP_3$ *for any* $s \geq 0$.

Theorem 5. *For any $k \geq 4$, the k-Coloring problem can be solved in polynomial time for H-free graphs if*

- $H = rP_1 + P_5$ *for any* $r \geq 0$
- $H = sP_2$ *for any* $s \geq 0$.

Our new result. Theorems 2–4 imply that for any fixed graph H on at most 6 vertices, 3-Coloring is polynomial-time solvable on H-free graphs whenever H is a linear forest and NP-complete otherwise. We prove the following result.

Theorem 6. *For any fixed graph H on at most 5 vertices, 4-Coloring is polynomial-time solvable on H-free graphs whenever H is a linear forest and NP-complete otherwise.*

Theorems 2, 3 and 5 imply that the only missing case is when $H = P_2 + P_3$. We present a polynomial-time algorithm for this case in Section 3. The correctness proof of this algorithm uses some known structural and algorithmic results stated in Section 2.

Future work. Whether k-Coloring is polynomial-time solvable on $(P_2 + P_3)$-free graphs for $k \geq 5$ is an open problem. Our techniques for solving 4-Coloring on $(P_2 + P_3)$-free graphs seem hard to generalize to higher values of k.

2 Preliminaries

In order to proceed we must slightly generalize the coloring concept as follows. A *list-assignment* of a graph $G = (V, E)$ is a function L that assigns a list $L(u)$ of so-called *admissible* colors to each $u \in V$. If $L(u) \subseteq \{1, \ldots, k\}$ for $u \in V$, then L is also called a *k-list-assignment*. Equivalently, L is a k-list-assignment if $|\bigcup_{u \in V} L(u)| \leq k$. We say that a coloring $c : V \rightarrow \{1, 2, \ldots\}$ *respects* L if $c(u) \in L(u)$ for all $u \in V$. For a fixed integer k, the List k-Coloring problem has as input a graph G with a k-list-assignment L and asks whether G has a coloring that respects L. If $|L(u)| = 1$ for every vertex u of some subset $W \subseteq V$ and $L(u) = \{1, \ldots, k\}$ for $u \in V \setminus W$, then we obtain the k-Precoloring Extension problem. In that case we also say that we want to *extend* the precoloring on W to a k-coloring of G.

We will frequently use the following observation, the proof of which follows from the fact that the problem in this case can be modeled and solved as an instance of the 2-Satisfiability problem. This approach has been introduced by Edwards [9] and is folklore now.

Lemma 1 ([9]). *Let G be a graph in which every vertex has a list of admissible colors of size at most 2. Then checking whether G has a coloring respecting these lists is solvable in polynomial time.*

Let $G = (V, E)$ be a graph. For a subset $U \subseteq V$ we define $N_G(U) = \{v \in V \setminus U \mid uv \in E \text{ for some } u \in U\}$; note that $N(\emptyset) = \emptyset$. A set $D \subseteq V$ *dominates* a set $S \subseteq V$ if $S \subseteq D \cup N_G(D)$; if $S = V$ then we say that D is a *dominating set* of G. The next lemma follows from Lemma 7 in the paper of Broersma et al. [4].

Lemma 2. *Let $G = (V, E)$ be a $(P_2 + P_3)$-free graph of minimum degree at least 4. If G has a 4-coloring, then G contains a dominating set D of size at most 39.*

Note that Lemma 2 involves a minimum degree condition. However, we can easily get around this by applying the following well-known procedure on a graph $G = (V, E)$. Remove all vertices of V with degree at most 3 from G. Propagate this until we obtain a graph G^* of minimum degree at least 4; note that G^* may be the empty graph. We observe the following straightforward result, see e.g. Broersma et al. [4] for a proof.

Lemma 3. *Let G be a graph. Then G has a 4-coloring if and only if G^* has a 4-coloring. Moreover, G^* can be obtained in polynomial time.*

The following result is slightly stronger than the corresponding result stated in Theorem 4. We can show this result by using exactly the same proof as the proof for 3-Precoloring Extension for this graph class by Broersma et al. [4].

Lemma 4. *The* List 3-Coloring *problem can be solved in polynomial time on sP_3-free graphs for any fixed integer $s \geq 1$.*

Due to Lemma 4 and the fact that every (P_2+P_3)-free graph is $2P_3$-free we obtain the next lemma, which we need for the correctness proof of our algorithm.

Lemma 5. *The* List 3-Coloring *problem can be solved in polynomial time for the class of $(P_2 + P_3)$-free graphs.*

We also need the following lemma, which follows immediately from Lemma 5.

Lemma 6. *Let $G = (V, E)$ be a $(P_2 + P_3)$-free graph. Then a partition of V into three (possibly empty) independent sets I_1, I_2, I_3 can be found in polynomial time if it exists.*

3 The Algorithm

Let G be a (P_2+P_3)-free graph that is an instance of 4-Coloring. By Lemma 3 we may assume that G has minimum degree at least 4. We also assume that each vertex u has been assigned an initial list $L_0(u) = \{1, 2, 3, 4\}$ of admissible colors. If at some moment we color a vertex u with a certain color, then we may remove this color from the list of every neighbor of u. This is what we call *updating* the list assignment. Also, when coloring a vertex, say with color i, then we set its list of admissible colors to $\{i\}$.

Outline. Our algorithm is a branching algorithm. The main idea is to obtain in polynomial time a polynomial-bounded set $\mathcal{L}$ of list assignments for G that have the following two properties. First, G has a 4-coloring if and only if G has a coloring that respects a list assignment in $\mathcal{L}$. Second, for every list assignment $L \in \mathcal{L}$, we either have that all its lists have size at most two or else that the union of its lists that contain at least 2 colors has size 3; in the first case we can use Lemma 1, and in the second case we can use Lemma 5 after removing all vertices with a single color in their list from the graph and updating the list assignment if necessary. Because we obtain $\mathcal{L}$ in polynomial time and its size is bounded by a polynomial, this means that the total running time of our algorithm is polynomial.

Our algorithm consists of two phases. At the end of Phase 1, we either have found that G has no 4-coloring or we have obtained a set $\mathcal{L}$ of list assignments, for which we will prove the desired properties specified in the outline; initially $\mathcal{L} = \emptyset$. In Phase 2 we consider the list assignments of $\mathcal{L}$ one by one to determine whether G has a coloring respecting at least one of them. During an execution of the algorithm we are not always so bothered if two adjacent vertices are colored

alike, e.g., when we assign a color i to a vertex without explicitly checking if i is in its list; this will be spotted in Phase 2.

Phase 1. Determining the set $\mathcal{L}$.

Step 1. Check if G has a dominating set of size at most 39. If such a set does not exist, then output `No`. Otherwise, let D be such a dominating set.

Step 2. Check if $G[D]$ is 4-colorable. If not, then output `No`.

Assume that $G[D]$ is 4-colorable and do the following for every 4-coloring of $G[D]$.

Step 3. For $i = 1, \ldots, 4$, let $D_i \subseteq D$ be the subset of vertices with color i, and let $F_i = G[V \setminus (D \cup N_G(D_i)]$. Note that $V_{F_h} \cap V_{F_i} \neq \emptyset$ is possible for $h \neq i$. For $i = 1, \ldots, 4$ check whether $N_G(D_i) \setminus D$ can be partitioned into three independent sets, where one or more of such sets are allowed to be empty; note that, in particular, all three sets are empty if $N_G(D_i) \setminus D = \emptyset$. If such a partition into such three sets does not exist for some i, then consider a different 4-coloring of $G[D]$ unless all 4-colorings of $G[D]$ have already been processed; in that case output `No`. Otherwise, let I_1^i, I_2^i, I_3^i be such a partition for $i = 1, \ldots, 4$.

Step 4. For $i = 1, \ldots, 4$, give each isolated vertex in F_i color i unless it already received a color h for some $h \leq i - 1$. Afterwards, update the list assignment.

For $i = 1, \ldots, 4$, let F_i' be the graph obtained from F_i by removing all isolated vertices.

Step 5. If there exists a graph F_i' that consists of at most 2 vertices for some $1 \leq i \leq 4$, then color the vertices of F_i' according to their list in every possible way, update the list assignment and put the resulting list assignment in $\mathcal{L}$. Then, after processing each possible coloring of the vertices of F_i' in this way, start with Phase 2 of the algorithm.

From now on assume that F_i' consists of at least three vertices for $i = 1, \ldots, 4$.

Step 6. For $i = 1, \ldots, 4$, check if F_i' is connected and bipartite. If so, then give the vertices of one partition class color i. Consider both possibilities. Each time, also update the list assignment, put the resulting list assignment in $\mathcal{L}$ and restore the lists to the situation at the end of Step 5, i.e., before applying Step 6. Afterwards consider F_{i+1}' or continue with Step 7 if $i = 4$.

Step 7. For $i = 1, \ldots, 4$ and $j = 1, \ldots, 3$, if $I_j^i \neq \emptyset$, then do as follows. Find a vertex $a_j^i \in D_i$ that is adjacent to as many vertices of I_j^i as possible. Here we allow $a_j^i = a_{j'}^i$ for some $j \neq j'$. Give every vertex of I_j^i that is not adjacent to a_j^i a color from its list. Consider each possible coloring of such vertices.

For $i = 1, \ldots, 4$ and $j = 1, \ldots 3$, let $\tilde{I}_j^i = I_j^i \cap N_G(a_j^i)$ if $I_j^i \neq \emptyset$, and let $\tilde{I}_j^i = \emptyset$ otherwise.

Step 8. For $i = 1, \ldots, 4$, do as follows. Check if $\tilde{I}_1^i \cup \tilde{I}_2^i \cup \tilde{I}_3^j \neq \emptyset$ and if F_i' is a disjoint union of edges. If both conditions are satisfied, then choose a nonempty

set $\tilde{I}_j^i$ in every possible way and use color i on every vertex of F_i' adjacent to all but at most two vertices of $\tilde{I}_j^i$ unless its neighbor in F_i' is already colored i; make an arbitrary choice if there is a choice between two end-vertices. Give every uncolored vertex in F_i' that is not adjacent to a vertex with color i a color from its list in every possible way. Each time update the list assignment, put the resulting list assignment in $\mathcal{L}$ and restore the lists to the situation at the end of Step 7, i.e., before applying Step 8. Note that the branching is done over the set of indices j with $\tilde{I}_j^i \neq \emptyset$ and all ways to color the uncolored vertices in F_i that are not adjacent to a vertex with color i. Afterwards consider F_{i+1}' or continue with Step 9 if $i = 4$.

Step 9. We consider all partitions of $\{1, 2, 3, 4\}$ into two sets M_1 and M_2 subject to the condition that $i \in M_1$ if F_i' is not a disjoint union of (at least two) edges. For $i = 1, \ldots, 4$ do as follows. If $i \in M_1$, then choose an edge $e^i = u^i v^i$ in F_i'. If $i \in M_2$, then for every $1 \leq j \leq 3$ choose a vertex $u_j^i \in F_i'$ that is adjacent to all but at most two vertices of $\tilde{I}_j^i$. We allow $e^i = e^{i'}$ for some $i \neq i'$ and also $u_j^i = u_{j'}^{i'}$ for some $(i, j) \neq (i', j')$. Color the end-vertices of every chosen edge and every chosen vertex with a color from their list without using color i. For all $i \in M_1$, give every vertex in $\tilde{I}_1^i \cup \tilde{I}_2^i \cup \tilde{I}_3^i$ that is neither adjacent to u^i nor to v^i a color from its list. For all $i \in M_2$ and all $1 \leq j \leq 3$, give every vertex in $\tilde{I}_j^i$ that is not adjacent to u_j^i a color from its list. Afterwards update the list assignment and put the resulting list assignment in $\mathcal{L}$. Then restore all lists to the situation at the end of Step 8, i.e., before applying Step 9. Before considering another partition of $\{1, \ldots, 4\}$, repeat Step 9 as many times as possible for the same partition M_1, M_2 until all possibilities of chosen edges, chosen vertices and chosen colorings have been considered. Afterwards continue with Phase 2.

Phase 2. Determining if G has a coloring that respects a list assignment in $\mathcal{L}$.

For each $L \in \mathcal{L}$ do as follows. Check if there exist two adjacent vertices that each have a list of exactly one admissible color that is the same for both vertices. If this happens, then stop considering L. Otherwise, remove all vertices from G that have a list of size 1. If the lists of all remaining vertices each have size at most two, then apply Lemma 1 in order to determine if G has a coloring respecting L. If so, then output YES. If the union of the list of all remaining vertices has size 3, then use Lemma 5 in order to determine if G has a coloring respecting L. If so, then output YES. Otherwise, if G has no coloring respecting L for all $L \in \mathcal{L}$, then output NO.

We prove the correctness of our algorithm and analyze its running time in Theorem 7. For doing this, we need the following lemmas.

Lemma 7. *For $i = 1, \ldots, 4$ and $j = 1, \ldots, 3$, the number of vertices of I_j^i that is not adjacent to a_j^i in Step 7 of Phase 1 is at most 38.*

Proof. In order to obtain a contradiction, suppose that there exists a pair of indices (i, j) such that $a_0 = a_j^i$ is not adjacent to 39 vertices b_h for $h = 1, \ldots, 39$

in I_j^i. Consider b_1. Because b_1 is in $N(D_i)$, it has a neighbor $a_1 \in D_i$. Suppose that a_1 is not adjacent to two vertices c and c' of I_j^i that are neighbors of a_0. Then G contains an induced $P_2 + P_3$, where $P_2 = a_1b_1$ and $P_3 = ca_0c'$. This is not possible. Hence, a_1 is adjacent to all neighbors of I_j^i except to one vertex b_0, as otherwise a_1 has more neighbors in I_j^i than a_0, contradicting our choice of a_0. For the same reason, a_1 cannot be adjacent to a vertex b_h with $2 \leq h \leq 39$. By the same arguments, we find that D_i contains vertices $a_2, \ldots, a_{39}$, where each a_i is adjacent to b_i and to all neighbors of a_0 in I_j^i except to one such neighbor. However, then $|D| \geq |D_i| \geq 40$, which is not possible. This completes the proof of Lemma 7. □

Lemma 8. *For each edge uv in each F_i', there exists at most one vertex in $\tilde{I}_j^i$ that is neither adjacent to u nor to v.*

Proof. Suppose that there exists a pair of indices (i, j) such that $\tilde{I}_j^i$ contains two vertices b and b' that are both neither adjacent to u nor to v. Then G contains an induced $P_2 + P_3$, where $P_2 = uv$ and $P_3 = ba_j^ib'$. This is not possible and we have proven Lemma 8. □

Lemma 9. *For all $1 \leq i \leq 4$ and all $1 \leq j \leq 3$, if F_i' is a disjoint union of at least two edges, then every edge of F_i' except at most one edge has at least one end-vertex that is adjacent to all but at most two vertices of $\tilde{I}_j^i$.*

Proof. Suppose that F_i' is a disjoint union of at least two edges. Let st and uv be two edges in F_i'. We claim that s is adjacent to all but at most one neighbors of u in $\tilde{I}_j^i$, or else that u is adjacent to all but at most one neighbors of s in $\tilde{I}_j^i$. In order to obtain a contradiction, suppose that s is not adjacent to two vertices b, b' in $\tilde{I}_j^i$ that are each neighbors of u, and that u is not adjacent to two vertices c, c' in $\tilde{I}_j^i$ that are each neighbors of s. Then G contains an induced $P_2 + P_3$, e.g., $P_2 = bu$ and $P_3 = csc'$. This is not possible. Hence, we may assume without loss of generality that s is adjacent to all but at most one neighbors of u in $\tilde{I}_j^i$.

By the same arguments as above, we find that t is adjacent to all but at most one neighbors of u in $\tilde{I}_j^i$, or else that u is adjacent to all but at most one neighbors of t in $\tilde{I}_j^i$. We observe that $\{s, t\}$ dominates all but at most one vertices of $\tilde{I}_j^i$, due to Lemma 8. Consequently, in the first case, u, and in the second case, t is adjacent to all but at most two vertices of $\tilde{I}_j^i$. Hence, in both cases we find a vertex of $\{s, t, u, v\}$ that is adjacent to all but at most two vertices of $\tilde{I}_j^i$, as desired. □

Lemma 10. *If F_i' is a disjoint union of edges, then in Step 8 of Phase 1 the number of vertices in F_i' that are not adjacent to a vertex with color i is at most 2.*

Proof. Note that each F_i' contains at least two edges, as otherwise the algorithm would have gone to Phase 2 after Step 5. By Lemma 9, every edge of F_i' except for at most one edge contains a vertex that is adjacent to all but at most two vertices of $\tilde{I}_j^i$. Hence, at most 2 vertices of a graph F_i' in Step 8 are not adjacent to a vertex with color i. □

Lemma 11. *For all $i \in M_1$ and all $1 \leq j \leq 3$, the number of vertices of $\tilde{I}_j^i$ that is neither adjacent to u^i nor to v^j in Step 9 of Phase 1 is at most one.*

Proof. This follows immediately from Lemma 8. □

Lemma 12. *For all $i \in M_2$ and all $1 \leq j \leq 3$, the number of vertices of $\tilde{I}_j^i$ not adjacent to u_j^i in Step 9 of Phase 1 is at most two.*

Proof. By definition, $i \in M_2$ implies that F_i' is a disjoint union of edges. Note that F_i' contains at least two edges, as otherwise the algorithm would have gone to Phase 2 after Step 5. Then we get the desired result immediately from Lemma 9. □

Finally we need the following lemma, the proof of which we omitted due to space restrictions.

Lemma 13. *For every list assignment of $\mathcal{L}$ in Phase 2, either all its lists have size at most 2, or the union of the lists that contain at least 2 colors has size 3.*

Theorem 7. *The* 4-Coloring *problem can be solved in polynomial time for $(P_2 + P_3)$-free graphs.*

Proof. Let $G = (V, E)$ be a $(P_2 + P_3)$-free graph. Recall that we may assume that G has minimum degree at least 4 due to Lemma 3.

Correctness. We start with proving that our algorithm is correct, i.e., that G has a 4-coloring if and only if its output is Yes. First suppose that G has a 4-coloring c. Lemma 2 tells us that G has a dominating set of size at most 39. Hence, our algorithm will find such a dominating set in Step 1. Because G is 4-colorable, $G[D]$ is 4-colorable. Hence, the algorithm does not output `No` in Step 2. Instead it considers each 4-coloring of $G[D]$ including the 4-coloring c' of $G[D]$ with $c'(u) = c(u)$ for each $u \in D$.

In Step 3, our algorithm checks if some sets I_1^i, I_2^i, I_3^i exist for $i = 1, \ldots, 4$. Because all the vertices in $N_G(D_i) \setminus D$ are adjacent to a vertex in D_i, i.e., to a vertex v with color $c(v) = i$ and because c is a 4-coloring, we find that the restriction of c to the vertices of $N_G(D_i) \setminus D$ is a 3-coloring. Hence, sets I_1^i, I_2^i, I_3^i exist for $i = 1, \ldots, 4$, and the algorithm will find them; note that the algorithm may find different sets than the ones induced by c. In Step 4, the algorithm assigns each isolated vertex in F_i color i unless it assigned such a vertex already a color earlier on in this step. Suppose that u is such a vertex that got assigned color i while $c(u) \neq i$. Then we may redefine c by setting $c(u) := i$. We may safely do so, because u is only adjacent to vertices that are adjacent to a vertex in D_i, i.e., to a vertex with color i. Hence, no neighbor v of u has color $c(v) = i$. If in Step 5 there exists a graph F_i' that consists of at most 2 vertices for some $1 \leq i \leq 4$, our algorithm considers each possible coloring of them including the coloring of F_i' that corresponds to c. Hence, it goes to Phase 2 with a set $\mathcal{L}$ of list assignments that include a list assignment L that is respected by c. This means that at the moment the algorithm considers L in Phase 2, which it will

due to Lemma 13, it will find that there exists a coloring of G respecting L (as c is such a coloring). As a result, it will output `Yes`.

If every F'_i has at least 3 vertices, then the algorithm performs Step 6 of Phase 1. Suppose that F'_i is connected and bipartite for some $1 \leq i \leq 4$. If $c(u) = i$ for every vertex u in one partition class of F'_i, then the algorithm will put a list assignment L that is respected by c in $\mathcal{L}$, and in Phase 2 it will output YES upon considering L; it will consider L due to Lemma 13. If no F'_i is a bipartite connected graph, or if c does not color every vertex u in a partition class of any bipartite connected F'_i with color i, then let us consider Step 7 and further. In Step 7, the algorithm assigns every vertex of every I^i_j not adjacent to the associated a^i_j a color from its list. Because each possible coloring is considered, the restriction of c to such vertices is included.

Suppose that there exists a graph F'_i that is a disjoint union of at least two edges such that every edge except for at most one edge contains a vertex with color i according to c. Then, in Step 8, the algorithm may put a list assignment L in $\mathcal{L}$ that is respected by c. As a result of Lemma 13 the algorithm will consider L in Phase 2 and will output `Yes`. However, this may not happen, because the algorithm can color different vertices of F'_i with color i than the ones colored with color i according to c. We take this into account when we consider the final step, which is Step 9.

In Step 9, we consider partitions M_1, M_2 of $\{1, \ldots 4\}$. We must show that there exists a partition M_1, M_2 for which the algorithm can color according to c. Let $1 \leq i \leq 4$. Assume that there is an edge uv in F'_i with $c(u) \neq i$ and $c(v) \neq i$. In one of the branches in Step 9 the algorithm considers $i \in M_1$ and the edge uv and colors u, v and every vertex in $\tilde{I}^i_1 \cup \tilde{I}^i_2 \cup \tilde{I}^i_3$ that is neither adjacent to u^i nor to v^i according to c.

Suppose now that for any edge uv in F'_i, either $c(u) = i$ or $c(v) = i$. We first observe that F'_i must be bipartite, because the vertices with color i and the vertices with a color different from i form a partition of the vertex set of F'_i. If F'_i is connected, then the vertices of one partition class all have color i. Because we already considered this case in Step 6, we find that F'_i is not connected. Then, because G is $(P_2 + P_3)$-free and F'_i contains no isolated vertices, F'_i is a disjoint union of edges. Because F'_i has at least 3 vertices, this means that F'_i has at least 2 edges. Note that $i \in M_2$ is allowed and assume that the algorithm has put $i \in M_2$. We must still show that the algorithm can color according to c. For this purpose we consider two cases for each $1 \leq j \leq 3$.

First suppose that $\tilde{I}^i_j = \emptyset$. Then any vertex in F'_i is trivially adjacent to all but at most two vertices of $\tilde{I}^i_j$. Hence, in one of the branches, the algorithm considers the choice of a vertex u that is adjacent to a vertex v with $c(v) = i$ as the vertex u^i_j. Then it colors u^i_j and every vertex in $\tilde{I}^i_j$ that is not adjacent to u^i_j according to c.

Now suppose that $\tilde{I}^i_j \neq \emptyset$. We would already have considered this situation in Step 8, unless for an edge uv in F'_i such that u and v are adjacent to all but at most two vertices of $\tilde{I}^i_j$, we picked the vertex u with $c(u) \neq i$ to get color i in Step 8. Then, in one of the branches, the algorithm considers the choice of u as

the vertex u_j^i. Then it colors u_j^i and every vertex in $\tilde{I}_j^i$ that is not adjacent to u_j^i according to c.

The above ensures that $\mathcal{L}$ contains a list assignment L that is respected by c. As a result of Lemma 13 the algorithm will consider L in Phase 2 and will output `Yes`.

Now suppose that the output of our algorithm is YES. Note that such an output only occurs in Phase 2. Hence, G has a coloring respecting a list assignment in $\mathcal{L}$. Because every list in every list assignment of $\mathcal{L}$ is a subset of $\{1,2,3,4\}$, this coloring is a 4-coloring of G.

Running time analysis. We prove that Phase 1 can be performed in polynomial time and leads to a set $\mathcal{L}$ of polynomial size. We perform Step 1 in $O(n^{39})$ time by brute force. In Step 2 we find at most $4^{|D|} \leq 4^{39}$ different 4-colorings of $G[D]$. We perform Step 3 in polynomial time by applying Lemma 6 at most four times. We perform Step 4 in linear time; here we need to assign a color to a set of vertices that we can detect in linear time. We perform Step 5 in polynomial time as we must color at most 2 vertices, each of which has a list of size at most 3. If we start Phase 2 immediately after Step 5, then we have a set $\mathcal{L}$ of size at most $4^{39} \cdot 3^2$, as desired. Otherwise, we do as follows. We perform Step 6 in linear time as we only have to consider at most two possibilities for at most one graph F_i'. For $i = 1,\ldots,4$ and $j = 1,\ldots,3$ we determine a required vertex $a_j^i \in D_i$ in Step 7 in polynomial time. By Lemma 7, each a_j^i is adjacent to all but at most 38 vertices of I_j^i, each of which we color in polynomial time in a total number of at most 3^{38} different ways. We perform Step 8 in polynomial time, as we only have to check whether the graphs F_i' are disjoint unions of edges, and then we have to consider at most $(3^2)^4 = 3^8$ different colorings according to Lemma 10. In Step 9 we consider all possible partitions of $\{1,\ldots,4\}$ subject to some further condition, which we can check in polynomial time. For each partition M_1, M_2, we perform Step 9 in polynomial time. For each M_1, M_2, there are at most $p(n) = (2^2 \cdot 3^3 \cdot n^2)^4 \cdot (2 \cdot 3^2 \cdot n)^{12}$ list assignments due to Lemmas 11 and 12, respectively. Because the number of partitions of $\{1,\ldots,4\}$ into M_1, M_2 is at most 2^4, the total number of list assignments obtained in Step 9 is at most $2^4 p(n)$. Hence, if we start Phase 2 after Step 9, then we have a set $\mathcal{L}$ of size at most $4^{39} \cdot (2 + 3^{38}(3^8 + 2^4 p(n)))$, which is polynomial, as desired.

Because $\mathcal{L}$ has polynomial size and we can process every $L \in \mathcal{L}$ in polynomial time either due to Lemma 1 or due to Lemma 5, we can perform Phase 2 in polynomial time. This completes the proof of Theorem 7. □

References

1. Balas, E., Yu, C.S.: On graphs with polynomially solvable maximum-weight clique problem. Networks 19, 247–253 (1989)
2. Bondy, J.A., Murtym, U.S.R.: Graph Theory. Springer Graduate Texts in Mathematics, vol. 244 (2008)
3. Broersma, H.J., Fomin, F.V., Golovach, P.A., Paulusma, D.: Three Complexity Results on Coloring P_k-Free Graphs. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) IWOCA 2009. LNCS, vol. 5874, pp. 95–104. Springer, Heidelberg (2009)

4. Broersma, H.J., Golovach, P.A., Paulusma, D., Song, J.: Updating the complexity status of coloring graphs without a fixed induced linear forest (manuscript), http://www.dur.ac.uk/daniel.paulusma/Papers/Submitted/updating.pdf
5. Broersma H.J., Golovach, P.A., Paulusma, D., Song, J.: Determining the chromatic number of triangle-free $2P_3$-free graphs in polynomial time (manuscript), http://www.dur.ac.uk/daniel.paulusma/Papers/Submitted/2p3.pdf
6. Bruce, D., Hoàng, C.T., Sawada, J.: A Certifying Algorithm for 3-Colorability of P_5-Free Graphs. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 594–604. Springer, Heidelberg (2009)
7. Couturier, J.F., Golovach, P.A., Kratsch, D., Paulusma, D.: List coloring in the absence of a linear forest. In: Proceedings of WG 2011. LNCS (to appear, 2011)
8. Dabrowski, K., Lozin, V., Raman, R., Ries, B.: Colouring Vertices of Triangle-Free Graphs. In: Thilikos, D.M. (ed.) WG 2010. LNCS, vol. 6410, pp. 184–195. Springer, Heidelberg (2010)
9. Edwards, K.: The complexity of coloring problems on dense graphs. Theoret. Comput. Sci. 43, 337–343 (1986)
10. Golovach, P.A., Paulusma, D., Song, J.: Coloring Graphs without Short Cycles and Long Induced Paths. In: Owe, O., Steffen, M., Telle, J.A. (eds.) FCT 2011. LNCS, vol. 6914, pp. 193–204. Springer, Heidelberg (2011)
11. Grötschel, M., Lovász, L., Schrijver, A.: Polynomial algorithms for perfect graphs. Ann. Discrete Math., Topics on Perfect Graphs 21, 325–356 (1984)
12. Hoàng, C.T., Kamiński, M., Lozin, V., Sawada, J., Shu, X.: Deciding k-colorability of P_5-free graphs in polynomial time. Algorithmica 57, 74–81 (2010)
13. Holyer, I.: The NP-completeness of edge-coloring. SIAM J. Comput. 10, 718–720 (1981)
14. Kamiński, M., Lozin, V.V.: Coloring edges and vertices of graphs without short or long cycles. Contributions to Discrete Math. 2, 61–66 (2007)
15. Kamiński, M., Lozin, V.V.: Vertex 3-colorability of Claw-free Graphs. Algorithmic Operations Research 21 (2007)
16. Král', D., Kratochvíl, J., Tuza, Z., Woeginger, G.J.: Complexity of Coloring Graphs without Forbidden Induced Subgraphs. In: Brandstädt, A., Le, V.B. (eds.) WG 2001. LNCS, vol. 2204, pp. 254–262. Springer, Heidelberg (2001)
17. Le, V.B., Randerath, B., Schiermeyer, I.: On the complexity of 4-coloring graphs without long induced paths. Theoret. Comput. Sci. 389, 330–335 (2007)
18. Leven, D., Galil, Z.: NP completeness of finding the chromatic index of regular graphs. Journal of Algorithms 4, 35–44 (1983)
19. Maffray, F., Preissmann, M.: On the NP-completeness of the k-colorability problem for triangle-free graphs. Discrete Math. 162, 313–317 (1996)
20. Randerath, B., Schiermeyer, I.: 3-Colorability $\in$ P for P_6-free graphs. Discrete Appl. Math. 136, 299–313 (2004)
21. Randerath, B., Schiermeyer, I.: Vertex colouring and forbidden subgraphs - a survey. Graphs Combin. 20, 1–40 (2004)
22. Schindl, D.: Some new hereditary classes where graph coloring remains NP-hard. Discrete Math. 295, 197–202 (2005)
23. Tsukiyama, S., Ide, M., Ariyoshi, H., Shirakawa, I.: A new algorithm for generating all the maximal independent sets. SIAM J. Comput. 6, 505–517 (1977)
24. Tuza, Z.: Graph colorings with local restrictions - a survey. Discuss. Math. Graph Theory 17, 161–228 (1997)
25. Woeginger, G.J., Sgall, J.: The complexity of coloring graphs without long induced paths. Acta Cybernet. 15, 107–117 (2001)

Computing q-Gram Non-overlapping Frequencies on SLP Compressed Texts

Keisuke Goto, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda

Department of Informatics, Kyushu University
744 Motooka, Nishiku, Fukuoka 819–0395, Japan
{keisuke.gotou,bannai,inenaga,takeda}@inf.kyushu-u.ac.jp

Abstract. Length-q substrings, or q-grams, can represent important characteristics of text data, and determining the frequencies of all q-grams contained in the data is an important problem with many applications in the field of data mining and machine learning. In this paper, we consider the problem of calculating the *non-overlapping frequencies* of all q-grams in a text given in compressed form, namely, as a straight line program (SLP). We show that the problem can be solved in $O(q^2n)$ time and $O(qn)$ space where n is the size of the SLP. This generalizes and greatly improves previous work (Inenaga & Bannai, 2009) which solved the problem only for $q = 2$ in $O(n^4 \log n)$ time and $O(n^3)$ space.

1 Introduction

In many situations, large-scale text data is first compressed for storage, and then is usually decompressed when it is processed afterwards, where we must again face the size of the data. To circumvent this problem, algorithms that work directly on the compressed representation without explicit decompression have gained attention, especially for the string pattern matching problem [1], and there has been growing interest in what problems can be efficiently solved in this kind of setting [11,13,6,12,7,5,3].

The *non-overlapping occurrence frequency* of a string P in a text string T is defined as the maximum number of non-overlapping occurrences of P in T [2]. In this paper, we consider the problem of computing the non-overlapping occurrence frequencies of *all* q-grams (length-q substrings) occurring in a text T, when the text is given as a *straight line program* (SLP) [8] of size n. An SLP is a context free grammar in the Chomsky normal form that derives a single string. SLPs are a widely accepted abstract model of various text compression schemes, since texts compressed by any grammar-based compression algorithm (e.g. [14,10]) can be represented as SLPs, and those compressed by the LZ-family (e.g. [15,16]) can be quickly transformed to SLPs. Theoretically, the length N of the text represented by an SLP of size n can be as large as $\Theta(2^n)$, and therefore a polynomial time algorithm that runs on an SLP representation is, in the worst case, faster than any algorithm which works on the uncompressed string.

For SLP compressed texts, the problem was first considered in [7], where an algorithm for $q = 2$ running in $O(n^4 \log n)$ time and $O(n^3)$ space was presented.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 301–312, 2012.

However, the algorithm cannot be readily extended to handle $q > 2$. Intuitively, the problem for $q = 2$ is much easier compared to larger values of q, since there is only one way for a 2-gram to overlap, while there can be many ways that a longer q-gram can overlap. In this paper we present the first algorithm for calculating the non-overlapping occurrence frequency of all q-grams, that works for any $q \geq 2$, and runs in $O(q^2 n)$ time and $O(qn)$ space. Not only do we solve a more general problem, but the complexity is greatly improved compared to previous work.

A similar problem for SLPs, where occurrences of q-grams are allowed to overlap, was also considered in [7], where an $O(|\Sigma|^2 n^2)$ time and $O(n^2)$ space algorithm was presented for $q = 2$. A much simpler and efficient $O(qn)$ time and space algorithm for general $q \geq 2$ was recently developed [5]. As is the case with uncompressed strings, ideas from the algorithms allowing overlapping occurrences can be applied *somewhat* to the problem of obtaining non-overlapping occurrence frequencies. However, there are still difficulties that arise from the overlapping of occurrences that must be overcome, i.e., the occurrences of each q-gram can be obtained in the same way, but we must somehow compute their non-overlapping occurrence frequency, which is not a trivial task.

For uncompressed texts, the problem considered in this paper can be solved in $O(|T|)$ time, by applying string indices such as suffix arrays. A similar problem is the *string statistics problem* [2], which asks for the non-overlapping occurrence frequency of a given string P in text string T. The problem can be solved in $O(|P|)$ time for any P, provided that the text is pre-processed in $O(|T| \log |T|)$ time using the sophisticated algorithm of [4]. However, note that the preprocessing requires only $O(|T|)$ time if occurrences are allowed to overlap. This perhaps indicates the intrinsic difficulty that arises when considering overlaps.

2 Preliminaries

2.1 Notation

Let Σ be a finite *alphabet*. An element of Σ^* is called a *string*. The length of a string T is denoted by $|T|$. The empty string ε is a string of length 0, namely, $|\varepsilon| = 0$. A string of length $q > 0$ is called a *q-gram*. The set of q-grams is denoted by Σ^q. For a string $T = XYZ$, X, Y and Z are called a *prefix*, *substring*, and *suffix* of T, respectively. The i-th character of a string T is denoted by $T[i]$ for $1 \leq i \leq |T|$, and the substring of a string T that begins at position i and ends at position j is denoted by $T[i : j]$ for $1 \leq i \leq j \leq |T|$. For convenience, let $T[i : j] = \varepsilon$ if $j < i$. Let T^R denote the reversal of T, namely, $T^R = T[N]T[N-1] \cdots T[1]$, where $N = |T|$.

For any integers i, j such that $i \leq j$, let $[i : j]$ denote the set of consecutive integers from i to j, i.e., $[i : j] = \{i, i+1, \ldots, j\}$.

For an integer i and a set of integers A, let $i \oplus A = \{i + x \mid x \in A\}$ and $i \ominus A = \{i - x \mid x \in A\}$. If $A = \emptyset$, then let $i \oplus A = i \ominus A = \emptyset$. Similarly, for a pair of integers (x, y), let $i \oplus (x, y) = (i + x, i + y)$.

2.2 Occurrences and Frequencies

For any strings T and P, let $Occ(T, P)$ be the set of occurrences of P in T, i.e.,

$$Occ(T, P) = \{k > 0 \mid T[k : k + |P| - 1] = P\}.$$

The number of occurrences of P in T, or the *frequency* of P in T is, $|Occ(T, P)|$. Any two occurrences $k_1, k_2 \in Occ(T, P)$ with $k_1 < k_2$ are said to be *overlapping* if $k_1 + |P| - 1 \geq k_2$. Otherwise, they are said to be *non-overlapping*. The *non-overlapping frequency* $nOcc(T, P)$ of P in T is defined as the size of a largest subset of $Occ(T, P)$ where any two occurrences in the set are non-overlapping. For any strings X, Y, we say that an occurrence i of a string Z in XY, with $|Z| \geq 2$, *crosses* X and Y, if $i \in [|X| - |Z| + 2 : |X|] \cap Occ(XY, Z)$.

For any strings T and P, we define the sets of *right and left priority non-overlapping occurrences* of P in T, respectively, as follows:

$$RnOcc(T, P) = \begin{cases} \emptyset & \text{if } Occ(T, P) = \emptyset, \\ \{i\} \cup RnOcc(T[1 : i - 1], P) & \text{otherwise,} \end{cases}$$

$$LnOcc(T, P) = \begin{cases} \emptyset & \text{if } Occ(T, P) = \emptyset, \\ \{j\} \cup (j+|P|-1) \oplus LnOcc(T[j + |P| : |T|], P) & \text{otherwise,} \end{cases}$$

where $i = \max Occ(T, P)$ and $j = \min Occ(T, P)$. For all $k \in RnOcc(T, P)$, it is trivially said that $RnOcc(T[k : |T|], P) \subseteq RnOcc(T, P)$. It can be said to $LnOcc$ similarly. Note that $RnOcc(T, P) \subseteq Occ(T, P)$, $LnOcc(T, P) \subseteq Occ(T, P)$, and $LnOcc(T, P) = (|T| - |P| + 2) \ominus RnOcc(T^R, P^R)$.

Lemma 1. $nOcc(T, P) = |RnOcc(T, P)| = |LnOcc(T, P)|$

Proof. Omitted.

Lemma 2. *For any strings T and P, and any integer i with $1 \leq i \leq |T|$, let $u_1 = \max LnOcc(T[1 : i - 1], P) + |P| - 1$ and $u_2 = i - 1 + \min RnOcc(T[i : |T|], P)$. Then $nOcc(T, P) = |LnOcc(T[1 : u_1], P)| + nOcc(T[u_1 + 1 : u_2 - 1], P) + |RnOcc(T[u_2 : |T|], P)|$.*

Proof. By Lemma 1 and the definitions of u_1, u_2, $LnOcc$ and $RnOcc$, we have

$$\begin{aligned} &nOcc(T, P) \\ &= |LnOcc(T[1 : u_1], P)| + |LnOcc(T[u_1 + 1 : |T|], P)| \\ &= |LnOcc(T[1 : u_1], P)| + |RnOcc(T[u_1 + 1 : |T|], P)| \\ &= |LnOcc(T[1 : u_1], P)| + |RnOcc(T[u_1 + 1 : u_2 - 1], P)| + |RnOcc(T[u_2 : |T|], P)| \\ &= |LnOcc(T[1 : u_1], P)| + nOcc(T[u_1 + 1 : u_2 - 1], P) + |RnOcc(T[u_2 : |T|], P)|. \end{aligned}$$

□

We will later make use of the solution to the following problem, where occurrences of q-grams are weighted and allowed to overlap.

Problem 1 (weighted overlapping q-gram frequencies). Given a string T, an integer q, and an integer array w ($|w| = |T|$), compute $\sum_{i \in Occ(T,P)} w[i]$ for all q-grams $P \in \Sigma^q$ where $Occ(T, P) \neq \emptyset$.

Theorem 1 ([5]). *Problem 1 can be solved in $O(|T|)$ time.*

2.3 Straight Line Programs

In this paper, we treat strings described in terms of *straight line programs* (*SLPs*). A straight line program $\mathcal{T}$ is a sequence of assignments $\{X_1 = expr_1, X_2 = expr_2, \ldots, X_n = expr_n\}$. Each X_i is a variable and each $expr_i$ is an expression where $expr_i = a$ ($a \in \Sigma$), or $expr_i = X_\ell X_r$ ($\ell, r < i$). We will sometimes abuse notation and denote $\mathcal{T}$ as $\{X_i\}_{i=1}^n$. Denote by T the string derived from the last variable X_n of the program $\mathcal{T}$. Fig. 1 shows an example of an SLP. The *size* of the program $\mathcal{T}$ is the number n of assignments in $\mathcal{T}$.

Let $val(X_i)$ represent the string derived from X_i. When it is not confusing, we identify a variable X_i with $val(X_i)$. Then, $|X_i|$ denotes the length of the string X_i derives, and $X_i[j] = val(X_i)[j]$, $X_i[j : k] = val(X_i)[j : k]$ for $1 \leq j, k \leq |X_i|$. Let $vOcc(X_i)$ denote the number of times a variable X_i occurs in the derivation of T. For example, $vOcc(X_4) = 3$ in Fig. 1.

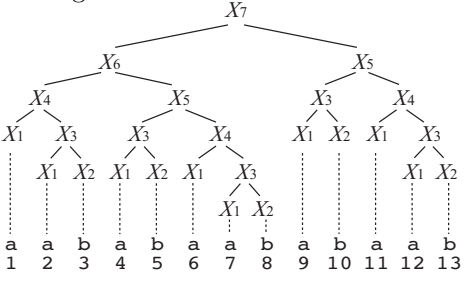


Fig. 1. The derivation tree of SLP $\mathcal{T} = \{X_1 = \mathtt{a}, X_2 = \mathtt{b}, X_3 = X_1X_2, X_4 = X_1X_3, X_5 = X_3X_4, X_6 = X_4X_5, X_7 = X_6X_5\}$, which represents string $T = val(X_7) = \mathtt{aababaababaab}$

Both $|X_i|$ and $vOcc(X_i)$ can be computed for all $1 \leq i \leq n$ in a total of $O(n)$ time by a simple iteration on the variables: $|X_i| = 1$ for any $X_i = a$ ($a \in \Sigma$), and $|X_i| = |X_\ell| + |X_r|$ for any $X_i = X_\ell X_r$. Also, $vOcc(X_n) = 1$ and for $i < n$, $vOcc(X_i) = \sum\{vOcc(X_k) \mid X_k = X_\ell X_i\} + \sum\{vOcc(X_k) \mid X_k = X_i X_r\}$.

We shall assume as in various previous work on SLP, that the word size is at least $\log |T|$, and hence, values representing lengths and positions of T in our algorithms can be manipulated in constant time.

3 *q*-gram Non-overlapping Frequencies on Compressed String

The goal of this paper is to efficiently solve the following problem.

Problem 2 (Non-overlapping q-gram frequencies on SLP). Given an SLP $\mathcal{T}$ of size n that describes string T and a positive integer q, compute $nOcc(T, P)$ for all q-grams $P \in \Sigma^q$.

If we decompress the given SLP $\mathcal{T}$ obtaining the string T, then we can solve the problem in $O(|T|)$ time. However, it holds that $|T| = \Theta(2^n)$. Hence, in order to solve the problem efficiently, we have to establish an algorithm that does not explicitly decompress the given SLP $\mathcal{T}$.

3.1 Key Ideas

For any variable X_i and integer $k \geq 1$, let $pre(X_i, k) = X_i[1 : \min\{k, |X_i|\}]$ and $suf(X_i, k) = X_i[|X_i| - \min\{k, |X_i|\} + 1 : |X_i|]$. That is, $pre(X_i, k)$ and $suf(X_i, k)$ are the prefix and the suffix of $val(X_i)$ of length k, respectively. For all variables X_i, $pre(X_i, k)$ can be computed in a total of $O(nk)$ time and space, as follows:

$$pre(X_i, k) = \begin{cases} val(X_i) & \text{if } |X_i| \leq k, \\ pre(X_\ell, k)pre(X_r, k - |X_\ell|) & \text{if } X_i = X_\ell X_r \text{ and } |X_\ell| < k < |X_i|, \\ pre(X_\ell, k) & \text{if } X_i = X_\ell X_r \text{ and } k \leq |X_\ell|. \end{cases}$$

$suf(X_i, k)$ can be computed similarly in $O(nk)$ time and space.

For any string T and positive integers q and j $(1 \leq j \leq j + q - 1 \leq |T|)$, the *longest overlapping cover* of the q-gram $P = T[j : j + q - 1]$ w.r.t. position j of T is an ordered pair $\overleftrightarrow{loc}_q(T, j) = (b, e)$ of positions in T which is defined as:

$$\overleftrightarrow{loc}_q(T, j) = \arg\max_{(b,e)} \left\{ (e - b) \;\middle|\; \begin{array}{l} (b, e) \in Occ(T, P) \times ((q - 1) \oplus Occ(T, P)), \\ b \leq j \leq j + q - 1 \leq e, \\ \forall k \in [b : e - q] \cap Occ(T, P), \\ \quad [k + 1 : \min\{k + q - 1, e - q + 1\}] \cap Occ(T, P) \neq \emptyset \end{array} \right\}$$

Namely, $\overleftrightarrow{loc}_q(T, j)$ represents the beginning and ending positions of the maximum chain of overlapping occurrences of q-gram $T[j : j + q - 1]$ that contains position j. For example, consider string $T = \texttt{aaabaabaaabaabaaaabaa}$ of length 21. For $q = 5$ and $j = 9$, we have $\overleftrightarrow{loc}_q(T, j) = (2, 16)$, since $T[2 : 6] = T[5 : 9] = T[9 : 13] = T[12 : 16] = \texttt{aabaa}$. Note that $T[17 : 21] = \texttt{aabaa}$ is not contained in this chain since it does not overlap with $T[12 : 16]$.

Lemma 3. *Given a string T and integers q, j, the longest overlapping cover $\overleftrightarrow{loc}_q(T, j)$ can be computed in $O(|T|)$ time.*

Proof. Using, for example, the KMP algorithm [9], we can obtain a sorted list of $Occ(T, T[j : j + q - 1])$ in $O(|T|)$ time. We can just scan this list forwards and backwards, to easily obtain b and e. □

For a variable $X_i = X_\ell X_r$ and a position $1 \leq j \leq |X_i| - q + 1$, a longest overlapping cover $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ is said to be *closed in* X_i if $q - 1 < b$ and $e < |X_i| - q + 2$.

Theorem 2. *Problem 2 can be solved in $O(q^2 n)$ time, provided that, for all variables $X_i = X_\ell X_r$ and j s.t. $|X_i| \geq q$ and $\max\{1, |X_\ell| - 2(q-1) + 1\} \leq j \leq \min\{|X_\ell| + q - 1, |X_i| - q + 1\}$, $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ and $nOcc(X_i[b : e], s)$ are already computed where $s = X_i[j : j + q - 1]$.*

Proof. Algorithm 1 shows a pseudo-code of our algorithm to solve Problem 2.

Consider q-gram $s = X_i[j : j+q-1]$ at position j for which $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ is closed in X_i. A key observation is that, if (b, e) is closed in X_i, then (b, e) is never closed in X_ℓ or X_r. Therefore, by summing up $vOcc(X_i) \cdot nOcc(X_i[b : e], s)$ for each closed (b, e) in X_i, for all such variables X_i, we obtain $nOcc(T, s)$. Line 1 is sufficient to check if (b, e) is closed.

For all $1 \leq i \leq n$, $vOcc(X_i)$ can be computed in $O(n)$ time, and $t_i = pre(X_i, 2(q-1))suf(X_i, 2(q-1))$ can be computed in $O(qn)$ time and space. The problem amounts to summing up the values of $vOcc(X_i) \cdot nOcc(X_i[b : e], s)$ for each q-gram s contained in each t_i, and can be reduced to Problem 1 on string z and integer array w of length $O(qn)$, which can be solved in $O(qn)$ time by Theorem 1.

In line 1, we check if there is no previous position h ($\max\{1, |X_\ell| - 2(q-1) + 1\} \leq h < j$) such that $X_i[h : h + q - 1] = X_i[j : j + q - 1]$ by $\overleftrightarrow{loc}_q(X_i, h) = \overleftrightarrow{loc}_q(X_i, j)$, so that we do not count the same q-gram more than once. If there is no such h, we set the value of $w_i[k - |X_\ell| + j]$ to $vOcc(X_i) \cdot nOcc(X_i[b : e], s)$. This can be checked in $O(q^2 n)$ time for all X_i and j.

For convenience, we assume that $T = val(X_n)$ starts and ends with special characters $\#^{q-1}$ and $\$^{q-1}$ that do not occur anywhere else in T, respectively. Then we can cope with the last variable X_n as described above. Hence the theorem holds. □

3.2 Computing Longest Overlapping Covers

In this subsection, we will show how to compute longest overlapping cover $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ where $s = X_i[j : j+q-1]$ for all X_i and all j required for Theorem 2.

For any string T and integers q and j $(1 \leq j < q)$, let

$$\overrightarrow{loc}_q(T, j) = \begin{cases} (j, be) & \text{if } j + q - 1 \leq |T|, \\ (j, |T|) & \text{otherwise,} \end{cases}$$

$$\overleftarrow{loc}_q(T, j) = \begin{cases} (eb, |T| - j + 1) & \text{if } |T| - j - q + 2 \geq 1, \\ (1, |T| - j + 1) & \text{otherwise,} \end{cases}$$

where $(j, be) = (j - 1) \oplus \overleftrightarrow{loc}_q(T[j : |T|], 1)$ and $(eb, |T| - j + 1) = \overleftrightarrow{loc}_q(T[1 : |T|-j+1], |T|-j-q+2)$. Namely, $\overrightarrow{loc}_q(T, j)$ is a suffix of the longest overlapping cover of the q-gram $T[j : j + q - 1]$ that begins at position j $(1 \leq j < q)$ in T, and $\overleftarrow{loc}_q(T, j)$ is a prefix of the longest overlapping cover of the q-gram $T[|T| - j - q + 2 : |T| - j + 1]$ that ends at position $|T| - j + 1$ in T.

Algorithm 1. Computing q-gram non-overlapping frequencies from SLP

Input: SLP $\mathcal{T} = \{X_i\}_{i=1}^{n}$ representing string T, integer $q \geq 2$.
Output: $nOcc(T, P)$ for all q-grams $P \in \Sigma^q$ where $Occ(T, P) \neq \emptyset$.

Compute $vOcc(X_i)$ for all $1 \leq i \leq n$;
Compute $pre(X_i, 2(q-1))$ and $suf(X_i, 2(q-1))$ for all $1 \leq i \leq n-1$;
$z \leftarrow \varepsilon$; $w \leftarrow []$;
for $i \leftarrow 1$ **to** n **do**
  if $|X_i| \geq q$ **then**
    let $X_i = X_\ell X_r$;
    $k \leftarrow |suf(X_\ell, 2(q-1))|$;
    $t_i = suf(X_\ell, 2(q-1))pre(X_r, 2(q-1))$;
    z.append(t_i);
    $w_i \leftarrow$ create integer array of length $|t_i|$, each element set to 0;
    for $j \leftarrow \max\{1, |X_\ell| - 2(q-1) + 1\}$ **to** $\min\{|X_\ell| + q - 1, |X_i| - q + 1\}$ **do**
      $s \leftarrow X_i[j : j + q - 1]$;
      $(b, e) \leftarrow \overleftrightarrow{loc}_q(X_i, j)$;
      if $q - 1 < b$ **and** $e < |X_i| - q + 2$ **then**
        if $\overleftrightarrow{loc}_q(X_i, h) \neq \overleftrightarrow{loc}_q(X_i, j)$ *for any position* h *s.t.* $\max\{1, |X_\ell| - 2(q-1) + 1\} \leq h < j$ **then**
          $w_i[k - |X_\ell| + j] \leftarrow vOcc(X_i) \cdot nOcc(X_i[b : e], s)$;
  w.append(w_i);
Calculate q-gram frequencies in z, where each q-gram starting at position d is *weighted* by $w[d]$.

Lemma 4. *For all* $1 \leq i \leq n$ *and* $1 \leq j \leq 2(q-1)$, $\overrightarrow{loc}_q(X_i, j)$ *can be computed in a total of* $O(q^2 n)$ *time.*

Proof. We use dynamic programming. Let $X_i = X_\ell X_r$, $p_j = X_i[j : j+q-1]$, and assume $\overrightarrow{loc}_q(X_\ell, j)$ and $\overrightarrow{loc}_q(X_r, j)$ have been calculated for all $1 \leq j \leq 2(q-1)$. We examine the string $X_i[\max\{j, |X_\ell| - q + 2\} : \min\{|X_i|, |X_\ell| + q - 1\}]$ for occurrences of p_j that cross X_ℓ and X_r, obtain its longest overlapping cover (b_i, e_i), and check if it overlaps with $\overrightarrow{loc}_q(X_\ell, j)$. Furthermore, let bb_r be the left most occurrence of p_j in X_r that has the possibility of overlapping with (b_i, e_i). Then, $\overrightarrow{loc}_q(X_i, j)$ is either $\overrightarrow{loc}_q(X_\ell, j)$, or its end can be extended to e_i, or further to the end of $\overrightarrow{loc}_q(X_r, bb_r)$, depending on how the covers overlap.

More precisely, let $(j, be_\ell) = \overrightarrow{loc}_q(X_\ell, j)$, $(b_i, e_i) = \max\{j - 1, |X_\ell| - q + 1\} \oplus \overleftrightarrow{loc}_q(X_i[\max\{j, |X_\ell| - q + 2\} : \min\{|X_i|, |X_\ell| + q - 1\}], h)$ where $h \in Occ(X_i[\max\{j, |X_\ell| - q + 2\} : \min\{|X_i|, |X_\ell| + q - 1\}], p_j)$, and $(bb_r, be_r) = (|X_\ell| + k - 1) \oplus \overrightarrow{loc}_q(X_r, k)$ where $k = \min Occ(pre(X_r, 2(q-1)), p_j)$. (Note that $(bb_r, be_r), (b_i, e_i)$ are not defined if occurrences h, k of p_j do not exist.) Then we have

$$\overrightarrow{loc}_q(X_i, j) = \begin{cases} (j, be_\ell) & \text{if } be_\ell < b_i \text{ or } \nexists h, \\ (j, e_i) & \text{if } b_i \le be_\ell \text{ and } (e_i < bb_r \text{ or } \nexists k) \\ (j, be_r) & \text{otherwise.} \end{cases}$$

For all variables X_i we pre-compute $pre(X_i, 2(q-1))$ and $suf(X_i, 2(q-1))$. This can be done in a total of $O(qn)$ time. Then, each $\overrightarrow{loc}_q(X_i, j)$ can be computed in $O(q)$ time using the KMP algorithm, Lemma 3, and the above recursion, giving a total of $O(q^2 n)$ time for all $1 \le i \le n$ and $1 \le j \le 2(q-1)$. □

Lemma 5. *For all* $1 \le i \le n$ *and* $1 \le j \le 2(q-1)$, $\overleftarrow{loc}_q(X_i, j)$ *can be computed in a total of* $O(q^2 n)$ *time.*

Proof. The proof is essentially the same as the proof for $\overrightarrow{loc}_q(X_i, j)$ in Lemma 4.

Recall that we have assumed in Theorem 2 that $\overleftrightarrow{loc}_q(X_i, j)$ are already computed. The following lemma describes how $\overleftrightarrow{loc}_q(X_i, j)$ can actually be computed in a total of $O(q^2 n)$ time.

Lemma 6. *For all variable* $X_i = X_\ell X_r$ *and* j *s.t.* $\max\{1, |X_\ell| - 2(q-1) + 1\} \le j \le \min\{|X_\ell| + q - 1, |X_i| - q + 1\}$, $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ *can be computed in a total of* $O(q^2 n)$ *time.*

Proof. Let $s_j = X_i[j : j + q - 1]$. At first, we compute $(b_i, e_i) = \overleftrightarrow{loc}_q(X_i[|X_\ell| - 2(q-1) + 1 : \min\{|X_i|, |X_\ell| + 2(q-1)\}], j)$ and then $\overleftrightarrow{loc}_q(X_i, j)$ can be computed based on (b_i, e_i), as follows: Let $(eb_\ell, ee_\ell) = \overleftarrow{loc}_q(X_\ell, |X_\ell| - ee_\ell + 1)$ and $(bb_r, be_r) = |X_\ell| \oplus \overrightarrow{loc}_q(X_r, bb_r - |X_\ell|)$, where $ee_\ell = \max Occ(X_i[\max\{1, |X_\ell| - 2(q-1) + 1\} : |X_\ell|], s_j)$ and $bb_r = \min Occ(X_i[|X_\ell| + 1 : \min\{|X_i|, |X_\ell| + 2(q-1)\}], s_j)$.

1. If $b_i \le |X_\ell|$ and $e_i > |X_\ell|$, then we have $b \le b_i \le |X_\ell| < e_i \le e$. $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ can be computed by checking whether (eb_ℓ, ee_ℓ), (b_i, e_i), and (bb_r, be_r) are overlapping or not.
2. If $e_i \le |X_\ell|$, then trivially $b = eb_\ell$ and $e = e_i$.
3. If $b_i > |X_\ell|$, then trivially $b = b_i$ and $e = be_r$.

Each $ee_\ell = h$ and $bb_r = |X_\ell| + k$ can be computed using the KMP algorithm on string $suf(X_\ell, 2(q-1))pre(X_r, 2(q-1))$ in $O(q)$ time. By Lemmas 4 and 5, (eb_ℓ, ee_ℓ) and (bb_r, be_r) can be pre-computed in a total of $O(q^2 n)$ time for all $1 \le i \le n$. Hence the lemma holds. □

3.3 Largest Left-Priority and Smallest Right-Priority Occurrences

In order to compute $nOcc(X_i[b : e], s)$ for all X_i and all j required for Theorem 2, where $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ and $s = X_i[j : j + q - 1]$, we will use the largest and second largest occurrences of $LnOcc$ and the smallest and second smallest occurrences of $RnOcc$.

For any set S of integers and integer $1 \leq k \leq |S|$, let $\max_k S$ and $\min_k S$ denote the k-th largest and the k-th smallest element of S.

For $1 \leq i \leq n$ and $1 \leq j \leq 2(q-1)$, consider to compute $\max_k LnOcc(X_i[j : be_i], p_j)$ for $k = 1, 2$, where $(j, be_i) = \overrightarrow{loc}_q(X_i, j)$ and $p_j = X_i[j : j+q-1]$. Intuitively, difficulties in computing $\max_k LnOcc(X_i[j : be_i], p_j)$ come from the fact that $X_i[j : be_i]$ can be as long as $\Theta(2^n)$, but we only have prefix $pre(X_i, 3(q-1))$ and suffix $suf(X_i, 3(q-1))$ of $val(X_i)$ of length $O(q)$. Hence we cannot compute the value of be_i by simply running the KMP algorithm on those partial strings. For the same reason, the size of $LnOcc(X_i[j : be_i], p_j)$ can be as large as $\Theta(2^n/q)$. Hence we cannot store $LnOcc(X_i[j : be_i], p_j)$ as is. Still, as will be seen in the following lemma, we can compute those values efficiently, only in $O(q^2 n)$ time.

Lemma 7. *For all variable $X_i = X_\ell X_r$ and $1 \leq j \leq 2(q-1)$, let $(j, be_i) = \overrightarrow{loc}_q(X_i, j)$, $p_j = X_i[j : j+q-1]$. We can compute the values $\max_1 LnOcc(X_i[j : be_i], p_j)$ and $\max_2 LnOcc(X_i[j : be_i], p_j)$ for all $1 \leq i \leq n$ and $1 \leq j \leq 2(q-1)$, in a total of $O(q^2 n)$ time.*

Proof. Omitted.

The next lemma can be shown similarly to Lemma 7.

Lemma 8. *For all variable $X_i = X_\ell X_r$ and $1 \leq j \leq 2(q-1)$, let $(eb, ee) = \overleftarrow{loc}_q(X_i, j)$, and $s_j = X_i[|X_i|-j-q+2 : |X_i|-j+1]$. We can compute the values $\min_1 RnOcc(X_i[eb : ee], s_j)$ and $\min_2 RnOcc(X_i[eb : ee], s_j)$ for all $1 \leq i \leq n$ and $1 \leq j \leq 2(q-1)$, in a total of $O(q^2 n)$ time.*

Lemma 9. *For all variable $X_i = X_\ell X_r$ and $1 \leq j < q$, $\max LnOcc(X_i[eb_i : ee_i], s_j)$ can be computed in a total of $O(q^2 n)$ time, where $(eb_i, ee_i) = \overleftarrow{loc}_q(X_i, j)$ and $s_j = X_i[|X_i| - j - q + 2 : |X_i| - j + 1]$.*

Proof. The lemma can be shown by using Lemma 7, but the proof is omitted.

Lemma 10. *For all variable $X_i = X_\ell X_r$ and $1 \leq j < q$, $\min RnOcc(X_i[bb_i : be_i], p_j)$ can be computed in a total of $O(q^2 n)$ time, where $(bb_i, be_i) = \overrightarrow{loc}_q(X_i, j)$ and $p_j = X_i[j : j+q-1]$.*

Proof. The lemma can be shown in a similar way to Lemma 9, using Lemma 8 instead of Lemma 7. □

3.4 Counting Non-overlapping Occurrences in Longest Overlapping Covers

First, we show how to count non-overlapping occurrences of q-gram p_j in $X_i[j : be_i]$, for all i and j, where $p_j = X_i[j : j+q-1]$ and $(j, be_i) = \overrightarrow{loc}_q(X_i, j)$.

Lemma 11. *For all variable $X_i = X_\ell X_r$ and $1 \leq j \leq 2(q-1)$, let $(j, be_i) = \overrightarrow{loc}_q(X_i, j)$ and $p_j = X_i[j : j+q-1]$. We can compute $nOcc(X_i[j : be_i], p_j)$ for all $1 \leq i \leq n$ and $1 \leq j \leq 2(q-1)$, in a total of $O(q^2 n)$ time.*

Proof. By Lemma 1, we have $nOcc(X_i[j : be_i], p_j) = |LnOcc(X_i[j : be_i], p_j)|$. We compute the occurrence b_i in $(j-1) \oplus LnOcc(X_i[j : be_i], p_j)$ that crosses X_ℓ and X_r, if such exists. Note that at most one such occurrence exists. Also, we compute the smallest occurrence bb_r in $(j-1) \oplus LnOcc(X_i[j : be_i], p_j)$ that is completely within X_r. Then the desired value $nOcc(X_i[j : be_i], p_j)$ can be computed depending whether b_i and bb_r exist or not.

Formally: Consider the set $S = ((j-1) \oplus LnOcc(X_i[j : be_i], p_j)) \cap [|X_\ell| - q + 2 : |X_\ell|]$ of occurrence of p_j which is either empty or singleton. If S is singleton, then let b_i be its single element. Let $bb_r = \min\{k \mid k \in ((j-1) \oplus LnOcc(X_i[j : be_i], p_j)) \cap [|X_\ell| + 1 : |X_\ell| + q - 1], \text{if } \exists b_i \text{ then } k \geq b_i + q\}$.

Then we have

$$nOcc(X_i[j : be_i], p_j) = \begin{cases} nOcc(X_r[j - |X_\ell| : be_i - |X_\ell|], p_j) & \text{if } j > |X_\ell|, \\ nOcc(X_\ell[j : be_\ell], p_j) & \text{if } \nexists b_i \text{ and } \nexists bb_r, \\ nOcc(X_\ell[j : be_\ell], p_j) + 1 & \text{if } \exists b_i \text{ and } \nexists bb_r \\ nOcc(X_\ell[j : be_\ell], p_j) + nOcc(X_r[b_r : be_r], p_j) & \text{if } \nexists b_i \text{ and } \exists bb_r, \\ nOcc(X_\ell[j : be_\ell], p_j) + nOcc(X_r[b_r : be_r], p_j) + 1 & \text{if } \exists b_i \text{ and } \exists bb_r, \end{cases}$$

where $(bb_r, be_r) = \overrightarrow{loc}_q(X_r, bb_r)$.

For all variables X_i we pre-compute $pre(X_i, 3(q-1))$ and $suf(X_i, 3(q-1))$. This can be done in a total of $O(qn)$ time. If b_i or bb_r exists, $|X_\ell| - 3(q-1) < j - 1 + \max LnOcc(X_\ell[j : be_\ell], p_j) \leq |X_\ell| - q + 2$. Then, each b_i and bb_r can be computed from $LnOcc(X_i[(j-1+\max LnOcc(X_\ell[j : be_\ell], p_j)) : |X_\ell|+3(q-1)], p_j)$ running the KMP algorithm on string $suf(X_\ell, 3(q-1))pre(X_r, 3(q-1))$. Based on the above recursion, we can compute $nOcc(X_i[j : be_i], p_j)$ in a total of $O(q^2 n)$ time for all $1 \leq i \leq n$ and $1 \leq j \leq 2(q-1)$. □

The next lemma can be shown similarly to Lemma 11.

Lemma 12. *For all variable $X_i = X_\ell X_r$ and $1 \leq j \leq 2(q-1)$, let $(eb_i, ee_i) = \overleftarrow{loc}_q(X_i, j)$ and $s_j = X_i[|X_i| - j - q + 2 : |X_i| - j + 1]$. We can compute $nOcc(X_i[eb_i : ee_i], s_j)$ for all $1 \leq i \leq n$ and $1 \leq j \leq 2(q-1)$, in a total of $O(q^2 n)$ time.*

We have also assumed in Theorem 2 that $nOcc(X_i[b : e], s_j)$ are already computed. This can be computed efficiently, as follows:

Lemma 13. *For all variable $X_i = X_\ell X_r$ and j s.t. $\min\{1, |X_\ell| - 2(q-1) + 1\} \leq j \leq \min\{|X_i| - q + 1, |X_\ell| + q - 1\}$, $nOcc(X_i[b : e], s_j)$ can be computed in a total of $O(q^2 n)$ time, where $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ and $s_j = X_i[j : j + q - 1]$.*

Proof. We consider the case where $\max\{1, |X_\ell| - q + 2\} \leq j \leq |X_\ell|$, as the other cases can be shown similarly. Our basic strategy for computing $nOcc(X_i[b : e], s_j)$ is as follows. First, we compute the largest element of $LnOcc(X_i[b : e], s_j)$ that occurs completely within X_ℓ. Second, we compute the smallest element of

$RnOcc(X_i[b:e], s_j)$ that occurs completely within X_r. Third, we compute an occurrence of s_j that crosses the boundary of X_ℓ and X_r, and do not overlap the above occurrences of s_j completely within X_ℓ and X_r.

Formally: Let $ee_\ell = b + q - 2 + \max Occ(X_i[b : |X_\ell|], s_j)$, $bb_r = |X_\ell| + \min Occ(X_i[|X_\ell| + 1 : e], s_j)$, $u_1 = b + q - 2 + \max LnOcc(X_i[b : ee_\ell], s_j)$, and $u_2 = bb_r - 1 + \min RnOcc(X_i[bb_r : e], s_j)$. We consider the case where all these values exist, as other cases can be shown similarly. It follows from Lemmas 1 and 2 that

$$\begin{aligned} &nOcc(X_i[b:e], s_j) \\ &= |LnOcc(X_i[b:u_1], s_j)| + nOcc(X_i[u_1+1:u_2-1], s_j) + |RnOcc(X_i[u_2:e], s_j)| \\ &= nOcc(X_i[b:ee_\ell], s_j) + nOcc(X_i[u_1+1:u_2-1], s_j) + nOcc(X_i[bb_r:e], s_j), \end{aligned}$$

By Lemma 6, $(b, e) = \overleftrightarrow{loc}_q(X_i, j)$ can be pre-computed in a total of $O(q^2 n)$ time. Since $b < ee_\ell$ and $bb_r < e$, ee_ℓ and bb_r can be computed in $O(q)$ time using the KMP algorithm. By Lemmas 11 and 12, $nOcc(X_i[b : ee_\ell], s_j)$ and $nOcc(X_i[bb_r : e], s_j)$ can be pre-computed in a total of $O(q^2 n)$ time (Notice $(b, ee_\ell) = \overleftarrow{loc}_q(X_\ell, ee_\ell)$ and $(bb_r, e) = |X_\ell| \oplus \overrightarrow{loc}_q(X_r, bb_r - |X_\ell|)$). By Lemmas 9 and 10, u_1 and u_2 can be pre-computed in a total of $O(q^2 n)$ time. Hence $nOcc(X_i[u_1 + 1 : u_2 - 1], s_j)$ can be computed in $O(q)$ time using the KMP algorithm for each i and j. The lemma thus holds. □

3.5 Main Result

The following theorem concludes this whole section.

Theorem 3. *Problem 2 can be solved in $O(q^2 n)$ time and $O(qn)$ space.*

Proof. The time complexity and correctness follow from Theorem 2, Lemma 6, and Lemma 13.

We compute and store strings $suf(X_i, 3(q-1))$ and $pre(X_i, 3(q-1))$ of length $O(q)$ for each variable X_i, hence this requires a total of $O(qn)$ space for all $1 \leq i \leq n$. We use a constant number of dynamic programming tables each of which is of size $O(qn)$. Hence the total space complexity is $O(qn)$. □

4 Conclusion

We considered the problem of computing the non-overlapping frequencies for all q-grams that occur in a given text represented as an SLP. Our algorithm greatly improves previous work which solved the problem only for $q = 2$ requiring $O(n^4 \log n)$ time and $O(n^3)$ space. We give the first algorithm which works for any $q \geq 2$, running in $O(q^2 n)$ time and $O(qn)$ space, where n is the size of the SLP.

References

1. Amir, A., Benson, G.: Efficient two-dimensional compressed matching. In: Proc. DCC 1992, pp. 279–288 (1992)
2. Apostolico, A., Preparata, F.P.: Data structures and algorithms for the string statistics problem. Algorithmica 15(5), 481–494 (1996)
3. Bille, P., Landau, G.M., Raman, R., Sadakane, K., Satti, S.R., Weimann, O.: Random access to grammar-compressed strings. In: Proc. SODA 2011, pp. 373–389 (2011)
4. Brodal, G.S., Lyngsø, R.B., Östlin, A., Pedersen, C.N.S.: Solving the String Statistics Problem in Time $O(n \log n)$. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 728–739. Springer, Heidelberg (2002)
5. Goto, K., Bannai, H., Inenaga, S., Takeda, M.: Towards efficient mining and classification on compressed strings. In: Accepted for SPIRE 2011 (2011), preprint available at arXiv:1103.3114v2
6. Hermelin, D., Landau, G.M., Landau, S., Weimann, O.: A unified algorithm for accelerating edit-distance computation via text-compression. In: Proc. STACS 2009, pp. 529–540 (2009)
7. Inenaga, S., Bannai, H.: Finding characteristic substring from compressed texts. In: Proc. The Prague Stringology Conference 2009, pp. 40–54 (2009); full version to appear in the International Journal of Foundations of Computer Science
8. Karpinski, M., Rytter, W., Shinohara, A.: An efficient pattern-matching algorithm for strings with short descriptions. Nordic Journal of Computing 4, 172–186 (1997)
9. Knuth, D.E., Morris, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM Journal on Computing 6(2), 323–350 (1977)
10. Larsson, N.J., Moffat, A.: Off-line dictionary-based compression. Proceedings of the IEEE 88(11), 1722–1732 (2000)
11. Lifshits, Y.: Processing Compressed Texts: A Tractability Border. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 228–240. Springer, Heidelberg (2007)
12. Matsubara, W., Inenaga, S., Ishino, A., Shinohara, A., Nakamura, T., Hashimoto, K.: Efficient algorithms to compute compressed longest common substrings and compressed palindromes. Theoretical Computer Science 410(8-10), 900–913 (2009)
13. Navarro, G., Mäkinen, V.: Compressed full-text indexes. ACM Computing Surveys 39(1), 2 (2007)
14. Nevill-Manning, C.G., Witten, I.H., Maulsby, D.L.: Compression by induction of hierarchical grammars. In: Proc. DCC 1994, pp. 244–253 (1994)
15. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory IT-23(3), 337–349 (1977)
16. Ziv, J., Lempel, A.: Compression of individual sequences via variable-length coding. IEEE Transactions on Information Theory 24(5), 530–536 (1978)

A Fast Approximation Scheme for the Multiple Knapsack Problem

Klaus Jansen*

Institut für Informatik, Christian-Albrechts-Universität zu Kiel,
24098 Kiel, Germany
kj@informatik.uni-kiel.de

Abstract. In this paper we propose an improved efficient approximation scheme for the multiple knapsack problem (MKP). Given a set $\mathcal{A}$ of n items and set $\mathcal{B}$ of m bins with possibly different capacities, the goal is to find a subset $S \subseteq \mathcal{A}$ of maximum total profit that can be packed into $\mathcal{B}$ without exceeding the capacities of the bins. Chekuri and Khanna presented a PTAS for MKP with arbitrary capacities with running time $n^{O(1/\epsilon^8 \log(1/\epsilon))}$. Recently we found an efficient polynomial time approximation scheme (EPTAS) for MKP with running time $2^{O(1/\epsilon^5 \log(1/\epsilon))} poly(n)$. Here we present an improved EPTAS with running time $2^{O(1/\epsilon \log^4(1/\epsilon))} + poly(n)$. If the integrality gap between the ILP and LP objective values for bin packing with different sizes is bounded by a constant, the running time can be further improved to $2^{O(1/\epsilon \log^2(1/\epsilon))} + poly(n)$.

1 Introduction

The knapsack problem is a fundamental problem in combinatorial optimization. One interesting generalization is the multiple knapsack problem (MKP), in which we are given a set $\mathcal{A}$ of n items and a set $\mathcal{B}$ of m bins or knapsacks. Each item $a \in \mathcal{A}$ has a size $size(a) \in \mathbb{Q}^+$ and a profit $profit(a) \in \mathbb{Q}^+$ and each bin $b \in \mathcal{B}$ has a capacity or size $c(b) \in \mathbb{Q}^+$. The goal of MKP is to find a subset $S \subseteq \mathcal{A}$ that can be packed into $\mathcal{B}$ without exceeding the capacities of the bins and that has maximum total profit $profit(S) = \sum_{a \in S} profit(a)$. The maximum total profit among all feasible subsets $S \subseteq \mathcal{A}$ that can be packed into $\mathcal{B}$ is denoted by $OPT(\mathcal{A}, \mathcal{B})$. MKP has many applications in computer science, operations research, and related disciplines; see also the book by Kellerer, Pferschy, and Pisinger [6].

Results. The decision version of MKP is to determine whether there is a feasible packing with profit at least p; this is a generalization of the classical bin packing problem and, therefore, strongly NP-complete. In contrast to the classical knapsack problem, MKP even with two bins with the same capacity does

* Supported in part by DFG Project, Entwicklung von effizienten polynomiellen Approximationsschemata für Scheduling- und verwandte Optimierungsprobleme.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 313–324, 2012.

not have a fully polynomial time approximation scheme (FPTAS) unless P=NP [1,2]. Chekuri and Khanna [2] presented a PTAS for MKP. The running time of their PTAS is $n^{O(1/\epsilon^8 \log(1/\epsilon))}$. Chekuri and Khanna [2] posed the question of whether there is a PTAS with an improved running time and conjectured that there exists an efficient polynomial time approximation scheme (EPTAS) with running time $f(1/\epsilon)poly(n)$ for some function f. Fellows [3] considered it as a significant open problem to determine whether MKP admits a fixed parameter tractable (FPT) algorithm or it is W[1]-hard. For a survey on approximation algorithms and parameterized complexity we refer to [8]. Recently we [4] found an EPTAS for MKP with running time $2^{O(1/\epsilon^5 \log(1/\epsilon))}poly(n)$ (that can be bounded also by $2^{O(1/\epsilon^5 \log(1/\epsilon))} + poly(n)$) answering the open question posed by Chekuri and Khanna in the affirmative. In this paper we improve the running time above and obtain the following main result:

Theorem 1. *There is an efficient polynomial time approximation scheme (EPTAS) for the multiple knapsack problem with running time* $2^{O(1/\epsilon \log^4(1/\epsilon))} + poly(n)$.

Interestingly, if the integrality gap between the ILP and LP objective values for the bin packing problem with different bin sizes is bounded by a constant C, similar to the modified round-up conjecture by Scheithauer and Terno [9] (i.e. that $ILP(I) \leq \lceil LP(I) \rceil + 1$ for the ILP and LP formulations for each instance I of the classical bin packing problem), then we can reduce the above running time to $2^{O(1/\epsilon \log^2(1/\epsilon))} + poly(n)$ (see also our full paper).

Techniques. In contrast to the previous approach by Chekuri and Khanna [2], we use a linear program relaxation for MKP. This allows us to select fractional pieces of items and to distribute them among different bin groups. We used this idea to obtain the first EPTAS for MKP, but still with a large running time $2^{O(1/\epsilon^5 \log(1/\epsilon))} + poly(n)$. In contrast to our first approach, we do not round the bin sizes in our new algorithm. To reduce the running time above we propose several interesting techniques and ideas. The first technique is a pre-assignment method for high and medium profit items into the block with the largest $\lceil 1/\delta \log^2(1/\delta) \rceil$ bins for some $\delta > 0$ (that depends on ϵ). Interestingly we are able to round up the sizes of certain items for this block. We show that the rounded items can be packed into the bin block plus one additional bin of small size. This step helps us to reduce the running time in the pre-assignment phase. The second technique is a rounding method for the solution of the LP relaxation of MKP. The solution of the LP can be interpreted as rectangles to be packed fractionally into blocks with $\lceil 1/\delta \log^2(1/\delta) \rceil$ bins. The height of a rectangle is equal to the size $size(a)$ of an item $a \in \mathcal{A}$ and the width is the total sum of all fractions of a assigned via configuration variables to a bin block. By rounding the rectangles, adding dummy rectangles and using a minimum cost flow problem with integral capacities, we are able to determine a selection of items that can be packed into the original and few additional bins. To pack a subset of the selected items into the bins with profit close to the optimum value,

we also generalize a classical result for bin packing by Karmarkar and Karp [5] (see also Shmonin [10]) to bin packing with different bin sizes: we prove that the integrality gap between the corresponding ILP and LP values for instances with d different item sizes and different bin sizes is bounded by $O(\log^2(d))$ (see our full paper for the details). We believe that the proposed new techniques and ideas are also useful for other combinatorial optimization problems (e.g. for scheduling with additional resources and 2D packing problems).

2 Main Algorithm for MKP

Let $\delta > 0$ be a constant such that $1/\delta$ is integral (δ will be specified later). Suppose that the number m of bins is a multiple of $M = \lceil 1/\delta \log^2(1/\delta) \rceil$ and $m \geq M$ (i.e. $m = (t+1)\lceil 1/\delta \log^2(1/\delta) \rceil$ with $t \geq 0$); otherwise simply use additional bins of zero capacity. Let us order the bins corresponding to their capacities $c(b_1) \leq \ldots \leq c(b_m)$. We build $t+1 = \frac{m}{M}$ blocks B_ℓ with M bins (see Figure 1 for an illustration with one zero capacity bin). We denote with $c_1^{(\ell)}, \ldots, c_M^{(\ell)}$ the capacities of the bins in B_ℓ for $\ell = 1, \ldots, t+1$. In the following we give an outline of our algorithm:

(1) Split the set $\mathcal{B}$ of bins into $t+1$ blocks B_ℓ with $M = \lceil 1/\delta \log^2(1/\delta) \rceil$ bins and reduce the number of high and medium profit items in the instance.
(2) Guess high profit and medium profit items (corresponding to an optimum solution) for block B_{t+1} with the largest bins and try to pack the items into block B_{t+1} plus one additional bin of small size.
(3) For each feasible guess of high and medium profit items
(3.1) solve a modified LP relaxation of MKP to select the other items for $\mathcal{B}$ (including small profit items for block B_{t+1}),
(3.2) construct 2D strip packing instances for each block B_ℓ of bins, add some dummy rectangles, round the rectangles and select items via a network flow computation,
(3.3) use a bin packing algorithm and a shifting argument to pack a subset of the selected items into the first t blocks $B_1, \ldots, B_t$.
(3.4) distribute the selected items of small profit fractionally into B_{t+1} and remove the fractional items,
(3.5) use a second shifting argument to eliminate the additional bin and at most $\lceil 1/\delta \log^2(1/\delta) \rceil$ further bins for B_{t+1}.
(4) Output a solution with largest profit among all feasible guesses.

3 Pre-assignment and LP Relaxation

Let $APP(\mathcal{A}, \mathcal{B})$ be an approximate value for MKP obtained by the greedy algorithm by Chekuri and Khanna [2] with accuracy $\epsilon'/2$ where $\epsilon' \leq 1$ is specified later; the algorithm computes a solution with profit at least $APP(\mathcal{A}, \mathcal{B}) \geq (1/2 - \epsilon'/4) OPT(\mathcal{A}, \mathcal{B})$. Note that $2(1+\epsilon')APP(\mathcal{A}, \mathcal{B}) \geq OPT(\mathcal{A}, \mathcal{B})$.

3.1 High Profit Items

In the first phase we place items with high profit at least $2\rho(1+\epsilon')APP(\mathcal{A},\mathcal{B}) \geq \rho OPT(\mathcal{A},\mathcal{B})$ into bin block B_{t+1} (see Figure 1 for an illustration), where $\rho = \Theta(\delta)$ is also a constant specified later. Note that an optimum solution can have at most $1/\rho = O(1/\delta)$ items with high profit. Using the following Lemma we can reduce the number of high profit items in our instance.

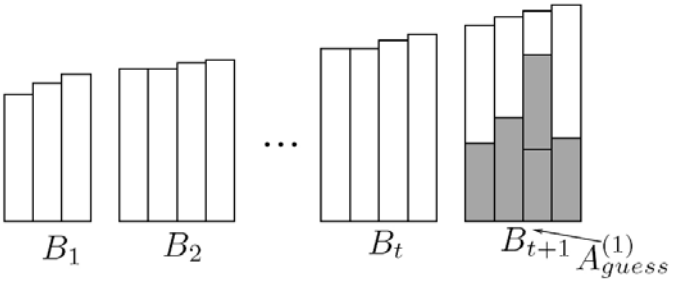


Fig. 1. Block structure for a general instance of MKP with guessed set $A^{(1)}_{guess}$ for B_{t+1}

Lemma 1. [4] *There is a set CA_h of high profit items in $\mathcal{A}$ with $|CA_h| \leq O(1/\delta^2 \log(1/\delta^2))$ such that an optimum solution which selects only high profit items from CA_h has profit at least $(1-3\epsilon')\ OPT(\mathcal{A},\mathcal{B})$ for $\epsilon' \leq 1/2$.*

Since there are at most $1/\rho$ items with high profit in any optimum solution, we can guess the high profit items out of the candidate set CA_h. The number of choices is at most $(|CA_h|+1)^{1/\rho} = 2^{O(1/\delta \log(1/\delta))}$ using $|CA_h|+1 \leq poly(1/\delta)$. For each feasible choice, we try to pack the chosen candidates into the $M = \lceil 1/\delta \log^2(1/\delta) \rceil$ bins in B_{t+1}. This can be done via an assignment from candidates to bins. The number of these assignments is at most $M^{1/\rho} = 2^{O(1/\delta \log(1/\delta))}$. An assignment is feasible if the assigned candidates fit into the corresponding bins. The total number of guesses in this phase is bounded by $2^{O(1/\delta \log(1/\delta))}$. Let $A^{(1)}_{guess}$ be the chosen candidates and let $Area_{Rem} = \sum_{b_i \in B_{t+1}} c(b_i) - size(A^{(1)}_{guess})$ be the remaining space. Furthermore, $\bar{c}(b_i)$ denotes the remaining capacity of bin $b_i \in B_{t+1}$ after the placement of the high profit items. Suppose for simplicity that the largest capacity $\bar{c}(b_i)$ among the bins is 1; i.e. $\max_{b_i \in B_{t+1}} \bar{c}(b_i) = 1$ (otherwise we scale the sizes of the items). This implies that $Area_{Rem} \leq \lceil 1/\delta \log^2(1/\delta) \rceil$.

3.2 Medium Profit Items

In the second phase we consider items with medium profit $profit(a_i) \in [2(\rho/M)(1+\epsilon')APP(\mathcal{A},\mathcal{B}), 2\rho\ (1+\epsilon')APP(\mathcal{A},\mathcal{B}))$. Since $2(\rho/M)(1+\epsilon)APP(\mathcal{A},\mathcal{B}) \geq (\rho/M)OPT(\mathcal{A},\mathcal{B})$, any feasible solution can have at most $M/\rho = \Theta(M/\delta)$ many items with medium or high profit using $1/\rho = \Theta(1/\delta)$. Using the same arguments as for the high profit items and $M = \lceil 1/\delta \log^2(1/\delta) \rceil$ we obtain:

Lemma 2. *[4] There is a set CA_m of medium profit items in $\mathcal{A}$ with $|CA_m| \leq O(1/\delta^3 \log^3(1/\delta))$ such that the profit loss of an optimum solution which selects only medium profit items from CA_m is at most $3\epsilon' OPT(\mathcal{A}, \mathcal{B})$ for $\epsilon' \leq 1/2$.*

In the next phase of the algorithm, we assign medium profit items to the bins in B_{t+1}. Depending on the sizes of these items we do the following steps.
Step A: Consider the medium profit items with $size(a_i) \in (\frac{\delta Area_{Rem}}{2K \log^3(1/\delta)}, 1]$ (where K is a constant specified later). Then, there are at most $\lfloor 2K/\delta \log^3(1/\delta) \rfloor$ many items of this form in B_{t+1}. Next, we guess the medium profit items of large size for bin group B_{t+1}. This can be done again via a guessing step to select the candidates. Afterwards, we assign the chosen candidates to the bins (if possible). Let $A^{(2)}_{guess}$ be the chosen candidate set. The number of choices and assignments using $K = O(1)$ is bounded by $(|CA_m| + 1)^{\lfloor 2K/\delta \log^3(1/\delta) \rfloor} = 2^{O(1/\delta \log^4(1/\delta))}$ and $M^{\lfloor 2K/\delta \log^3(1/\delta) \rfloor} = 2^{O(1/\delta \log^4(1/\delta))}$, respectively.
Step B: Now we consider medium profit items with size smaller than or equal to $\frac{\delta Area_{Rem}}{2K \log^3(1/\delta)}$. The main idea is to round the medium sizes $size(a_i) \in [\delta^6 Area_{Rem}, \frac{\delta Area_{Rem}}{2K \log^3(1/\delta)}]$ corresponding to the optimum solution using linear grouping over sizes $(2^{-(r+1)}, 2^{-r}]$ and to reduce the number of different medium sizes in the instance (similar to the algorithm by Karmarkar and Karp [5] for bin packing). Let Opt be an optimum set of items and $Opt_{medium}, Opt_{small} \subseteq Opt$ be a subset of medium profit items with medium and small size placed into B_{t+1}. Let I_r be the set of all medium items in Opt_{medium} whose sizes lie in $(2^{-(r+1)}, 2^{-r}]$ where $2^r > \frac{K \log^3(1/\delta)}{\delta Area_{Rem}}$. For each r let J_r and J'_r be the instances obtained by applying linear grouping with group size $g = \lceil \frac{2^r \delta Area_{Rem}}{K \log^3(1/\delta)} \rceil$. To apply linear grouping divide each set I_r into groups $G_{r,1}, \ldots, G_{r,q_r}$ such that $G_{r,1}$ contains the g largest items in I_r, $G_{r,2}$ contains the next g largest items and so on. Each group is rounded up to the largest size within the group. Let $G'_{r,i}$ be the multi-set of items obtained by rounding the size of each item in $G_{r,i}$. Then, $J_r = \bigcup_{i \geq 2} G'_{r,i}$ and $J'_r = G'_{r,1}$. In the full paper we prove the following result:

Lemma 3. *The rounded medium items in $\bigcup_r (J_r \cup J'_r)$ and all small items with medium profit fit into B_{t+1} plus one additional bin of size $\frac{\delta Area_{Rem}}{4 \log^2(1/\delta)}$ for $K = 56$ and $\delta \leq 1/10$. Furthermore, the number of rounded medium sizes is bounded by $O(1/\delta \log^3(1/\delta))$ and packing a set of items with rounded medium sizes into the bins in B_{t+1} and an additional small bin can be computed in time $2^{O(1/\delta \log^4(1/\delta))}$.*

Guessing the rounded medium sizes. In our algorithm, we now guess the rounded medium sizes of the medium profit items that are placed in B_{t+1}. The number *med* of medium sizes is bounded by $\lfloor C/\delta \log^3(1/\delta) \rfloor$ where $C \leq 2K + 6$ is a constant. We show in our full paper:

Lemma 4. *Our algorithm guesses the rounded medium sizes $b^{(r)}_1 < \ldots < b^{(r)}_{\ell(r)}$ of Opt_{medium} within each interval $(2^{-(r+1)}, 2^{-r}]$ and the numbers $k^{(r)}_i$ of items of size within $(b^{(r)}_{i-1}, b^{(r)}_i]$ which are placed into B_{t+1}. The number of different guesses is bounded by $2^{O(1/\delta \log^4(1/\delta))}$.*

If our instance does not have at least $k_i^{(r)}$ items with medium profit and size in $(b_{i-1}^{(r)}, b_i^{(r)}]$, then we discard the corresponding guess. Note that the algorithm above only selects the structure of the medium profit items, but not the items themselves. For the selection step of the medium sized items we use the following exchange result (for the proof we refer to the full paper).

Lemma 5. *Suppose that there is a packing of a subset $S \subseteq \mathcal{A}$ of items into all bins $\bigcup_{\ell=1}^{t+1} B_\ell$. Then we can generate a modified packing of S into two bin groups $\mathcal{B}_1 = \bigcup_{\ell=1}^{t} B_\ell$ and $\mathcal{B}_2 = B_{t+1}$ plus one additional bin of size $\frac{\delta Area_{Rem}}{4\log^2(1/\delta)}$ such that $\mathcal{B}_2$ contains the larger medium sizes for each subinterval $(b_{i-1}^{(r)}, b_i^{(r)}]$ where $\min_r b_0^{(r)} \geq \delta^6 Area_{Rem}$ and $\max_{r,i} b_i^{(r)} \leq \frac{\delta Area_{Rem}}{2K\log^3(1/\delta)}$.*

We use this exchange step above for all medium items with medium profit. Sort all medium items of the set CA_m in non-decreasing order of their sizes and guess the smallest index of a medium profit item that is packed into $\mathcal{B}_2 = B_{t+1}$ for each subinterval $(b_{i-1}^{(r)}, b_i^{(r)}]$. Using the modification above $\mathcal{B}_2$ contains only items with indices larger than or equal to the guessed index for each subinterval. Since we have a constant number of candidates in each subinterval (bounded by a polynomial in $1/\delta$) and the number of subintervals is $O(1/\delta \log^3(1/\delta))$, we can guess all these indices in time $(|CA_m|)^{O(1/\delta \log^3(1/\delta))} = 2^{O(1/\delta \log^4(1/\delta))}$. In each subinterval we choose $k_i^{(r)}$ items in CA_m with index larger than or equal to the guessed index. A greedy algorithm that takes $k_i^{(r)}$ feasible items ordered by their profits generates the best solution for the guessed index. Let $\bar{K}_{medium}$ be the selected set for the guess above. Let $A_{guess}^{(3)} = \bar{K}_{medium} \cup \bar{K}_{small}$ be the selected set of medium profit items of medium and small size for $\mathcal{B}_2 = B_{t+1}$. Let $A_i^{(r)}$ be the set of medium profit items in the subinterval $(b_{i-1}^{(r)}, b_i^{(r)}]$ with index smaller than the guessed index. Furthermore, let $\bar{K}_{large}$ be the set of all medium profit items with large size. Note that $\mathcal{A}_1 = \bigcup_{i,r} A_i^{(r)} \cup (CA_h \setminus A_{guess}^{(1)}) \cup (\bar{K}_{large} \setminus A_{guess}^{(2)})$ consists of items with medium and large profit that may be selected for $\mathcal{B}_1 = \bigcup_{\ell=1}^{t} B_\ell$. Furthermore, the set $\mathcal{A}_2$ consists of items with small profit $\leq (\rho/M)2(1+\epsilon')APP(\mathcal{A},\mathcal{B})$ that may be selected for $\mathcal{B}_1 \cup \mathcal{B}_2$. On the other hand, $\bar{A}_i^{(r)}$ denotes the set of medium profit items with index larger than or equal to the guessed index for subinterval $(b_{i-1}^{(r)}, b_i^{(r)}]$. Then, $F = \bigcup_{i,r} \bar{A}_{i,r} \cup \bar{K}_{small} \cup \bigcup_{k=1}^{2} A_{guess}^{(k)}$ is the forbidden set of medium profit items for $\mathcal{B}_1 = \bigcup_{\ell=1}^{t} B_\ell$.

3.3 Linear Program Relaxation

In the next phase we select the other items in $\mathcal{A}_1 \cup \mathcal{A}_2$ via a linear program (LP). Let $cap = \sum_{b_k \in B_{t+1}} \bar{c}(b_k) - size(A_{guess}^{(2)}) - size(J_{medium})$ be the remaining total capacity in the largest bins, where J_{medium} is the total size of all rounded medium size items with medium profit for B_{t+1} without the largest group for each interval $(2^{-(r+1)}, 2^{-r}]$. Note that $size(J_{medium}) \leq size(Opt_{medium})$ is a

lower bound on the size of Opt_{medium} of medium sized items with medium profit packed into B_{t+1} (corresponding to the optimum guess). For simplicity suppose that the first $n' \leq n$ items have small profit and fit into the knapsack of size *cap*; i.e. $\{a_1, \ldots, a_{n'}\} \subset \mathcal{A}_2$. Next suppose that the next $n'' - n'$ items have either a small profit and size larger than *cap* or have medium or high profit; i.e. $\{a_{n'+1}, \ldots, a_{n''}\} = \mathcal{A}_1 \cup (\mathcal{A}_2 \setminus \{a_1, \ldots, a_{n'}\})$. For each bin b_k with capacity c_k, let $C_1^{(k)}, \ldots, C_{H_k}^{(k)}$ be the set of all configurations for the bin b_k; where a configuration is a subset $S \subseteq \mathcal{A}$ of items with total size $\sum_{a \in S} size(a) \leq c_k$. We use the following linear program:

$$
\begin{array}{lll}
\max \sum_{i=1}^{n''} profit(a_i) x_i & & \\
\sum_{k=1}^{m-M} \sum_{j: a_i \in C_j^{(k)}} y_j^{(k)} + z_i = x_i & \text{for} & i = 1, \ldots, n' \\
\sum_{k=1}^{m-M} \sum_{j: a_i \in C_j^{(k)}} y_j^{(k)} = x_i & \text{for} & i = n'+1, \ldots, n'' \\
\sum_{j=1}^{H_k} y_j^{(k)} \leq 1 & \text{for} & k = 1, \ldots, m-M \\
\sum_{i=1}^{n'} size(a_i) z_i \leq cap & & \\
y_j^{(k)} \geq 0 & \text{for} & j = 1, \ldots, H_k \text{ and } k = 1, \ldots, m-M \\
z_i \in [0,1] & \text{for} & i = 1, \ldots, n' \\
x_i \in [0,1] & \text{for} & i = 1, \ldots, n''
\end{array}
$$

The main idea of the relaxation is to use a fractional variable $x_i \in [0,1]$ for each item a_i (which selects a piece of each item) and to distribute the corresponding piece as smaller fractional pieces among the configurations for different bin capacities. The variable $y_j^{(k)}$ in the LP denotes the length of configuration $C_j^{(k)}$ in bin b_k. For each of the first n' items with small profit, we use also a variable z_i to indicate a fractional piece selected for the bins in B_{t+1}.

Lemma 6. *The linear program LP is a relaxation of the MKP instance* $(\mathcal{A}, \mathcal{B})$ *where* $\mathcal{B} = \mathcal{B}_1 \cup \mathcal{B}_2$ *after selecting the items of* $\bigcup_{k=1}^{3} A_{guess}^{(k)}$ *for* $\mathcal{B}_2 = B_{t+1}$ *and the additional bin; i.e. the objective value of the LP is at least the maximum profit of a subset of* $\mathcal{A} \setminus F$ *(where* F *is the forbidden set of medium profit items) packed together with* $\bigcup_{k=1}^{3} A_{guess}^{(k)}$ *into* $\mathcal{B}$ *where we allow to pack further high and medium profit items into* $\mathcal{B}_1$ *and small profit items into* $\mathcal{B}_1 \cup \mathcal{B}_2$.

We suppose that all additional items with small profit that may be placed into $\mathcal{B}_2 = B_{t+1}$ have size at most $\delta\, cap$. We need this property in the rounding strategy later. Note that there are at most $1/\delta$ small profit items of larger size in these bins. The total profit of these items can be bounded by $(1/\delta)2(\rho/M)(1+\epsilon')APP(\mathcal{A}, \mathcal{B})$. In the LP we simply remove a z_i variable, if the size of the corresponding item is too large. This implies for the modified linear program LP' that $OPT(LP') \geq OPT(LP) - 2\rho(1+\epsilon')OPT(\mathcal{A}, \mathcal{B})$.

Lemma 7. [4] *We can compute an approximate solution* $(\bar{x}, \bar{y}, \bar{z})$ *of the modified* LP' *in time poly*$(n, 1/\alpha)$ *where* $\sum_{j=1}^{H_k} \bar{y}_j^{(k)} \leq (1+2\alpha)$, $\sum_{i=1}^{n'} size(a_i)\bar{z}_i \leq cap(1+2\alpha)$, *and whose objective value is at least* $(1-3\alpha)OPT(LP')$.

The solution of the linear program LP' can be transformed into another solution $(\tilde{x}, \tilde{y}, \tilde{z})$ with objective value at least $(1-5\alpha)OPT(LP')$ without violating the constraints above. Here again we simply scale the values $\tilde{y}_j^{(k)} = \bar{y}_j^{(k)}/(1+2\alpha)$, $\tilde{z}_i = \bar{z}_i/(1+2\alpha)$ and $\tilde{x}_i = \bar{x}_i/(1+2\alpha)$.

4 New Rounding Strategy

Now we describe how to round the $(\tilde{x}, \tilde{y}, \tilde{z})$ solution of the modified LP. First we generate t stacks and rounded sets of rectangles $L_{wide}^{(\ell)}$ for the blocks B_ℓ, $\ell = 1, \ldots, t$. Consider an item a_i with $\tilde{x}_i > 0$. Let $z_i^{(\ell)} = \sum_{b_k \in B_\ell} \sum_{j:a_i \in C_j^{(k)}} \tilde{y}_j^{(k)}$ be the fraction of item a_i assigned to block B_ℓ. For each block B_ℓ, all large pieces with $size(a_i) > \delta c_{max}^{(\ell)}$ can be interpreted as wide rectangles of the form $(size(a_i), z_i^{(\ell)})$ with width $size(a_i) \leq c_{max}^{(\ell)}$ and height $z_i^{(\ell)} \leq 1$. Next we stack all these rectangles ordered by their widths. We obtain a stack St_ℓ of height H_ℓ (see Figure 2). Now we add to the stack St_ℓ a set X_ℓ of dummy rectangles of width $\delta^2 c_{max}^{(\ell)}$ and height 1 (with the exception of one rectangle with height at most 1) until the modified stack $\overline{St}_\ell$ has total height $\overline{H}_\ell = d_\ell/\delta^2$ where $d_\ell \in \mathbb{Z}^+$ and $(d_\ell - 1)/\delta^2 < H_\ell \leq \bar{H}_\ell$. Note that the total height of the rectangles in X_ℓ is at most $1/\delta^2$. Let $L_{wide}^{(\ell)}$ and $\bar{L}_{wide}^{(\ell)}$ be the sets of all rectangles on stack St_ℓ and $\overline{St}_\ell$, respectively. We split the stack $\overline{St}_\ell$ into $1/\delta^2$ groups of height $\delta^2 \bar{H}_\ell = d_\ell$. If a piece lies in two groups of the stack $\overline{St}_\ell$ (more than two groups is not possible, since the height of each rectangle is at most $1 \leq d_\ell$), then we split the rectangle into two rectangles that fit into their groups completely.

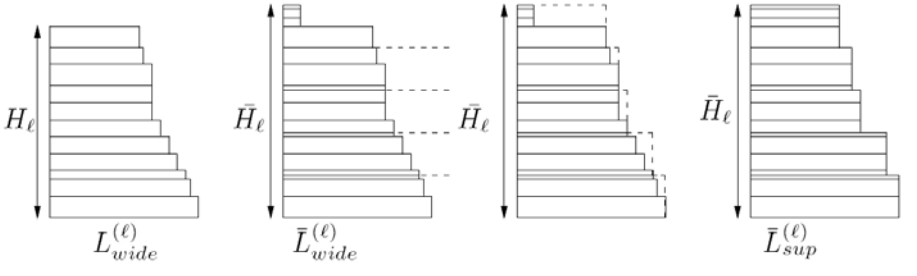


Fig. 2. The construction of the stacks for $L_{wide}^{(\ell)}$, $\bar{L}_{wide}^{(\ell)}$ and $\bar{L}_{sup}^{(\ell)}$

Finally, we round up each rectangle in group j on stack $\overline{St}_\ell$ to the maximal width in group j. Let $\bar{L}_{sup}^{(\ell)}$ be the set of rectangles obtained after the rounding (see also Figure 2 for the construction of the stacks and sets of rectangles). Next we compare the minimum fractional strip packing height for the instances $L_{wide}^{(\ell)}$ and $\bar{L}_{sup}^{(\ell)}$ into a strip with different horizontal layers. Let $c_1 \leq \ldots \leq c_M$ be the widths of the M horizontal layers. The first $M-1$ layers have height 1 and

layer M has unbounded height. The widths $c_1, \ldots, c_M$ are the bin capacities $c_1^{(\ell)}, \ldots, c_M^{(\ell)}$ in block B_ℓ. For a set L of N rectangles of the form $r_i = (w_i, h_i)$ with heights $h_i \leq 1$, let $R^{(k)}$ be a set of rectangles (called a configuration) that fits into a horizontal layer of width c_k; i.e. $\sum_{r_i \in R^{(k)}} w_i \leq c_k$. Let $R_1^{(k)}, \ldots, R_{H_k}^{(k)}$ be the set of configurations for width c_k. Use variables $v_1^{(k)}, \ldots, v_{H_k}^{(k)}$ to denote the heights of the configurations. The linear program $LP(L, B_\ell)$ for an instance L with N rectangles and block B_ℓ has the following form:

$$
\begin{array}{ll}
\min \sum_{j=1}^{H_M} v_j^{(M)} & \\
\sum_{k=1}^{M} \sum_{j: r_i \in R_j^{(k)}} v_j^{(k)} & = h_i \text{ for } \; i = 1, \ldots, N, \\
\sum_{j=1}^{H_k} v_j^{(k)} & \leq 1 \;\; \text{for} \;\; k = 1, \ldots, M-1, \\
v_j^{(k)} & \geq 0 \;\; \text{for} \;\; j = 1, \ldots, H_k \;\; \text{and} \;\; k = 1, \ldots, M.
\end{array}
$$

For each instance L, let $LIN(L, B_\ell)$ be the value of the linear program above where the widths $c_1, \ldots, c_M$ are the capacities of the bins in block B_ℓ. This value is the minimum height of a fractional strip packing into a strip consisting of M horizontal layers of widths $c_1 \leq \ldots \leq c_M$. Note that we count in the objective function of the LP above only the packing into the widest layer of width c_M. Let $AREA(L)$ be the total area of all rectangles in L. Since $L_{wide}^{(\ell)}$ fits fractionally into the M bins, $LIN(L_{wide}^{(\ell)}, B_\ell) \leq 1$. We show in our full paper:

Lemma 8. *For the blocks $B_1, \ldots, B_t$, we obtain*

$$
\begin{array}{ll}
LIN(\bar{L}_{sup}^{(\ell)}, B_\ell) & \leq \delta M + 3, \\
AREA(\bar{L}_{sup}^{(\ell)}) & \leq (1+\delta) AREA(L_{wide}^{(\ell)}) + 2c_{max}^{(\ell)}.
\end{array}
$$

We build one additional stack for the small profit items placed into B_{t+1}. Sort the small profit items $a_1, \ldots, a_{n'}$ in non-decreasing order of their sizes and put the items with sizes in $[\delta^2 cap, \delta cap]$ as rectangles $(size(a_i), z_i)$ with width $size(a_i)$ and height z_i in the order above on a stack. This generates a stack St_{t+1} with a set $L_{wide}^{(t+1)}$ of rectangles of total height H_{t+1}. Similar to the above construction, we generate a modified stack $\overline{St}_{t+1}$ and the set $L_{sup}^{(t+1)}$ of rounded rectangles. Here we obtain $Area(\bar{L}_{sup}^{(t+1)}) \leq (1+\delta) Area(L_{wide}^{(t+1)}) + \delta cap$.

The set $\bar{L}_{sup}^{(\ell)}$ consists of a set of rectangles with at most $1/\delta^2$ (which is constant) different widths $w_1^{(\ell)} > \ldots > w_{a(\ell)}^{(\ell)}$ for each $\ell = 1, \ldots, t+1$ where $a(\ell) \leq 1/\delta^2$. For each width $w_j^{(\ell)}$, let $\beta_j^{(\ell)}$ be the total height of rectangles in $\bar{L}_{sup}^{(\ell)}$ with rounded width $w_j^{(\ell)}$ for $\ell = 1, \ldots, t+1$. Since each stack $\overline{St}_\ell$ has height a_ℓ/δ^2 where $a_\ell \in \mathbb{Z}^+$, the numbers $\beta_j^{(\ell)}$ are integral for each group j and each block B_ℓ.

Let $L_{narrow}^{(\ell)}$ be the set of *narrow rectangles* $(size(a_i), z_i^{(\ell)})$ with $size(a_i) \leq \delta c_{max}^{(\ell)}$ allocated to block B_ℓ for $\ell = 1, \ldots, t$. The total area $AREA(L_{narrow}^{(\ell)})$

is $\sum_{i:size(a_i)\leq\delta c_{max}^{(\ell)}} z_i^{(\ell)} size(a_i)$. We divide this area into smaller areas as follows. For each interval $int_{\ell,k} = (\frac{\delta}{(1+\delta)^k}c_{max}^{(\ell)}, \frac{\delta}{(1+\delta)^{k-1}}c_{max}^{(\ell)}]$ with $k \in \mathbb{N}$, let $Area(\ell, int_{\ell,k}) = \sum_{i:size(a_i)\in int_{\ell,k}} z_i^{(\ell)} size(a_i)$ be the total area of pieces of items a_i with $size(a_i) \in int_{\ell,k}$ allocated to bins in block B_ℓ. Using the smallest original item size in interval $int_{\ell,k}$, the maximum number of possible items in $int_{\ell,k}$ with this total area is $\eta_{\ell,k} = \left\lceil \frac{Area(\ell,int_{\ell,k})}{\frac{\delta}{(1+\delta)^k}c_{max}^{(\ell)}} \right\rceil$. In our algorithm, we take all intervals $int_{\ell,k}$ for which $\frac{\delta}{(1+\delta)^k}c_{max}^{(\ell)} \geq \frac{1}{2n}c_{max}^{(\ell)}$. Let $index(\ell)$ be the largest index k such that this inequality is satisfied. The inequality is also equivalent to $2\delta n \geq (1+\delta)^k$ or $\log(2\delta n) \geq k\log(1+\delta)$. This implies an upper bound $k \leq \frac{\log(2\delta n)}{\log(1+\delta)}$. Therefore, we set $index(\ell) = \lfloor \frac{\log(2\delta n)}{\log(1+\delta)} \rfloor + 1$ and obtain that $index(\ell) = O([\log(\epsilon)+\log(n)]/\log(1+\epsilon))$ using $\delta = \Theta(\epsilon)$. For items with $size(a_i) \leq \frac{1}{2n}c_{max}^{(\ell)}$ we use an additional interval $int_{\ell,index(\ell)+1} = (-\infty, \delta/(1+\delta)^{index(\ell)}]$ and set $\eta_{\ell,index(\ell)+1} = n$. Note that the number of intervals is bounded by a polynomial in n and $1/\epsilon$. In the full paper we prove:

Lemma 9. *If $A^{(\ell)} \subseteq \mathcal{A}$ is a set of small items with $|\{a_i \in A^{(\ell)} | size(a_i) \in int_{\ell,k}\}| \leq \eta_{\ell,k}$ for each k, then $Area(A^{(\ell)}) \leq (1+\delta)Area(L_{narrow}^{(\ell)}) + (3/2+\delta)c_{max}^{(\ell)}$ for $\ell = 1, \ldots, t$.*

For the small profit items in B_{t+1} with size $< \delta^2 cap$ we form intervals $int_{t+1,k} = (\frac{\delta^2 cap}{(1+\delta)^k}, \frac{\delta^2 cap}{(1+\delta)^{k-1}}]$ for $k \in \mathbb{N}$. Similar to the construction above we define the area values $Area(t+1, int_{t+1,k})$ and set the values $\eta_{t+1,k}$ for $k = 1, \ldots, index(t+1)+1$. Summing over all intervals $int_{t+1,k}$, the total area of a set $A^{(t+1)}$ with at most $\eta_{t+1,k}$ items with size in $int_{t+1,k}$ is at most $(1+\delta)Area(L_{narrow}^{(t+1)}) + (2+\delta)\delta cap$ where $L_{narrow}^{(t+1)}$ is the set of rectangles $(size(a_i), z_i)$ over all small profit items a_i with $size(a_i) < \delta^2 cap$.

Flow Network. Now we set up a flow network $G = (N, E)$ of the following form. The vertex set N consists of a source s and sink t, a node x_i for each item a_i and several nodes for each block B_ℓ with (at most) M bins. For each block, we have a node $y_{\ell,j}$ for each rounded wide width $w_j^{(\ell)}$ where $j = 1, \ldots, a(\ell)$ and a node $\bar{y}_{\ell,k}$ for each interval $int_{\ell,k}$ where $k = 1, \ldots, index(\ell)+1$. The edge set E is defined by $\{(s, x_i) | i = 1, \ldots, n\} \cup \{(x_i, y_{\ell,j}) | w_{j+1}^{(\ell)} \leq size(a_i) \leq w_j^{(\ell)}\} \cup \{(x_i, \bar{y}_{\ell,k}) | size(a_i) \in int_{\ell,k}\} \cup \{(y_{\ell,j}, t) | \ell = 1, \ldots, t+1 \text{ and } j = 1, \ldots, a(\ell)\} \cup \{(\bar{y}_{\ell,k}, t) | \ell = 1, \ldots, t+1 \text{ and } k = 1, \ldots, index(\ell)+1\}$. All edges have lower capacities 0. The upper capacities of the edges (s, x_i), $(x_i, y_{\ell,j})$ and $(x_i, \bar{y}_{\ell,k})$ are 1. In addition the capacity of the edge $(y_{\ell,j}, t)$ is $\beta_j^{(\ell)}$ and the capacity of the edge $(\bar{y}_{\ell,k}, t)$ is $\eta_{\ell,k}$. Note that all capacities are integral and that the number of vertices in G is bounded by a polynomial in n and $1/\epsilon$ using $\delta = O(\epsilon)$. Furthermore, we have cost values for each edge: for each edge (s, x_i) the cost value $c(s, x_i) = profit(a_i)$ and for each other edge the cost value is 0.

Lemma 10. *There is a fractional flow in the network $G = (N, E)$ with profit at least $(1 - 5\alpha)OPT(LP)$.*

Taking negative profit values, there is a minimum cost flow in the network with cost $\leq -(1 - 5\alpha)OPT(LP)$. Since we have a totally unimodular constraint matrix, each basic solution in the linear program corresponding to the flow problem is integral. Therefore, there is a minimum cost integral flow (among all flow values) in the network with the same cost. By computing the minimum cost flow for each integral flow value $v = 1, \ldots, n$ and taking the best solution, we obtain an integral flow $g : E \to \mathbb{N}$ in the network with profit $\geq (1 - 5\alpha)OPT(LP)$. This integral flow gives us a subset of selected items $A_{select} = \{a_i | g((s, x_i)) = 1\}$ with profit close to the optimum profit. For each block B_ℓ, let $A^{(\ell)}_{wide} = \{a_i | \exists j \in \{1, \ldots, a(\ell)\}$ with $g(x_i, y_{\ell,j}) = 1\}$ and $A^{(\ell)}_{narrow} = \{a_i | \exists k \in \{1, \ldots, index(\ell)+1\}$ with $g(x_i, \bar{y}_{\ell,k}) = 1\}$ be the set of wide and narrow items for the block B_ℓ, respectively. Let $\bar{A}^{(\ell)}_{wide}$ and $\bar{A}^{(\ell)}_{narrow}$ be corresponding sets with rectangles of width equal to the $size(a)$ and height 1. We obtain the following result.

Lemma 11. *The algorithm based on minimum cost flow problems above computes sets $A^{(\ell)}_{wide}, A^{(\ell)}_{narrow}$ of items for each block B_ℓ ($\ell = 1, \ldots, t + 1$) with $profit(\bigcup_\ell A^{(\ell)}_{wide} \cup A^{(\ell)}_{narrow}) \geq (1 - 5\alpha)OPT(LP')$ such that the following properties are satisfied:*

- $|\{a_i \in A^{(\ell)}_{wide} | g(x_i, x_{\ell,j}) = 1\}| \leq \beta^{(\ell)}_j$ *for each* $j = 1, \ldots, a(\ell)$ *and* $\ell = 1, \ldots, t + 1$.
- $|\{a_i \in A^{(\ell)}_{narrow} | g(x_i, \bar{y}_{\ell,k}) = 1\}| \leq \eta_{\ell,k}$ *for each* $j = 1, \ldots, index(\ell) + 1$ *and* $\ell = 1, \ldots, t + 1$.

In addition, $LIN(A^{(\ell)}_{wide}, B_\ell) \leq \delta M + 3$, $Area(A^{(\ell)}_{wide}) \leq (1+\delta)Area(L^{(\ell)}_{wide}) + 2c^{(\ell)}_{max}$, and $Area(A^{(\ell)}_{narrow}) \leq (1 + \delta)Area(L^{(\ell)}_{narrow}) + (3/2 + \delta)c^{(\ell)}_{max}$ for $\ell = 1, \ldots, t$. Furthermore, $Area(A^{(t+1)}_{wide} \cup A^{(t+1)}_{narrow}) \leq (1+\delta)Area(L^{(t+1)}_{wide} \cup L^{(t+1)}_{narrow}) + 4\delta cap$.

5 Packing and Shifting Arguments

In the following we discuss that most of the selected items can be placed into the bins. Let $Opt_{ILP}(S, B_\ell)$ be the minimum value of an integral solution for the $LP(S, B_\ell)$ for a selected subset S of items. This value gives the number of bins of capacity $c^{(\ell)}_{max}$ used for S, where the first $M - 1$ bins of capacities $c^{(\ell)}_1, \ldots, c^{(\ell)}_{M-1}$ are not counted, but can be used as additional space. In the full paper we show the following bound for the number of extra bins.

Lemma 12.

$$OPT_{ILP}(A^{(\ell)}_{wide} \cup A^{(\ell)}_{narrow}) \leq C' \log^2(1/\delta)$$

where C' is a constant.

Since the proof is constructive, we obtain also an algorithm that generates a packing for $A^{(\ell)}_{wide} \cup A^{(\ell)}_{narrow}$ into $M + \lfloor \bar{C}' \log^2(1/\delta) \rfloor$ bins: a subset $A^{(\ell)}_1$ fits into block B_ℓ and the remaining set $A^{(\ell)}_2$ fits into $\lfloor \bar{C}' \log^2(1/\delta) \rfloor$ bins of size $c^{(\ell)}_{max}$ where $\bar{C}'$ is a constant. Via the shifting argument we can select a subset $X_{\ell+1} \subset A^{(\ell)}_2 \cup A^{(\ell+1)}_1$ that fits into block $B_{\ell+1}$ for $\ell = 1, \ldots, t-1$.

Lemma 13. *For each $\ell = 1, \ldots, t-2$, we can select a subset $X_{\ell+1} \subseteq A^{(\ell)}_2 \cup A^{(\ell+1)}_1$ with profit at least $(1 - \bar{C}'\delta) profit(A^{(\ell)}_2 \cup A^{(\ell+1)}_1)$ that can be packed into block $B_{\ell+1}$.*

For the first block B_1 we take $X_1 = A^{(1)}_1$. For the last block B_{t+1} with selected set $A^{(t+1)} = A^{(t+1)}_{wide} \cup A^{(t+1)}_{narrow}$ the situation is a bit more complicated and is discussed in the full paper. We show there that the profit of the entire selected set is at least $(1 - \max\{28 + \bar{C}', 23 + 2\bar{C}'\}\delta) OPT(\mathcal{A}, \mathcal{B})$ (using $\epsilon' \leq 1/6$ and $\delta = \rho = \epsilon'$). For $\delta \leq \epsilon / \max\{28 + \bar{C}', 23 + 2\bar{C}'\}$, the profit of our solution is at least $(1 - \epsilon) OPT(\mathcal{A}, \mathcal{B})$. In our algorithm we set $\delta = 1/(5\lceil \max\{28 + \bar{C}', 23 + 2\bar{C}'\}/5\epsilon \rceil)$ and obtain the property that $1/\delta$ is integral. The running time of our algorithm is dominated by the guessing steps for B_{t+1} and can be bounded by $2^{O(1/\epsilon \log^4(1/\epsilon))} poly(n, 1/\epsilon) \leq 2^{O(1/\epsilon \log^4(1/\epsilon))} poly(n) \leq 2^{O(1/\epsilon \log^4(1/\epsilon))} + poly(n)$.

References

1. Caprara, A., Kellerer, H., Pferschy, U.: The multiple subset sum problem. SIAM Journal of Optimization 11, 308–319 (2000)
2. Chekuri, C., Khanna, S.: A PTAS for the multiple knapsack problem. SIAM Journal on Computing 35, 713–728 (2006)
3. Fellows, M.R.: Blow-Ups, Win/Win's, and Crown Rules: Some New Directions in *FPT*. In: Bodlaender, H.L. (ed.) WG 2003. LNCS, vol. 2880, pp. 1–12. Springer, Heidelberg (2003)
4. Jansen, K.: Parameterized approximation scheme for the multiple knapsack problem. SIAM Journal on Computing 39, 1392–1412 (2009)
5. Karmarkar, N., Karp, R.M.: An efficient approximation scheme for the one-dimensional bin-packing problem. In: Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science, FOCS 1982, pp. 312–320 (1982)
6. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Berlin (2004)
7. Kenyon, C., Remila, E.: Approximate strip packing. Mathematics of Operations Research 25, 645–656 (2000)
8. Marx, D.: Parametrized complexity and approximation algorithms. The Computer Journal 51, 60–78 (2008)
9. Scheithauer, G., Terno, J.: Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem. European Journal of Operational Research 20, 93–100 (1997)
10. Shmonin, G.: Parameterised integer programming, integer cones, and related problems, PhD thesis, Universität Paderborn (2007)

Counting Maximal Independent Sets in Subcubic Graphs

Konstanty Junosza-Szaniawski and Michał Tuczyński

Warsaw University of Technology
Faculty of Mathematics and Information Science
Pl. Politechniki 1/207, 00-661 Warsaw, Poland
{k.szaniawski,m.tuczynski}@mini.pw.edu.pl

Abstract. The main result of this paper is an algorithm counting maximal independent sets in graphs with maximum degree at most 3 in time $O^*(1.2570^n)$ and polynomial space.

1 Introduction

Recently much attention has been paid to algorithmic aspects of some counting problems. Although many of the problems (e.g. counting independent sets, maximal independent set or matchings in a graph) are known to be #P-Complete (see Vadhan [15], Greenhill [9]), a remarkable progress has been done in designing exponential time algorithms solving them. Dahllöf, Jonsson, Wahlström [3] constructed algorithms that count maximum weight models of 2-SAT and 3-SAT formulas in time $O^*(1.2561^n)$ and $O^*(1.6737^n)$, respectively. The former bound was later improved to $O^*(1.2461^n)$ by Fürer and Kasiviswanathan [7] and subsequently to $O^*(1.2377^n)$ by Wahlström [16]. The latter bound was improved by Kutzkov [13] to $O^*(1.6423^n)$. Maximal independent sets can be counted by an algorithm of Gaspers, Kratsch and Liedloff in time $O^*(1.3642^n)$. This algorithm restricted to subcubic graphs performs in time $O^*(1.3532^n)$ (this complexity bound can be proved by applying the measure and conquer method, where weights of vertices of degree 1, 2, and 3 are equal to $w_1 = 0.361958$ and $w_2 = w_3 = 0.436386$, respectively). Maximal independent sets in a tree can be counted in polynomial time (see Jou, Chang [10]). All algorithms mentioned above work in polynomial space.

We can construct a pathwidth-based algorithm counting all maximal independent sets in a graph (similar to the one presented in Alber and Niedermeier [1]). Such an algorithm works in time $O * (3^{pw(G)})$, where $pw(G)$ is the pathwidth of G. Using the bound on the pathwidth of graphs of maximum degree at most 3 given by Fomin and Hoie in [5] we get the time complexity $O^*(3^{(1/6+\epsilon)n}) = O^*(1.2010^n)$. This method can be also applied for graphs with vertices of degree larger than 3 (see [6]). However pathwidth-based algorithms use exponential amount of space.

Independent sets in a graph naturally correspond to models of 2-SAT formulas with all variables negated. In particular the algorithm of Wahlström [16] can be

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 325–336, 2012.

applied to count all independent sets and all independent sets of maximum size in a graph. In fact if we use this algorithm as a subroutine in the algorithm of Björklund, Husfeldt and Koivisto [2] (based on the inclusion-exclusion principle) then we obtain a fastest known algorithm for the graph coloring problem which works in polynomial space.

In this paper we present an algorithm for counting maximal independent sets (equivalently minimal coverings) in subcubic graphs. The case of graphs with maximum degree at most 3 is the first non trivial case, because maximal independent sets in graphs with maximum degree at most 2 can be counted in polynomial time. Our algorithm is a recursive algorithm with four branching rules. To control maximality of counted independent sets we use a new type of edges - red edges. Each counted independent set must contain at least one end of every red edge, otherwise it is not maximal in the instance graph. Our algorithm can be modified to count minimal models of 2-SAT formulas, where no variable appears more than 3 times. However, for simplicity, we present only the version for counting maximal independent sets in subcubic graphs.

In the running time analysis of recursive algorithms a method sometimes called the "measure and conquer" method (introduced by Kullman [12], see also [4]) is very useful. In particular this method was used in analyzing the running time of the algorithm in Dahllöf *et al.* [3]. We use similar measure of graphs as they did. We define the measure of sparse graphs to be equal to the number of vertices of degree 3. Then we prove that the higher density of a graph is, the better configuration for branching can be found. To take advantage of this we define (after [3]) a measure which is a piece-wise linear function of the number of the vertices and the number of the edges. Finally, we reduce the constant c in the running time bound $O^*(c^n)$ of an algorithm counting maximal independent sets is a graph of maximum degree at most 3 from $c = 1.3532$ (Gaspers *et al.*) to $c = 1.2570$ (the algorithm presented in this paper).

2 Preliminaries

We denote by $V(G)$ the vertex set of a graph G and by $E(G)$ its edge set. Let $n(G)$ and $m(G)$ be the number of vertices and the number of edges of G, respectively. We write n instead of $n(G)$ and m instead of $m(G)$ whenever it does not lead to a confusion. By $dist(v, u)$ we denote the number of edges of a shortest $v - u$ path. An *open neighborhood* of a vertex v is the set of vertices $N(v) = \{u \in V(G) : uv \in E(G)\}$ and a *closed neighborhood* of v is $N[v] = N(v) \cup \{v\}$. Let $d(v) = |N(v)|$ be the *degree* of a vertex v. By $n_i(G)$ and $n_{\geq i}(G)$ we denote the number of vertices of degree i and at least i in G, respectively. A vertex of degree 0 is called *isolated* and a vertex of degree 1 is called a *leaf*. By $\Delta(G)$ (resp. $\delta(G)$) we mean the maximum (resp. minimum) degree of a vertex in G. Let $NN[v] = \{u \in V : \text{dist}(v, u) \leq 2\}$, $NN(v) = NN[v] - \{v\}$, $S(v) = \sum_{u \in N[v]} d(u)$, and let $SS(v)$ denote the sequence $(S(v), S(v_1), \ldots, S(v_{d(v)}))$, where $\{v_1, \ldots, v_{d(v)}\} = N(v)$ and $S(v_1) \geq \ldots \geq S(v_{d(v)})$. We write $SS(v) \leq_L SS(u)$ if the sequence $SS(v)$ is not greater lexicographically than $SS(u)$. We say

that u is a *topological neighbor* of v or that u and v are topological neighbors if there exists a $v-u$ path with all internal vertices of degree 2. For a vertex v of degree 3 let $n_2(v) = \frac{1}{2}|\{u : d(u) = 2$ and there exists a $v-u$ path consisting only of vertices of degree 2 and $v\}|$. Notice that for any connected graph G with $\delta(G) = 2$ and $\Delta(G) = 3$ holds $n_2 = \sum_{v:d(v)=3} n_2(v)$.

For a vertex set $U \subset V(G)$, $G[U]$ is the subgraph of G induced by U and $G - U = G[V(G) - U]$. If $U = \{u\}$, then we write $G - u$ instead of $G - \{u\}$. For a subgraph H of G instead of $G - V(H)$, we write $G - H$. For the empty graph $(\emptyset, \emptyset)$ we simple write $\emptyset$. A set U of vertices of G is a *cut set* if $G - U$ has more components than G. A vertex u is a *cut vertex*, if $U = \{u\}$ is a cut set. A graph G is called *k-connected* if $n(G) > k$ and $G - U$ is connected for every set $U \subset V(G)$ such that $|U| < k$.

A set $S \subset V(G)$ is an *independent set* (or an *IS* for short) in G, if no edge in G has both ends in S. An independent set is *maximal* (or a *MIS* for short) if it is not contained as a proper subset in any other independent set. A set $T \subset V(G)$ is a *covering* if every edge in G has at least one end in T. A covering is minimal if it contains no covering as a proper subset. Notice that T is a minimal covering if and only if $V(G) - T$ is a MIS.

To control maximality of ISs in the counting process we introduce *red edges* and we count only the ISs containing at least one end of every red edge. For example if $N(u) = \{v, x_1, x_2\}$, and we want to count maximal independent sets containing none of u and v in a branch of the tree of recursive calls of our algorithm, then we add a red edge x_1x_2 before removing u and v, since at least one of x_1, x_2 must be contained in a MIS of G in this branch. We call the original edges of the graph *blue edges*. It is possible that an edge is both blue and red.

A *blue-red graph* (or a *br-graph* for short) G is a triple $G = (V, E_b, E_r)$ such that $(V, E_b \cup E_r)$ is a graph. Edges from E_b are called blue edges, edges from E_r are called red edges. The sets V, E_b, E_r of the br-graph$G$ are referred to as $V(G), E_b(G), E_r(G)$, respectively. If $uv \in E_b$ (resp. $uv \in E_r$), then we call u a blue (resp. red) neighbor of v. Let $N_b(v) = \{u \in V : vu \in E_b\}$, $N_b[v] = N_b(v) \cup \{v\}$ and $N_r(v) = \{u \in V : vu \in E_r\}$. By $d(v) = |N_b(v) \cup N_r(v)|$ we denote the degree of a vertex v in a *br*-graph. For $S \subset V$ let $N_b(S) = \bigcup_{v \in S} N_b(v)$ and $N_b[S] = S \cup N_b(S)$.

Let $IS(G)$ be the family of all sets $S \subset V(G)$ satisfying the conditions:

(I1) If vu is a blue edge, then at most one of vertices v and u belongs to S.

(I2) If vu is a red edge, then at least one of vertices v and u belongs to S.

For disjoint subsets A^+, A^- of $V(G)$ let $IS(G, A^+, A^-) = \{S \in IS(G) \colon A^+ \subseteq S \subseteq V(G) - A^-\}$.

In our algorithm we use a function $\mathbf{c} : \{1, 0, \bar{0}\} \times V(G) \to \{0, 1, \ldots\}$ called the *cardinality function*. For convenience we write $\mathbf{c_1}(v)$, $\mathbf{c_0}(v)$ and $\mathbf{c_{\bar{0}}}(v)$ instead of $\mathbf{c}(1, v)$, $\mathbf{c}(0, v)$ and $\mathbf{c}(\bar{0}, v)$, respectively. During the course of the algorithm we will remove some vertices. Similarly as in [3] we will use the cardinality function to store information about the factor of the number of MIS in a primal graph comparing to the current one (with some vertices removed). Our cardinality

function is, however, more complicated because unlike in [3], we have to control maximality of the independent sets. This is why we introduce the part $\mathbf{c}_{\bar{\mathbf{0}}}$ of the cardinality function.

Given a cardinality function $\mathbf{c}$ and $S \in IS(G)$ we define

$$C_G(S, \mathbf{c}) = \prod_{w \in S} \mathbf{c_1}(w) \prod_{w \in N_b(S)} \mathbf{c_0}(w) \prod_{w \in V(G) - N_b[S]} \mathbf{c}_{\bar{\mathbf{0}}}(w) \quad .$$

We omit the index referring to the graph if the reference is clear.
Let

$$\#IS(G, \mathbf{c}, A^+, A^-) = \sum_{S \in IS(G, A^+, A^-)} C(S, \mathbf{c}) \quad .$$

We write $\#IS(G, \mathbf{c})$ instead of $\#IS(G, \mathbf{c}, \emptyset, \emptyset)$.

To compute the number of MISs in a graph $G = (V, E)$ we apply our algorithm to the br-graph$(V, E, \emptyset)$ and cardinality function $\mathbf{c}$ such that $\mathbf{c_1}(v) = \mathbf{c_0}(v) = 1$ and $\mathbf{c}_{\bar{\mathbf{0}}}(v) = 0$ for every $v \in V$. Notice that in such case, for $S \in IS(G)$, $C(S, \mathbf{c}) = \begin{cases} 1 & \text{if } S \text{ is a MIS in } G \\ 0 & \text{otherwise} \end{cases}$.

To make proofs shorter we will use the following notation. For $S \in IS(G)$ and $U \subset V$ let $C(S/U, \mathbf{c}) = \prod_{w \in S - U} \mathbf{c_1}(w) \prod_{w \in N_b(S) - U} \mathbf{c_0}(w) \prod_{w \in (V(G) - N_b[S]) - U} \mathbf{c}_{\bar{\mathbf{0}}}(w)$.

3 The Algorithm

Our main algorithm MISCount computes the number $\#IS(G, \mathbf{c})$ for a br-graph$G$ with maximum degree 3 and the cardinality function $\mathbf{c}$ satisfying the following technical condition:

(A1) for every vertex v of degree 3, $\mathbf{c_1}(v) = 1$, $\mathbf{c_0}(v) = 1$, and $\mathbf{c}_{\bar{\mathbf{0}}}(v) = 0$.

To count all maximal independent sets of a given graph (V, E) we call MISCount$(G, \mathbf{c})$, for the br-graph$G = (V, E, \emptyset)$ and the cardinality function $\mathbf{c}$ defined as follows $\mathbf{c_1}(v) = \mathbf{c_0}(v) = 1$, and $\mathbf{c}_{\bar{\mathbf{0}}}(v) = 0$ for every vertex $v \in V(G)$. Notice that is such case the condition (A1) is satisfied. In the course of the algorithm MISCount the graph is modified, but only in a way the condition (A1) is still satisfied. The algorithm uses three auxiliary procedures.

Propagation. The procedure Prop takes as input a br-graph$G$, a cardinality function $\mathbf{c}$ satisfying (A1) and two disjoint sets of vertices A^+, A^- satisfying the conditions

(A2) there is no vertex v in A^- such that $d(v) = d_b(v) = 3$ and $N(v) \cap (A^+ \cup A^-) = \emptyset$,

(A3) for every vertex v in A^- such that $d(v) = d_b(v) = 2$ and $N(v) \cap (A^+ \cup A^-) = \emptyset$, $\mathbf{c_1}(v) = 1$, $\mathbf{c_0}(v) = 1$ and $\mathbf{c}_{\bar{\mathbf{0}}}(v) = 0$.

The procedure simplifies given br-graphby performing operations such as removing vertices, adding red edges and changing the values of the cardinality

function until $A^+ \cup A^-$ is empty in such a way that after every execution of the main loop the conditions (A1)-(A3) are satisfied.

Moreover, it removes isolated vertices and leaves. It returns a br-graphG', a cardinality function $\mathbf{c}'$ satisfying (A1) and an integer c such that $\#IS(G, \mathbf{c}, A^+, A^-) = c \cdot \#IS(G', \mathbf{c}')$.

Algorithm 1. Prop$(G, \mathbf{c}, A^+, A^-)$

$c \leftarrow 1$
while *any of steps is applicable* **do**
  if *there exists a vertex* $v \in A^+$ **then**
    if v *has a blue neighbor in* A^+ **then return** $(\emptyset, \mathbf{c}, 0)$ **(F1)**
    else if $A^- \cup N_b(v)$ *contains a red edge* **then return** $(\emptyset, \mathbf{c}, 0)$
    foreach $u \in N_b(v)$ **do**
      $A^+ \leftarrow A^+ \cup N_r(u)$, $c \leftarrow c \cdot \mathbf{c_0}(u)$, $G \leftarrow G - u$, $A^- \leftarrow A^- - u$
    $c \leftarrow c \cdot \mathbf{c_1}(v)$, $G \leftarrow G - v$, $A^+ \leftarrow A^+ - v$
  else if *there exists a vertex* $v \in A^-$ **then**
    if v *has a red neighbor in* A^- **then return** $(\emptyset, \mathbf{c}, 0)$ **(F2)**
    if v *has a red neighbor* **then** $A^+ \leftarrow A^+ \cup N_r(v)$
    if $N_b(v) \subset A^-$ **then**
      $c \leftarrow c \cdot \mathbf{c_{\bar{0}}}(v)$, $G \leftarrow G - v$, $A^- \leftarrow A^- - v$ **(F3)**
    else if $|N_b(v) - A^-| = 1$ **then**
      Let $\{u\} = N_b(v) - A^-$ **(F4)**
      $\mathbf{c_1}(u) \leftarrow \mathbf{c_1}(u)\mathbf{c_0}(v)$, $\mathbf{c_0}(u) \leftarrow \mathbf{c_0}(u)\mathbf{c_{\bar{0}}}(v)$, $\mathbf{c_{\bar{0}}}(u) \leftarrow \mathbf{c_{\bar{0}}}(u)\mathbf{c_{\bar{0}}}(v)$, $G \leftarrow G - v$, $A^- \leftarrow A^- - v$
    else if $|N_b(v) - A^-| = 2$ **then**
      Let $\{u, w\} = N_b(v) - A^-$ **(F5)**
      $E_r \leftarrow E_r \cup \{uw\}$
      $c \leftarrow c \cdot \mathbf{c_0}(v)$, $G \leftarrow G - v$, $A^- \leftarrow A^- - v$
  else if *there exists an isolated vertex* v **then**
    $c \leftarrow c(\mathbf{c_1}(v) + \mathbf{c_{\bar{0}}}(v))$, $G \leftarrow G - v$ **(F6)**
  else if *there exists a leaf* v **then**
    Let u be the neighbor of v **(F7)**
    if uv *is both blue and red* **then**
      $\mathbf{c_1}(u) \leftarrow \mathbf{c_1}(u)\mathbf{c_0}(v)$, $\mathbf{c_0}(u) \leftarrow \mathbf{c_0}(u)\mathbf{c_1}(v)$
    else if uv *is red* **then**
      $\mathbf{c_1}(u) \leftarrow \mathbf{c_1}(u)(\mathbf{c_1}(v) + \mathbf{c_{\bar{0}}}(v))$, $\mathbf{c_0}(u) \leftarrow \mathbf{c_0}(u)\mathbf{c_1}(v)$, $\mathbf{c_{\bar{0}}}(u) \leftarrow \mathbf{c_{\bar{0}}}(u)\mathbf{c_1}(v)$
    else if uv *is blue* **then**
      $\mathbf{c_1}(u) \leftarrow \mathbf{c_1}(u)\mathbf{c_0}(v)$, $\mathbf{c_0}(u) \leftarrow \mathbf{c_0}(u)(\mathbf{c_1}(v) + \mathbf{c_{\bar{0}}}(v))$, $\mathbf{c_{\bar{0}}}(u) \leftarrow \mathbf{c_0}(u)\mathbf{c_1}(v) + \mathbf{c_{\bar{0}}}(u)\mathbf{c_{\bar{0}}}(v)$
    $G \leftarrow G - v$
return $(G, \mathbf{c}, c)$

The following statement can be shown by induction using some definitions and simple calculations (see [11] for a detailed proof.)

Lemma 1. *Let $G = (V, E_b, E_r)$ be a br-graph, and $\mathbf{c}$ a cardinality function satisfying the condition (A1). Let $(G', \mathbf{c}', c) = \mathrm{Prop}(G, \mathbf{c}, A^+, A^-)$.*
Then $\#IS(G, \mathbf{c}, A^+, A^-) = c \cdot \#IS(G', \mathbf{c}')$.

Lemma 2. *The procedure* Prop *runs in polynomial time.* □

Reduction. The procedure Reduction can be applied if the br-graph$G$ has a cut vertex. It takes as input a br-graph$G$, a cardinality function $\mathbf{c}$, a cut vertex v and a component H of $G - v$ with the smallest number of vertices of degree 3. If there is a cut vertex in G then we can choose a vertex v such that $d(v) = 3$ and $d_{G-H}(v) = 2$, so we assume that these conditions hold. Reduction returns a br-graph$G - H$ and a cardinality function $\mathbf{c}'$ such that $\#IS(G, c) = \#IS(G - H, \mathbf{c}')$.

Algorithm 2. Reduction$(G, \mathbf{c}, v, H)$

$G_1 \leftarrow G[V(H) \cup \{v\}]$, $G_2 \leftarrow G - H$, $u \leftarrow$unique neighbor of v in G_1
$(\tilde{G}_1, \tilde{\mathbf{c}}, p) \leftarrow \mathrm{Prop}(G_1, \mathbf{c}, \{v\}, \emptyset)$
$t \leftarrow \mathrm{MISCount}(\tilde{G}_1, \tilde{\mathbf{c}})$
$\mathbf{c_1}(v) \leftarrow p \cdot t$
if *vu is red* **then**
  $(G_1', \mathbf{c}', p) \leftarrow \mathrm{Prop}((G_1 - v), \mathbf{c}, \{u\}, \emptyset)$
  $t \leftarrow \mathrm{MISCount}(G_1', \mathbf{c}')$
  $\mathbf{c_0}(v) \leftarrow p \cdot t \cdot \mathbf{c_0}(v)$, $\mathbf{c_{\bar{0}}}(v) \leftarrow p \cdot t \cdot \mathbf{c_{\bar{0}}}(v)$
else if *vu is blue* **then**
  $(G_1', \mathbf{c}', p_1) \leftarrow \mathrm{Prop}((G_1 - v), , \mathbf{c}, \{u\}, \emptyset)$
  $t_1 \leftarrow \mathrm{MISCount}(G_1', \mathbf{c}')$
  $(G_1'', \mathbf{c}'', p_0) \leftarrow \mathrm{Prop}((G_1 - v), \mathbf{c}, \emptyset, \{u\})$
  $t_0 \leftarrow \mathrm{MISCount}(G_1'', \mathbf{c}'')$
  $\mathbf{c_0}(v) \leftarrow p_1 \cdot t_1 \cdot \mathbf{c_0}(v) + p_0 \cdot t_0 \cdot \mathbf{c_0}(v)$, $\mathbf{c_{\bar{0}}}(v) \leftarrow p_1 \cdot t_1 \cdot \mathbf{c_0}(v) + p_0 \cdot t_0 \cdot \mathbf{c_{\bar{0}}}(v)$
return $\mathrm{MISCount}(G_2, \mathbf{c})$

Branching. We call a pair $A = (A^+, A^-)$ of disjoint subsets of V such that $A^+ \cup A^- \neq \emptyset$ a *branch condition*. A set $\mathcal{A} = \{(A_1^+, A_1^-), (A_2^+, A_2^-), \ldots, (A_k^+, A_k^-)\}$ of branch conditions is called *complete* for $(G, \mathbf{c})$ if the sets $IS(G, A_1^+, A_1^-), IS(G, A_2^+, A_2^-), \ldots, IS(G, A_k^+, A_k^-)$ are pairwise disjoint and cover the set $\{S \in IS(G) : C(S, \mathbf{c}) > 0\}$. By G_A we denote a graph returned by $\mathrm{Prop}(G, \mathbf{c}, A^+, A^-)$.

For a br-graph$G$, a cardinality function $\mathbf{c}$ and a complete set of branch conditions $\{(A_1^+, A_1^-), (A_2^+, A_2^-), \ldots, (A_k^+, A_k^-)\}$ the procedure Branching separately computes the numbers $\#IS(G, \mathbf{c}, A_i^+, A_i^-)$ and returns the sum of them.

Algorithm 3. Branching$(G, \mathbf{c}, \{(A_1^+, A_1^-), (A_2^+, A_2^-), \ldots, (A_k^+, A_k^-)\})$

foreach $i \in \{1, \ldots, k\}$ **do**
  $(G_i, \mathbf{c^i}, p_i) \leftarrow \mathrm{Prop}(G, \mathbf{c}, A_i^+, A_i^-)$
  $t_i \leftarrow \mathrm{MISCount}(G_i, \mathbf{c^i})$
return $p_1 \cdot t_1 + \ldots + p_k \cdot t_k$

Algorithm 4. MISCount$(G, \mathbf{c})$

$(G, \mathbf{c}, c) \leftarrow \text{Prop}(G, \mathbf{c}, \emptyset, \emptyset)$ **(P)**
if *G is empty* **then return** 1
if *G is disconnected* **then**
  return $c \cdot \prod_{i=1}^{s} c_i$, where $c_i \leftarrow \text{MISCount}(C_i, \mathbf{c})$, for the components $C_1, \ldots, C_s$ of G **(R0)**
if *there exists a cut vertex in G* **then**
  Let v be a cut vertex and H a component of $G - v$ such that $d_{G-H}(v) = 2$ and $n(H) \leq n(G - H)$ **(R1)**
  return $c \cdot \text{Reduction}(G, \mathbf{c}, v, H)$
if $\Delta(G) = 2$ **then**
  return $c \cdot \text{Branching}(G, \mathbf{c}, \{(\{v\}, \emptyset), (\{x\}, \emptyset), (\emptyset, \{v, x\}), (\{v, x\}, \emptyset)\})$, where vx is any edge in G **(C)**
if *there exists a 2-element cut set $\{a,b\}$ and a component L of $G - \{a, b\}$ such that $1 \leq n_3(L) \leq 12$ and $d_{G-L}(a) = d_{G-L}(b) = 2$* **then**
  Let d be the unique neighbor in $V(L)$ of a **(B+R)**
  if *ad is red* **then return** $c \cdot \text{Branching}(G, \mathbf{c}, \{(\{a\}, \emptyset), (\emptyset, \{a\})\})$
  if *ad is blue* **then return** $c \cdot \text{Branching}(G, \mathbf{c}, \{(\{a\}, \emptyset), (\{d\}, \emptyset), (\emptyset, \{a, d\})\})$
if *there exists a vertex v of degree 3 incident to two red edges* **then**
  return $c \cdot \text{Branching}(G, \mathbf{c}, \{(\{v\}, \emptyset), (\emptyset, \{v\})\})$ **(B)**
if *there exists a vertex v of degree 3 incident to a red edge* **then**
  if $SS(v) = (9, 8, 8, 8)$ **then**
    Let x and y be blue neighbors of v **(B)**
    return $c \cdot \text{Branching}(G, \mathbf{c}, (\{v\}, \emptyset), (\{x\}, \{v\}), (\{y\}, \{v, x\})\})$
  else return $c \cdot \text{Branching}(G, \mathbf{c}, \{(\{v\}, \emptyset), (\emptyset, \{v\})\})$ **(B)**
if *there exists a triangle xyz in G* **then**
  return $c \cdot \text{Branching}(G, \mathbf{c}, \{(\{x\}, \emptyset), (\{y\}, \emptyset), (\{z\}, \emptyset), (N(\{x, y, z\}) - \{x, y, z\}, \{x, y, z\})\})$ **(B)**
Let $S = \max\{S(v) : v \in V\}$
if $9 \leq S \leq 10$ **then**
  Let v be a vertex such that $SS(v)$ is maximal and let x, y, z be neighbors of v and $d(y) = d(z) = 2$ **(B)**
  return $c \cdot \text{Branching}(G, \mathbf{c}, \{(\{v\}, \emptyset), (\{z\}, \{v\}), (\{y\}, \{v, z\}), (\{x\}, \{v, z, y\})\})$
else
  if $S = 11$ *and there exists a path $v_1v_2v_3v_4$ such that* $S(v_1) = S(v_2) = S(v_3) = S(v_4) = 11$ **then**
    if *v_1 and v_3 are topological neighbors* **then** $v \leftarrow v_3$, $x \leftarrow v_4$
    else if *v_2 and v_4 are topological neighbors* **then** $v \leftarrow v_2$, $x \leftarrow v_1$
    else $v \leftarrow v_2$, $x \leftarrow v_3$
  else if $S = 12$ *and there exist an edge v_1v_2 such that* $SS(v_1) \geq_L S(v_2) \geq_L (12, 12, 12, 10)$ **then**
    $v \leftarrow v_1$, $x \leftarrow v_2$
  else
    Let v be a vertex such that $SS(v)$ is maximal and let $\{v_1, v_2, v_3\} = N(v)$ and $S(v_1) \geq S(v_2) \geq S(v_3)$
    if $S(v_1) = S(v_3)$, *and two of v_1, v_2, v_3 are topological neighbors, say v_1 and v_2* **then** $x \leftarrow v_1$
    if $S(v_1) = S(v_2)$, *and v_1 and v_3 are topological neighbors* **then** $x \leftarrow v_2$
    else $x \leftarrow v_1$
  return $c \cdot \text{Branching}(G, \mathbf{c}, \{(\{v\}, \emptyset), (\{x\}, \emptyset), (\emptyset, \{v, x\})\})$ **(B)**

Main Algorithm

Theorem 3. *Let $G = (V, E_b, E_r)$ be a br-graphand* $\mathbf{c}$ *a cardinality function satisfying the conditions (A1), (A2), (A3). Then* $\mathrm{MISCount}(G, \mathbf{c}) = \#IS(G, \mathbf{c})$.

Proof. The proof is by induction on $n(G)$. If $n(G) \leq 1$, then the assertion clearly holds. Assume that the assertion holds for all graphs with at most n vertices and let G be a graph with $n+1$ vertices. The br-graph$G$ is processed in one of the following steps: **(P)**, **(R0)**, **(R1)**, **(C)**, **(B)**, or **(B+R)**.
(P) By Lemma 1 and the induction hypothesis $\mathrm{MISCount}(G, \mathbf{c}) = \#IS(G, \mathbf{c})$.
(R0) Observe that if C is a component of G, then $\#IS(G, \mathbf{c}) = \#IS(C, \mathbf{c}) \cdot \#IS(G - C, \mathbf{c})$. From this fact and the induction hypothesis $\mathrm{MISCount}(G, \mathbf{c}) = \#IS(G, \mathbf{c})$.
(R1) Notice that when a red edge incident to a vertex u is added (step **(F5)** of Prop), then in the same step **(F5)** a blue edge incident with v is removed. Hence in any br-graphG' appearing in the course of the algorithm the sum of the number of blue edges incident to a vertex v and the number of red edges incident to v is at most 3. In other words the ends of an edge that is both blue and red are of degree at most 2. Since $d(v) = 3$, the only edge incident with v and the other end in $V(H)$ is either blue or red.

Let $\mathbf{c}^b$ and $\mathbf{c}^a$ be the values of $\mathbf{c}$ before and after applying Reduction.

$$\#IS(G, \mathbf{c}^b) = \sum_{S \in IS(G)} C(S, \mathbf{c}^b) = \sum_{S \in IS(G): v \in S} C(S, \mathbf{c}^b) + \sum_{S \in IS(G): v \notin S} C(S, \mathbf{c}^b)$$

We will deal with these summands separately.

If $S \in IS(G)$ and $v \in S$ then S is a union of $S_1 \in IS(G_1)$ and $S_2 \in IS(G_2)$ such that $S_1 \cap S_2 = \{v\}$. Hence

$$\sum_{S \in IS(G): v \in S} C(S, \mathbf{c}^b) = \sum_{\substack{S_2 \in IS(G_2): \\ v \in S_2}} \sum_{\substack{S_1 \in IS(G_1): \\ v \in S_1}} C(S_2/\{v\}, \mathbf{c}^b) C(S_1/\{v\}, \mathbf{c}^b) \mathbf{c}^b_{\mathbf{1}}(v) =$$

$$\sum_{S_2 \in IS(G_2): v \in S_2} C(S_2/\{v\}, \mathbf{c}^b) \Big(\sum_{S_1 \in IS(G_1): v \in S_1} C(S_1/\{v\}, \mathbf{c}^b) \mathbf{c}^b_{\mathbf{1}}(v) \Big) =$$

$$\sum_{S_2 \in IS(G_2): v \in S_2} C(S_2/\{v\}, \mathbf{c}^b) \Big(\sum_{S_1 \in IS(G_1): v \in S_1} C(S_1, \mathbf{c}^b) \Big) = \sum_{S_2 \in IS(G_2): v \in S_2} C(S_2/\{v\}, \mathbf{c}^a) \mathbf{c}^a_{\mathbf{1}}(v) = \sum_{S \in IS(G_2): v \in S} C(S, \mathbf{c}^a).$$

In a similar way it can be shown (see [11] for details) that

$$\sum_{S\in IS(G):v\notin S} C(S,\mathbf{c}^b) = \sum_{S\in IS(G_2):v\notin S} C(S,\mathbf{c}^a).$$

Thus

$$\#IS(G,\mathbf{c}^b) = \sum_{S\in IS(G):v\in S} C(S,\mathbf{c}^b) + \sum_{S\in IS(G):v\notin S} C(S,\mathbf{c}^b) =$$
$$\sum_{S\in IS(G_2):v\in S} C(S,\mathbf{c}^a) + \sum_{S\in IS(G_2):v\notin S} C(S,\mathbf{c}^a) = \#IS(G_2,\mathbf{c}^a).$$

(C),**(B)** Observe that in each case of applying the Branching procedure we branch on a complete set of branch conditions. From this fact, Lemma 1 and the induction hypothesis we get $\mathrm{MISCount}(G,\mathbf{c}) = \#IS(G,\mathbf{c})$. Notice that for any branch condition used in MISCount the conditions (A1)-(A3) are satisfied.
(B+R) The proof follows from the previous cases **(B)**, **(R1)**, and **(R0)**.

4 Complexity

For positive real numbers $t_0,\ldots,t_d$ we denote by $\tau(t_0,\ldots,t_d)$ the unique solution $\tau > 1$ of the equation $\sum_{i=0}^{d}\tau^{-t_i} = 1$. One can readily verify that

$$\text{if } t_i \le t'_i \text{ for } i\in\{0,\ldots,d\}, \text{ then } \tau(t'_0,\ldots,t'_d) \le \tau(t_0,\ldots,t_d). \tag{1}$$

In the complexity analysis we use a measure μ, which is a function assigning a nonnegative real number to every graph. Consider an arbitrary graph G' which labels some internal vertex of the tree of recursive calls of MISCount. Let the children of G' be vertices labeled with the subgraphs $G_0,\ldots,G_d$ of G' for which our algorithm is next called. Assume that $\Delta_i\mu(G') = \mu(G') - \mu(G_i) > 0$ for $i = 0,\ldots,d$. Then the number $\tau(\Delta_0\mu(G'),\ldots,\Delta_d\mu(G'))$ is well defined and we call this number the *branching number* for G' (with respect to the measure μ). Kullmann [12] proved that if this assumption is satisfied for all internal vertices of the tree of recursive calls of the algorithm for a graph G, then the number of leaves of this tree is bounded by $O(\tau_0^{\mu(G)})$, where τ_0 is the largest branching number for the internal vertices of the tree. In our analysis it is convenient to consider a subtree of the tree of recursive calls of our algorithm whose internal vertices are restricted to vertices labeled with graphs whose measure is larger than some constant, say c. The leaves of this tree are labeled with graphs whose measure is at most c. We show that our algorithm applied to each graph which labels a leaf works in polynomial time. Moreover, in our algorithm, the height of the tree of recursive calls is bounded by the number of vertices of the instance graph. Therefore the number of internal vertices of the tree is bounded by a linear function of the number of leaves. We also show that the amount of time between one recursive call of our algorithm and the next is polynomial with respect to the order of the graph. Hence the running time of the considered algorithm applied to a graph G is bounded by $O^*(\tau_0^{\mu(G)})$.

The following lemma guarantees desired configuration for the branching depending on graph density.

Lemma 4. *Let G be a 2-connected triangle-free graph.*

1. *If $\frac{2m}{n} > \frac{9}{4}$ then there exists a vertex v such that $SS(v) \geq_L (9,8,7,7)$ or $S(v) \geq 10$ (there exists a vertex of degree 3 with another vertex of degree 3 within a distance at most 2).*
2. *If $\frac{2m}{n} > \frac{16}{7}$ then there exists a vertex v such that $SS(v) \geq_L (9,8,8,7)$ or $S(v) \geq 10$ (there exists a vertex of degree 3 with another two vertices of degree 3 within a distance at most 2).*
3. *If $\frac{2m}{n} > \frac{7}{3}$ then there exists a vertex v such that $SS(v) = (9,8,8,8)$ or $SS(v) \geq_L (10,10,8,7)$ or $S(v) \geq 11$ (there exists a vertex of degree 3 with another three vertices of degree 3 within a distance at most 2 or a vertex of degree 3 with a neighbor of degree 3 and another vertex of degree 3 within a distance at most 2).*
4. *If $\frac{2m}{n} > \frac{28}{11}$ then there exist a vertex v such that $SS(v) \geq_L (11,11,10,7)$ or $S(v) = 12$ (there exist two adjacent vertices of degree 3 both having at least two neighbors of degree 3 or there is a vertex of degree 3 with three neighbors of degree 3).*
5. *If $\frac{2m}{n} > \frac{8}{3}$ then there exists a vertex v such that $S(v) = 12$ (there exists a vertex of degree 3 with all neighbors of degree 3).*
6. *If $\frac{2m}{n} > \frac{14}{5}$ then there exist a vertex v such that $SS(v) \geq_L (12,12,12,10)$ (there exists a vertex of degree 3 with all neighbors of degree 3 and such that at least two of its neighbors have all neighbors of degree 3).*
7. *If $\frac{2m}{n} > \frac{48}{17}$ then there exists a vertex v such that $SS(v) \geq_L (12,12,12,10)$ with a neighbor x such that $SS(x) \geq_L (12,12,12,10)$ (there exist two adjacent vertices of degree 3 both having all neighbors of degree 3 and at least two neighbors with all neighbors of degree 3).*

Proof. We observe that $\frac{2m}{n} = \frac{3n_3+2n_2}{n_3+n_2} = \frac{3\frac{n_3}{n_2}+2}{\frac{n_3}{n_2}+1} = 3 - \frac{1}{\frac{n_3}{n_2}+1}$ is an increasing function of $\frac{n_3}{n_2}$. Clearly, $S(v) \geq 9$ for every neighbor of degree 3. We prove here the case 4. The proves of the other cases are similar and can be found in [11].

4. Suppose that for all vertices $v \in V$, $SS(v) \leq_L (11,10,10,7)$. Then there is no path consisting only of vertices of degree 3 containing more than 3 vertices. Let a_1, a_2 and a_3 be the number of all maximal paths consisting only of vertices of degree 3 containing exactly 1, 2 and 3 vertices, respectively. Then $n_3 = a_1+2a_2+3a_3$ and $n_2 = \sum_{v:d(v)=3} n_2(v) \geq \frac{3}{2}a_1 + 2a_2 + \frac{5}{2}a_3$. We have $\frac{n_2}{n_3} \geq \frac{\frac{3}{2}a_1+2a_2+\frac{5}{2}a_3}{a_1+2a_2+3a_3} \geq \frac{\frac{5}{6}a_1+\frac{5}{3}a_2+\frac{5}{2}a_3}{a_1+2a_2+3a_3} = \frac{5}{6}$. Finally we get $\frac{2m}{n} = 3 - \frac{1}{\frac{n_3}{n_2}+1} \leq 3 - \frac{1}{\frac{6}{5}+1} = \frac{28}{11}$.

It can be shown that the asymptotic behavior of the running time of the algorithm MISCount is determined by the calls of the procedure Branching. As in [3], we define a measure of a connected graph G which depends on $n(G)$ and $m(G)$ only, i.e $\mu(G) = \mu'(n(G), m(G))$. For a disconnected graph G let $\mu(G) = \sum_{C:C \text{ is a component of G}} \mu(C)$.

The function $\mu'(n,m)$ is a piecewise linear function defined as follows. We partition the interval $(0,3]$ into subintervals $(k_i, k_{i+1}]$ for $i = 0,\ldots,8$ using as k_is

the density values appearing in Lemma 4 (see Table 1 for the values of the k_is). We define $\mu'(n,m) = \mu_i(n,m) = a_i n + b_i m$, if $\frac{2m}{n} \in (k_i; k_{i+1}]$ and $\mu'(n,0) = 0$. We observe that $\frac{2m(G)}{n(G)} \leq 3$, for graphs G whose vertices have degrees at most 3, so the measure μ has been defined for all such graphs. The coefficients a_i, b_i (whose approximate values are given in Table 1) are chosen in such a way that $\mu_1(n(G), m(G)) = n_3(G)$ for graphs with $\delta(G) \geq 2$, the function μ' is *continuous* (i.e. $\mu_{i-1}(n,m) = \mu_i(n,m)$ when $\frac{2m}{n} = k_i$, for $i = 1, \ldots, 8$) and optimal in the sense that the largest of the branching numbers of graphs computed using these values of a_i and b_i is as small as possible. The optimality is achieved when the largest branching numbers are equal in every density interval $(k_i; k_{i+1}]$.

For convenience we introduce some auxiliary numbers χ_i, for $i = 0, \ldots, 8$. The approximate values of a_i, b_i, k_i, χ_i are given in the table. We assume that

i	k_i	k_{i+1}	a_i	b_i	χ_i	$O^*(\tau_0^{\chi_i n})$
0	0	2	0	0	0	$O^*(1)$
1	2	2.25	-2	2	0.25	$O^*(1.090508^n)$
2	2.25	$2\frac{2}{7}$	-1.667718	1.704638	0.28044	$O^*(1.102073^n)$
3	$2\frac{2}{7}$	$2\frac{1}{3}$	-1.37003	1.444162	0.314826	$O^*(1.115285^n)$
4	$2\frac{1}{3}$	$2\frac{6}{11}$	-1.21159	1.308356	0.45359	$O^*(1.170232^n)$
5	$2\frac{6}{11}$	$2\frac{2}{3}$	-0.947936	1.1012	0.52033	$O^*(1.197616^n)$
6	$2\frac{2}{3}$	2.8	-0.750862	0.953394	0.58389	$O^*(1.22429^n)$
7	2.8	$2\frac{14}{17}$	-0.58457	0.834614	0.593708	$O^*(1.228463^n)$
8	$2\frac{14}{17}$	3	-0.465968	0.750604	0.659938	$O^*(1.256986^n)$

the numbers a_i, b_i, χ_i satisfy the following conditions:

$$a_0 = b_0 = \chi_0 = 0 \tag{2}$$

$$\chi_i = \chi_{i-1} + \frac{b_i}{2}(k_{i+1} - k_i), \text{ for } i = 1, \ldots, 8, \tag{3}$$

$$a_i = \chi_{i-1} - \frac{k_i b_i}{2}, \text{ for } i = 1, \ldots, 8, \tag{4}$$

$$\mu_i(n,m) = a_i n + b_i m = \chi_{i-1} n + (m - \frac{k_i n}{2}) b_i, \text{ for } i = 1, \ldots, 8. \tag{5}$$

Using (2)-(5) and the fact that $b_1 \geq b_2 \geq \ldots \geq b_7$ one can easily show that the function $\mu'(n,m)$ has the following properties:

(**P1**) $\mu'(n,m)$ is continuous.

(**P2**) $\mu'(n,m)$ is concave, i.e. $\mu'(n,m) \leq \mu_i(n,m)$, for $m \geq n$ and $i = 1 \ldots, 8$.

(**P3**) If $\frac{2m}{n} \in (k_i; k_{i+1}]$, then $0 \leq \mu_i(n,m) \leq \chi_i n$, for $i = 0 \ldots, 8$.

By an exhaustive case analysis we prove that the largest branching number defined for the algorithm MISCount and the measure μ defined in this section is $\tau_0 = \tau(4,4,4,4) = \sqrt{2}$ (see [11] for details).

Let C be a component of a graph G and let $j = 0, \ldots, 8$ be such that $\frac{2m(C)}{n(C)} \in (k_j, k_{j+1}]$. Then, by properties (**P2**) and (**P3**), $\mu(C) = \mu'(n(C), m(C))$

$\leq \mu_j(n(C), m(C)) \leq \chi_j n(C) \leq \chi_8 n(C)$ so, consequently, $\mu(G) \leq \chi_8 n(G)$. By a remark at start of this section, the running time of the algorithm MISCount is bounded by $O^*(\tau_0^{\mu(G)}) = O^*(\sqrt{2}^{\chi_8 n(G)})$. This way we obtain the following result.

Theorem 5. *The algorithm* MISCount *runs in time* $O^*(1.256986^n)$.

It is easy to observe that our algorithm requires a polynomial amount of space.

References

1. Alber, J., Niedermeier, R.: Improved Tree Decomposition Based Algorithms for Domination-like Problems. In: Rajsbaum, S. (ed.) LATIN 2002. LNCS, vol. 2286, pp. 613–628. Springer, Heidelberg (2002)
2. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion-exclusion. SIAM J. Comput. 39(2), 546–563 (2009)
3. Dahllöf, V., Jonsson, P., Wahlström, M.: Couning models for 2SAT and 3SAT formulae. Theor. Comput. Sci. 332, 265–291 (2005)
4. Fomin, F.V., Kratsch, D.: Exact Exponential Algorithms. Springer, Heidelberg (2010)
5. Fomin, F.V., Hoie, K.: Pathwidth of cubic graphs and exact algorithms. Inf. Process. Lett. 97(5), 191–196 (2006)
6. Fomin, F.V., Gaspers, S., Saurabh, S., Stepanov, A.A.: On Two Techniques of Combining Branching and Treewidth. Algorithmica 54(2), 181–207 (2009)
7. Fürer, M., Kasiviswanathan, S.P.: Algorithms for Counting 2-Sat Solutions and Colorings with Applications. In: Kao, M.-Y., Li, X.-Y. (eds.) AAIM 2007. LNCS, vol. 4508, pp. 47–57. Springer, Heidelberg (2007)
8. Gaspers, S., Kratsch, D., Liedloff, M.: On Independent Sets and Bicliques in Graphs. In: Broersma, H., Erlebach, T., Friedetzky, T., Paulusma, D. (eds.) WG 2008. LNCS, vol. 5344, pp. 171–182. Springer, Heidelberg (2008)
9. Greenhill, C.: The complexity of counting colourings and independent sets in sparse graphs and hypergraphs. Comput. Complex 9, 52–73 (2000)
10. Jou, M.J., Chang, G.J.: Algorithmic aspects of counting independent sets. Ars. Comb. 65, 265–277 (2002)
11. Junosza-Szaniawski, K., Tuczyński, M.: Counting maximal independent sets in subcubic graphs, Tech Rep., www.mini.pw.edu.pl/~szaniaws
12. Kullmann, O.: New methods for 3-SAT decision and worst-case analysis. Theor. Comput. Sci. 223(1-2), 1–72 (1999)
13. Kutzkov, K.: New upper bound for the #3-SAT problem. Inform. Process. Lett. 105, 1–5 (2007)
14. Lonc, Z., Truszczynski, M.: Computing minimal models, stable models and answer sets. Theory and Practice of Logic Prog. 6(4), 395–449 (2006)
15. Vadhan, S.P.: The Complexity of Counting in Sparse, Regular, and Planar Graphs. SIAM J. on Comput. 31, 398–427 (1997)
16. Wahlström, M.: A Tighter Bound for Counting Max-Weight Solutions to 2SAT Instances. In: Grohe, M., Niedermeier, R. (eds.) IWPEC 2008. LNCS, vol. 5018, pp. 202–213. Springer, Heidelberg (2008)

Iterated Hairpin Completions of Non-crossing Words*

Lila Kari[1], Steffen Kopecki[1,2], and Shinnosuke Seki[3]

[1] Department of Computer Science
University of Western Ontario, London
lila@csd.uwo.ca
[2] Institute for Formal Methods in Computer Science
University of Stuttgart
kopecki@fmi.uni-stuttgart.de
[3] Department of Systems Biosciences for Drug Discovery
Graduate School of Pharmaceutical Sciences, Kyoto University
sseki@pharm.kyoto-u.ac.jp

Abstract. Iterated hairpin completion is an operation on formal languages that is inspired by the hairpin formation in DNA biochemistry. Iterated hairpin completion of a word (or more precisely a singleton language) is always a context-sensitive language and for some words it is known to be non-context-free. However, it is unknown whether regularity of iterated hairpin completion of a given word is decidable. Also the question whether iterated hairpin completion of a word can be context-free but not regular was asked in literature. In this paper we investigate iterated hairpin completions of non-crossing words and, within this setting, we are able to answer both questions. For non-crossing words we prove that the regularity of iterated hairpin completions is decidable and that if iterated hairpin completion of a non-crossing word is not regular, then it is not context-free either.

1 Introduction

On an abstract level, a DNA single strand can be viewed as a word over the four-letter alphabet $\{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$ where the letters represent the nucleobases adenine, cytosine, guanine, and thymine, respectively. The *Watson-Crick complement* of A is T and the complement of C is G. Two complementary single strands of opposite orientation can bond to each other and form a DNA double strand. Throughout the paper, we use the bar-notation for complementary strands of opposite orientation.

In the same manner, a single strand can bond to itself if two of its substrands are complementary and do not overlap with each other. Such an intramolecular

* This research was supported by the Natural Sciences and Engineering Research Council of Canada Discovery Grant R2824A01 and Canada Research Chair Award to L. K., and by the Funding Program for Next Generation World-Leading Researchers (NEXT Program) to Yasushi Okuno, the current supervisor of S. S.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 337–348, 2012.

base pairing is called a *hairpin*. We are especially interested in hairpins of single strands of the form $\sigma = \gamma\alpha\beta\overline{\alpha}$. Here, the substrand $\overline{\alpha}$ can bond to the substrand α. Then, by extension, a new single strand can be synthesized which we call a *hairpin completion* of σ, see Figure 1. In this situation we call the substrands that initiate the hairpin completion, α and $\overline{\alpha}$, *primers*.

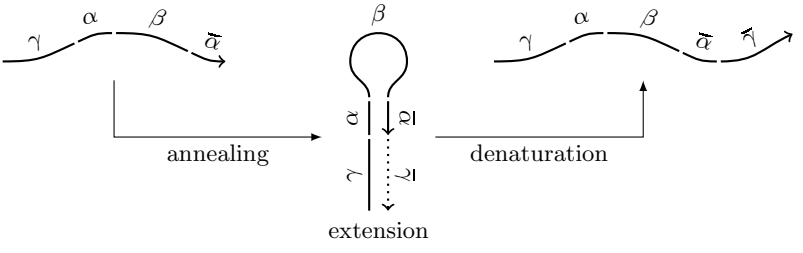


Fig. 1. Hairpin completion of a DNA single strand

In DNA computing hairpins and hairpin completions are often undesired by-products. Therefore, sets of strands have been analyzed and designed that are unlikely to form hairpins or lead to other *undesireable hybridization*, see [1, 2, 7, 8, 10, 19] and the references within.

However, there are DNA computational models that rely on hairpins, e. g., DNA RAM [9, 21, 22] and Whiplash PCR [5, 20, 23]. For the Whiplash PCR consider a single strand just like in Figure 1, but where the length of extension is controlled by *stopper sequences*. Repeating this operation, DNA can be used to solve combinatorial problems like the Hamiltonian path problem.

Inspired by hairpins in biocomputing, the hairpin completion of a formal language has been introduced by Cheptea, Martín-Vide, and Mitrana in [3]. In several papers hairpin completion and its iterated variant have been investigated, see [4, 13–18]. In this paper we consider iterated hairpin completions of singletons, that is, informally speaking, iterated hairpin completions of words. The class of iterated hairpin completions of singletons is denoted by HCS. It is known that every language in HCS is decidable in NL (non-deterministic, logarithmic space) as NL is closed under iterated hairpin completion [3]; hence, HCS is a proper subclass of the context-sensitive languages. It is also known that HCS contains regular as well as non-context-free languages [13]. In the latter paper, two open problems have been stated:

1. Is it decidable whether the iterated hairpin completion of a singleton is regular?
2. Does a singleton exist whose iterated hairpin completion is context-free but not regular?

We solve both questions for non-crossing words (or rather, singletons containing a non-crossing word). A word w is said to be non-crossing if, for a given primer α,

the right-most occurrence of the factor α in w precedes the left-most occurrence of the factor $\overline{\alpha}$ in w, see Section 3. We provide a necessary and sufficient condition for regularity of iterated hairpin completion of a given non-crossing word (Theorem 2 and Corollary 4) and, since this condition is decidable, we answer the first question positively (Corollary 5). Furthermore, we show that iterated hairpin completion of a non-crossing word is either regular or it is not context-free (Corollary 6). Thus, we give a negative answer to the second question.

This paper is the continuation of the studies in [11]. Due to the page limitation some proofs have been omitted. The missing proofs can be found in the arXiv version [12].

2 Preliminaries

We assume the reader to be familiar with the fundamental concepts of language theory, see [6].

Let Σ be an alphabet, Σ^* be the set of all words over Σ, and for an integer $k \geq 0$, Σ^k be the set of all words of length k over Σ. The word of length 0 is called the empty word, denoted by ε, and we let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. A subset of Σ^* is called a language over Σ. For a word $w \in \Sigma^*$, we employ the notation w when we mean the word as well as the singleton language $\{w\}$ unless confusion arises.

We equip Σ with a function $\overline{\ }\colon \Sigma \to \Sigma$ satisfying $\forall a \in \Sigma, \overline{\overline{a}} = a$; such a function is called an *involution*. This involution $\overline{\ }$ is naturally extended to words as: for $a_1, \ldots, a_n \in \Sigma$, $\overline{a_1 a_2 \cdots a_n} = \overline{a_n} \cdots \overline{a_2}\,\overline{a_1}$. For a word $w \in \Sigma^*$, we call $\overline{w}$ the *complement of* w, being inspired by this application. A word $w \in \Sigma^*$ is called a *pseudo-palindrome* if $w = \overline{w}$. For a language $L \subseteq \Sigma^*$, we let $\overline{L} = \{\overline{w} \mid w \in L\}$.

For words $u, w \in \Sigma^*$, if $w = xuy$ holds for some words $x, y \in \Sigma^*$, then u is called a *factor* of w; a factor that is distinct from w is said to be *proper*. If the equation holds with $x = \varepsilon$ ($y = \varepsilon$), then the factor u is especially called a *prefix* (resp. a *suffix*) of w. The prefix relation can be regarded as a partial order $\leq_p$ over Σ^* whereas the proper prefix relation can be regarded as a strict order $<_p$ over Σ^*; $u \leq_p w$ means that u is a prefix of w and $u <_p w$ means that u is a proper prefix of w. Analogously, by $w \geq_s u$ (or $w >_s u$) we mean that u is a suffix (resp. proper suffix) of w. Note that $u \leq_p w$ if and only if $\overline{w} \geq_s \overline{u}$. For a word $w \in \Sigma^*$ and a language $L \subseteq \Sigma^*$, a factor u of w is *minimal with respect to* L if $u \in L$ and none of the proper factors of u is in L.

2.1 Hairpin Completion

Let k be a constant that is assumed to be the length of a primer and let $\alpha \in \Sigma^k$ be a primer. If a given word $w \in \Sigma^*$ can be written as $\gamma\alpha\beta\overline{\alpha}$ for some $\gamma, \beta \in \Sigma^*$, then its *right hairpin completion* (with respect to α) results in the word $\gamma\alpha\beta\overline{\alpha}\overline{\gamma}$. By $w \to_{\mathcal{RH}_\alpha} z$, we mean that z can be obtained from w by right hairpin completion (with respect to α). The *left hairpin completion* is defined analogously as an operation to derive $\gamma\alpha\beta\overline{\alpha}\overline{\gamma}$ from $\alpha\beta\overline{\alpha}\overline{\gamma}$, and the relation $\to_{\mathcal{LH}_\alpha}$ is naturally introduced. We write $w \to_{\mathcal{H}_\alpha} z$ if $w \to_{\mathcal{RH}_\alpha} z$ or $w \to_{\mathcal{LH}_\alpha} z$. By $\to^*_{\mathcal{LH}_\alpha}$, $\to^*_{\mathcal{RH}_\alpha}$,

and $\to^*_{\mathcal{H}_\alpha}$ we denote the reflexive and transitive closure of $\to_{\mathcal{LH}_\alpha}$, $\to_{\mathcal{RH}_\alpha}$, and $\to_{\mathcal{H}_\alpha}$, respectively. Whenever α is clear from the context, we omit the subscript α and write $\to_{\mathcal{RH}}$, $\to_{\mathcal{LH}}$, or $\to_{\mathcal{H}}$, respectively.

For a language $L \subseteq \Sigma^*$, we define the set of words obtained by hairpin completion from L, and the set of words obtained by iterated hairpin completion from L, respectively, as follows:

$$\mathcal{H}_\alpha(L) = \{z \mid \exists w \in L, w \to_{\mathcal{H}_\alpha} z\}, \qquad \mathcal{H}^*_\alpha(L) = \left\{z \;\middle|\; \exists w \in L, w \to^*_{\mathcal{H}_\alpha} z\right\}.$$

In this paper the hairpin completion is always considered with respect to a fixed primer α. However, in other literature the hairpin completion is often considered with respect to the length k of primers instead of a specific primer α and defined as

$$\mathcal{H}_k(L) = \bigcup_{\alpha \in \Sigma^k} \mathcal{H}_\alpha(L), \qquad \mathcal{H}^*_k(L) = \bigcup_{i \geq 0} \mathcal{H}^i_k(L).$$

3 Non-crossing Words and Their Properties

In this section, we describe some structural properties of non-crossing words and their iterated hairpin completions and we introduce the notation of α-prefixes, α-suffixes, and α-indexes.

For a word α, we say that w is *non-α-crossing* if the rightmost occurrence of α precedes the leftmost occurrence of $\overline{\alpha}$ on w (yet these factors may overlap). If α is understood from the context, we simply say that w is *non-crossing*. Otherwise, the word is *α-crossing* or *crossing*. The definition of a word w being non-α-crossing becomes useful in our work only if $w \in \alpha\Sigma^*$ or $w \in \Sigma^*\overline{\alpha}$, and therefore, α and $\overline{\alpha}$ are primers; actually, we will assume both. The main purpose of this paper is to prove a necessary and sufficient condition for the regularity of the iterated hairpin completion $\mathcal{H}^*_\alpha(w)$, where $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ is non-α-crossing.

Note that if $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ and $\alpha = \overline{\alpha}$, then either $w = \alpha$ and $\mathcal{H}^*_\alpha(w) = \{w\}$ or w can be considered crossing. Thus, whenever we consider non-crossing words, we assume that $\alpha \neq \overline{\alpha}$.

Any word obtained from a non-crossing word by hairpin completion is non-crossing. Though being easily confirmed, this closure property forms the foundation of our discussions in this paper.

Proposition 1. *For a non-crossing word $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$, every word in $\mathcal{H}^*_\alpha(w)$ is non-crossing.*

Let us provide another characterization for a word $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ to be non-crossing. With Proposition 1, this characterization will bring a unique factorization of any word z in $\mathcal{H}^*_\alpha(w)$ as $z = xwy$ for some words x, y (Corollary 1).

Proposition 2. *A word $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ is non-crossing if and only if it contains exactly one factor x which is minimal with respect to $\alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$.*

Corollary 1. *Let $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ be non-crossing. The factor w occurs exactly once in every word from $\mathcal{H}^*_\alpha(w)$.*

3.1 α-Prefixes and $\overline{\alpha}$-Suffixes

Let u, v, w be words. We call u an *α-prefix* of w if $u\alpha \leq_p w$. This means, if $\overline{\alpha}$ is a suffix of w, then the suffix can bond to the factor α which directly follows the prefix u (unless they overlap with each other) and $w\overline{u}$ can be obtained from w by right hairpin completion. By $\mathrm{P}_\alpha(w)$, we denote the set of all α-prefixes of w. Note that if $x, y \in \mathrm{P}_\alpha(w)$ and $|x| \leq |y|$, then $x\alpha \leq_p y\alpha$ and $x \in \mathrm{P}_\alpha(y\alpha)$. Symmetrically, we call $\overline{v}$ an *$\overline{\alpha}$-suffix* if $w \geq_s \overline{\alpha}\overline{v}$ and $\mathrm{S}_{\overline{\alpha}}(w)$ is the set of all $\overline{\alpha}$-suffixes of w. If $\alpha \leq_p w$ and $|w| \geq |v| + 2k$, then $w \to_{\mathcal{LH}} vw$. Note that $\mathrm{S}_{\overline{\alpha}}(w) = \overline{\mathrm{P}_\alpha(\overline{w})}$. Therefore, all results we derive for α-prefixes also hold for the complements of $\overline{\alpha}$-suffixes.

When $m = |\mathrm{P}_\alpha(w)|$ and $n = |\mathrm{S}_{\overline{\alpha}}(w)|$ for a word w, then w is called *(m, n)-α-word* (or simply *(m, n)-word*). Throughout the paper, it will be convenient to let $\mathrm{P}_\alpha(w) = \{u_0, \ldots, u_{m-1}\}$ and $\mathrm{S}_{\overline{\alpha}}(w) = \{\overline{v_0}, \ldots, \overline{v_{n-1}}\}$ where the words are ordered such that $u_0 <_p \cdots <_p u_{m-1}$ and $\overline{v_{n-1}} >_s \cdots >_s \overline{v_0}$. Note that $\mathrm{P}_\alpha(u_i\alpha) = \{u_0, \ldots, u_i\}$ for $0 \leq i < m$ and $\overline{\mathrm{S}_{\overline{\alpha}}(\overline{\alpha}\overline{v_j})} = \{v_0, \ldots, v_j\}$ for $0 \leq j < n$.

Let us begin with a basic observation.

Lemma 1. *For a word $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$, the following statements hold:*

1. *For every $x \in \mathrm{P}_\alpha(w) \cup \overline{\mathrm{S}_{\overline{\alpha}}(w)}$, we have $\alpha \leq_p x\alpha$.*
2. *For every $x_1, \ldots, x_\ell \in \mathrm{P}_\alpha(w) \cup \overline{\mathrm{S}_{\overline{\alpha}}(w)}$, we have $\alpha \leq_p x_\ell \cdots x_1\alpha$.*

Consider $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$. Note that this means $u_0 = \overline{v_0} = \varepsilon$. It is easy to see that every word z which belongs to $\mathcal{H}_\alpha(w)$ has a factorization $z = w\overline{u}$ for some $u \in \mathrm{P}_\alpha(w)$ or $z = vw$ for some $\overline{v} \in \mathrm{S}_{\overline{\alpha}}(w)$. By the previous lemma we see that $z \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ and by induction $\mathcal{H}^*_\alpha(w) \subseteq \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$.

The next lemma tells if $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ is a non-crossing (m, n)-word with $n \geq 2$, then the suffix $\overline{\alpha}$ does not overlap with any of the factors α and, therefore, $w \to_{\mathcal{RH}} w\overline{u}$ for all $u \in \mathrm{P}_\alpha(w)$.

Lemma 2. *Let $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ be a non-crossing (m, n)-word with $n \geq 2$ and let u_{m-1} be the longest α-prefix in $\mathrm{P}_\alpha(w)$. Then $|u_{m-1}| + 2k \leq |w|$ holds.*

Since the analogous argument is valid for left hairpin completion, Lemma 2 leads us to one important corollary on non-crossing (m, n)-words for $m, n \geq 2$.

Corollary 2. *Let $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ be a non-crossing (m, n)-word with $m, n \geq 2$.*

$$\mathcal{H}_\alpha(w) = \{w\} \cup w\overline{\mathrm{P}_\alpha(w)} \cup \overline{\mathrm{S}_{\overline{\alpha}}(w)}w.$$

Next, we concern the case when there is a prefix $u \in \mathrm{P}_\alpha(w)$ and a suffix $\overline{v} \in \mathrm{S}_{\overline{\alpha}}(w)$ such that $v \in \mathrm{P}_\alpha(u\alpha)^*$ and $u \in \overline{\mathrm{S}_{\overline{\alpha}}(\overline{\alpha}\overline{v})}^*$.

Lemma 3. *Let $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ be an (m, n)-word. For $u \in \mathrm{P}_\alpha(w)$ and $\overline{v} \in \mathrm{S}_{\overline{\alpha}}(w)$,*

1. *if $v \in \mathrm{P}_\alpha(u\alpha)^*$, then $\overline{\mathrm{S}_{\overline{\alpha}}(\overline{\alpha}\overline{v})} \subseteq \mathrm{P}_\alpha(u\alpha)^*$.*
2. *if $v \in \mathrm{P}_\alpha(u\alpha)^*$ and $u \in \overline{\mathrm{S}_{\overline{\alpha}}(\overline{\alpha}\overline{v})}^*$, then $\mathrm{P}_\alpha(u\alpha)^* = \overline{\mathrm{S}_{\overline{\alpha}}(\overline{\alpha}\overline{v})}^*$.*

Lemma 7 and Corollary 3 in Section 4 will describe the consequences for the iterated hairpin completion of a non-crossing word w, if we find such a situation.

3.2 The α-Index

The α-*index* of a word x is the number of occurrences of the factor α in the word $x\alpha$ except for the suffix. Formally, we define a function $\mathrm{ind}_\alpha \colon \Sigma^* \to \mathbb{N}$ as

$$\mathrm{ind}_\alpha(x) = |\mathrm{P}_\alpha(x\alpha)| - 1.$$

Recall that $\mathrm{P}_\alpha(w) = \{u_0, \ldots, u_{m-1}\}$ such that $u_0 <_p \cdots <_p u_{m-1}$. Note that, by this ordering, for all $0 \leq i < m$ we have $\mathrm{ind}_\alpha(u_i) = i$ and if $x \in \mathrm{P}_\alpha(w)$, then $x = u_{\mathrm{ind}_\alpha(x)}$. Symmetrically, if $\overline{x} \in \mathrm{S}_{\overline{\alpha}}(w)$, then $x = v_{\mathrm{ind}_\alpha(x)}$.

Also note that for words x, y with $\mathrm{ind}_\alpha(x) > \mathrm{ind}_\alpha(y)$ the word x cannot be a factor of y as the positions of the factors α cannot match. Later we will consider the α-indices of words from $\alpha\Sigma^*\alpha^{-1}$ (Note that $x \in \alpha\Sigma^*\alpha^{-1}$ if and only if $\alpha \leq_p x\alpha$). If $x \in \alpha\Sigma^*\alpha^{-1}$ and $y \in \Sigma^*$, then $\mathrm{ind}_\alpha(yx) = \mathrm{ind}_\alpha(y) + \mathrm{ind}_\alpha(x)$. These observations lead to the following properties.

Lemma 4. *Let* $\mathrm{P}_\alpha(w) = \{u_0, \ldots, u_{m-1}\}$ *such that* $u_0 <_p \cdots <_p u_{m-1}$, *let* $x \in \alpha\Sigma^*\alpha^{-1}$, *and let* $0 \leq j < m$. *If* x *is a suffix of* u_j, *then* $u_j = u_{j-\mathrm{ind}_\alpha(x)}x$. *In particular, if* $w \in \alpha\Sigma^*$ *and* $u_j \geq_s u_i$ *for* $0 \leq i \leq j < m$, *then* $u_j = u_{j-i}u_i$.

Lemma 5. *Let* $\mathrm{P}_\alpha(w) = \{u_0, \ldots, u_{m-1}\}$ *such that* $u_0 <_p \cdots <_p u_{m-1}$, *let* $1 \leq i < m$, *and let* x *be a word with* $\mathrm{ind}_\alpha(x) \leq i$.

$$x \in \{u_1, \ldots, u_i\}^* \quad \Longleftrightarrow \quad x \in \left\{u_1, \ldots, u_{\mathrm{ind}_\alpha(x)}\right\}^*.$$

4 Iterated Hairpin Completion of Non-crossing Words

Now we are prepared to prove a necessary and sufficient condition for the regularity of $\mathcal{H}^*_\alpha(w)$, where $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ is a non-crossing (m, n)-word with $\mathrm{P}_\alpha(w) = \{u_0, \ldots, u_{m-1}\}$ and $\mathrm{S}_{\overline{\alpha}}(w) = \{\overline{v_0}, \ldots, \overline{v_{n-1}}\}$ which are ordered as in the previous section. (Keep in mind that $u_0 = \overline{v_0} = \varepsilon$.) By a result from [11] it is enough to consider the case where $m, n \geq 2$ and in this case, by Corollary 2,

$$\mathcal{H}_\alpha(w) = \{w\} \cup w\,\{\overline{u_1}, \ldots, \overline{u_{m-1}}\} \cup \{v_1, \ldots, v_{n-1}\}\,w.$$

Theorem 1 (See [11]). *If* $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ *is a non-crossing* (m, n)*-word with* $m = 1$ *or* $n = 1$, *then* $\mathcal{H}^*_\alpha(w)$ *is regular.*

The next two lemmas lead to a first sufficient condition for the regularity of $\mathcal{H}^*_\alpha(w)$, see Corollary 3.

Lemma 6. *For non-crossing* $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$,

$$\mathcal{H}^*_\alpha(w) \subseteq \left(\mathrm{P}_\alpha(w) \cup \overline{\mathrm{S}_{\overline{\alpha}}(w)}\right)^* w \left(\overline{\mathrm{P}_\alpha(w)} \cup \mathrm{S}_{\overline{\alpha}}(w)\right)^*.$$

Lemma 7. *Let* $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ *be non-crossing. If for some* $u \in \mathrm{P}_\alpha(w)$ *and* $\overline{v} \in \mathrm{S}_{\overline{\alpha}}(w)$ *we have* $v \in \mathrm{P}_\alpha(u\alpha)^*$ *and* $u \in \overline{\mathrm{S}_{\overline{\alpha}}(\overline{\alpha}\overline{v})}^*$, *then*

$$\mathrm{P}_\alpha(u\alpha)^* w \overline{\mathrm{P}_\alpha(u\alpha)}^* = \overline{\mathrm{S}_{\overline{\alpha}}(\overline{\alpha}\overline{v})}^* w \mathrm{S}_{\overline{\alpha}}(\overline{\alpha}\overline{v})^* \subseteq \mathcal{H}^*_\alpha(w).$$

Suppose that the longest α-prefix u_{m-1} belongs to $\overline{\mathrm{S}_{\overline{\alpha}}(w)}^*$ and the longest $\overline{\alpha}$-suffix $\overline{v_{n-1}}$ belongs to $\overline{\mathrm{P}_\alpha(w)}^*$. By Lemma 3, we see that $\mathrm{P}_\alpha(w)^* = \overline{\mathrm{S}_{\overline{\alpha}}(w)}^*$ and, by Lemmas 6 and 7, we infer $\mathcal{H}_\alpha^*(w) = \mathrm{P}_\alpha(w)^* w \overline{\mathrm{P}_\alpha(w)}^*$. (Note that $\mathrm{P}_\alpha(w) = \mathrm{P}_\alpha(u_{m-1}\alpha)$.)

Corollary 3. *Let $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ be non-crossing, let u_{m-1} be the longest α-prefix of w, and let v_{n-1} be the longest $\overline{\alpha}$-suffix of w. If $u_{m-1} \in \overline{\mathrm{S}_{\overline{\alpha}}(w)}^*$ and $v_{n-1} \in \mathrm{P}_\alpha(w)^*$, then $\mathcal{H}_\alpha^*(w)$ is regular.*

Our next result, Theorem 2, shows that if we can state a necessary and sufficient condition for the special case where $u_1 = v_1$, then we can extend this condition to the general case. We need a preliminary lemma in order to prove Theorem 2.

Lemma 8. *Let $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ be a non-crossing (m,n)-word with $m,n \geq 2$, let $\mathrm{P}_\alpha(w) = \{u_0, \ldots, u_{m-1}\}$ such that $u_0 <_p \cdots <_p u_{m-1}$, and let $\mathrm{S}_{\overline{\alpha}}(w) = \{\overline{v_0}, \ldots, \overline{v_{v-1}}\}$ such that $\overline{v_{n-1}} >_s \cdots >_s \overline{v_0}$.*

1. *If $u_1 = v_1$, then $\mathcal{H}_\alpha^*(w) \subseteq u_1\alpha\Sigma^* \cap \Sigma^*\overline{\alpha u_1}$.*
2. *If $u_1 \neq v_1$, then $\mathcal{H}_\alpha^*(w\overline{u_i}) \cap \mathcal{H}_\alpha^*(v_j w) = \emptyset$ for all $1 \leq i < m$ and $1 \leq j < n$.*
3. *Let $1 \leq i < j < m$. If $u_j >_s u_i$, then $\mathcal{H}_\alpha^*(w\overline{u_j}) \subseteq \mathcal{H}_\alpha^*(w\overline{u_i})$; otherwise, $\mathcal{H}_\alpha^*(w\overline{u_i}) \cap \mathcal{H}_\alpha^*(w\overline{u_j}) = \emptyset$.*

Let us define the index sets

$$I = \left\{ i \;\middle|\; 1 \leq i < m \wedge u_i \notin \{u_1, \ldots, u_{i-1}\}^* \right\},$$
$$J = \left\{ j \;\middle|\; 1 \leq j < n \wedge v_j \notin \{v_1, \ldots, v_{j-1}\}^* \right\}.$$

Thus, for all $i \in I$, no proper suffix of u_i belongs to $\mathrm{P}_\alpha(w)$ and for all $j \in J$, no proper prefix of $\overline{v_j}$ belongs to $\mathrm{S}_{\overline{\alpha}}(w)$, see Lemma 4. By the previous lemma, if $v_1 \neq u_1$, then $\mathcal{H}_\alpha^*(w)$ is the disjoint union

$$\mathcal{H}_\alpha^*(w) = \{w\} \cup \bigcup_{i \in I} \mathcal{H}_\alpha^*(w\overline{u_i}) \cup \bigcup_{j \in J} \mathcal{H}_\alpha^*(v_j w).$$

Note that for every word $w\overline{u_i}$ with $i \in I$ or $v_j w$ with $j \in J$, the shortest α-prefix is complementary to the shortest $\overline{\alpha}$-suffix. This observation leads us to an important theorem that allows us to reduce the general case to the special case where $u_1 = v_1$.

Theorem 2. *Let $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ be a non-crossing (m,n)-word with $m,n \geq 2$, let $\mathrm{P}_\alpha(w) = \{u_0, \ldots, u_{m-1}\}$ such that $u_0 <_p \cdots <_p u_{m-1}$, let $\mathrm{S}_{\overline{\alpha}}(w) = \{\overline{v_0}, \ldots, \overline{v_{v-1}}\}$ such that $\overline{v_{n-1}} >_s \cdots >_s \overline{v_0}$, and define I, J as above.*

For $u_1 \neq v_1$, $\mathcal{H}_\alpha^*(w)$ is regular if and only if $\mathcal{H}_\alpha^*(w\overline{u_i})$ is regular for all $i \in I$ and $\mathcal{H}_\alpha^*(v_j w)$ is regular for all $j \in J$.

Theorem 2 justifies the assumption $u_1 = v_1$ that we make from now on. The next two theorems prove a necessary and sufficient condition for the regularity of $\mathcal{H}_\alpha^*(w)$. We start by proving that the condition is sufficient.

Theorem 3. *Let $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ be a non-crossing (m,n)-word with $m,n \geq 2$, let $\mathrm{P}_\alpha(w) = \{u_0, \ldots, u_{m-1}\}$ such that $u_0 <_p \cdots <_p u_{m-1}$, and let $\mathrm{S}_{\overline{\alpha}}(w) = \{\overline{v_0}, \ldots, \overline{v_{v-1}}\}$ such that $\overline{v_{n-1}} >_s \cdots >_s \overline{v_0}$.*

$\mathcal{H}^*_\alpha(w)$ is regular if both of the following conditions hold:

1. *for all $1 \leq s < m$, either $u_s \in \overline{\mathrm{S}_{\overline{\alpha}}(w)}^*$ or $\overline{\mathrm{S}_{\overline{\alpha}}(w)}^* \subseteq \{u_1, \ldots, u_s\}^*$, and*
2. *for all $1 \leq t < n$, either $v_t \in P_\alpha(w)^*$ or $P_\alpha(w) \subseteq \{v_1, \ldots, v_t\}^* t$.*

Proof. Assume that both the conditions 1 and 2 are satisfied. We may assume that there is $1 \leq s < m$ such that $u_s \notin \overline{\mathrm{S}_{\overline{\alpha}}(w)}^*$ or there is $1 \leq t < n$ such that $v_t \notin P_\alpha(w)^*$; otherwise Corollary 3 yields the regularity. In addition, we cannot assume the existence of both such s and t as $u_s \notin \overline{\mathrm{S}_{\overline{\alpha}}(w)}^*$ implies $\overline{\mathrm{S}_{\overline{\alpha}}(w)}^* \subseteq \{u_1, \ldots, u_s\}^*$ due to the condition 1. The symmetry in the roles of conditions 1 and 2 enables us to assume that such s exists without loss of generality, and moreover, we can assume that for all $1 \leq i < s$, $u_i \in \overline{\mathrm{S}_{\overline{\alpha}}(w)}^*$ and for all $s \leq i < m$, $u_i \notin \overline{\mathrm{S}_{\overline{\alpha}}(w)}^*$ by Lemma 3.

Let $R' = \overline{\mathrm{S}_{\overline{\alpha}}(w)}^* w \{\overline{u_1}, \ldots, \overline{u_{s-1}}\}^*$ and for $s \leq i < m$, let

$$R_i = \{u_1, \ldots, u_i\}^* w \{\overline{u_1}, \ldots, \overline{u_i}\}^* \overline{u_i} \{\overline{u_1}, \ldots, \overline{u_i}\}^*.$$

Then we let $R = \bigcup_{s \leq i < m} R_i \cup R'$ and we claim $\mathcal{H}^*_\alpha(w) = R$.

Firstly, we prove that $R \subseteq \mathcal{H}^*_\alpha(w)$. Since $m, n \geq 2$, Corollary 2 can be used to see that $w\{\overline{u_1}, \ldots, \overline{u_i}\}^* \overline{u_i} \subseteq \mathcal{H}^*_\alpha(w)$ for $s \leq i < m$, and by Lemma 7,

$$R_i = \{u_1, \ldots, u_i\}^* w \{\overline{u_1}, \ldots, \overline{u_i}\}^* \overline{u_i} \{\overline{u_1}, \ldots, \overline{u_i}\}^* \subseteq \mathcal{H}^*_\alpha(w).$$

Consider $z \in R'$. We may factorize $z = x_i \cdots x_1 w \overline{y_1} \cdots \overline{y_j}$ where $x_1, \ldots, x_i \in \overline{\mathrm{S}_{\overline{\alpha}}(w)}$ and $y_1, \ldots, y_j \in \{u_1, \ldots, u_{s-1}\}$. Let $1 \leq t < n$ be minimal such that $v_t \notin \{u_1, \ldots, u_{s-1}\}^*$. As $v_t \in \{u_1, \ldots, u_s\}^*$, we see that u_s is a factor of v_t and $t \geq s$. By Lemma 5, $u_{s-1} \in \{v_1, \ldots, v_{t-1}\}^*$ and, by the minimality of t, $v_{t-1} \in \{u_1, \ldots, u_{s-1}\}^*$. If $\mathrm{ind}_\alpha(x_\ell) < t$ for all $1 \leq \ell \leq i$, then, due to Lemma 7,

$$z \in \{v_1, \ldots, v_{t-1}\}^* w \{\overline{v_1}, \ldots, \overline{v_{t-1}}\}^* \subseteq \mathcal{H}^*_\alpha(w).$$

Otherwise, let $1 \leq \ell \leq i$ be maximal such that $\mathrm{ind}_\alpha(x_\ell) \geq t$ and let $w' = x_\ell \cdots x_1 w$. Observe that $w \to^*_{\mathcal{LH}} w'$ and, again by Lemma 7,

$$z \in \{v_1, \ldots, v_{t-1}\}^* w' \{\overline{v_1}, \ldots, \overline{v_{t-1}}\}^* \subseteq \mathcal{H}^*_\alpha(w') \subseteq \mathcal{H}^*_\alpha(w).$$

Thus, $R \subseteq \mathcal{H}^*_\alpha(w)$.

Now we prove the opposite inclusion by induction on the length of a derivation to generate a word in $\mathcal{H}^*_\alpha(w)$ from w. Clearly, $w \in R$ (base case). As an induction hypothesis, we assume that any word which can be derived from w by $\ell - 1$ hairpin completions is in R, and consider $z \in \mathcal{H}^*_\alpha(w)$ whose shortest derivation from w by hairpin completions is of length ℓ. Let w' be the word that precedes z on this shortest derivation, that is, $w' \in \mathcal{H}^{\ell-1}_\alpha(w)$; hence, $w' \in R$ by the induction hypothesis. Therefore, w' must be either in R_i for some $s \leq i < m$ or in R'. Let us consider the first case first. If z is obtained from w' by right hairpin completion,

then the complement of the extended part is in $\{u_1, \ldots, u_i\}^*\{\varepsilon, u_1, \ldots, u_{m-1}\}$, and hence, $z \in R_j$ for some $i \leq j < m$. Otherwise $(w' \to_{\mathcal{LH}} z)$, $z \in R_i$ because $\overline{\mathrm{S}_{\overline{\alpha}}(w')} \subseteq \{u_1, \ldots, u_s\}^* \subseteq \{u_1, \ldots, u_i\}^*$. Next we consider the case when $w' \in R'$. Since $\{u_1, \ldots, u_{s-1}\}^* \subseteq \overline{\mathrm{S}_{\overline{\alpha}}(w)}^*$ it is easy to see that if $w' \to_{\mathcal{LH}} z$, then $z \in R'$ as well. Otherwise $(w' \to_{\mathcal{RH}} z)$, $z \in w'\{\varepsilon, \overline{u_1}, \ldots, \overline{u_{m-1}}\}\,\mathrm{S}_{\overline{\alpha}}(w)^*$ and, as $\mathrm{S}_{\overline{\alpha}}(w)^* \subseteq \{\overline{u_1}, \ldots, \overline{u_s}\}^*$, if $z \notin R'$, this word is covered by some language R_i where $s \leq i < m$.

Consequently, $\mathcal{H}^*_\alpha(w) = R$ is regular. □

Theorem 4. *Let $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ be a non-crossing (m, n)-word with $m, n \geq 2$, let $\mathrm{P}_\alpha(w) = \{u_0, \ldots, u_{m-1}\}$ such that $u_0 <_p \cdots <_p u_{m-1}$, let $\mathrm{S}_{\overline{\alpha}}(w) = \{\overline{v_0}, \ldots, \overline{v_{v-1}}\}$ such that $\overline{v_{n-1}} >_s \cdots >_s \overline{v_0}$, and let $u_1 = v_1$.*

1. *$\mathcal{H}^*_\alpha(w)$ is not regular if there are $1 \leq s < m$ and $1 \leq t < n$ such that $u_s \notin \{v_1, \ldots, v_{n-1}\}^*$ and $v_t \notin \{u_1, \ldots, u_s\}^*$.*
2. *$\mathcal{H}^*_\alpha(w)$ is not regular if there are $1 \leq s < m$ and $1 \leq t < n$ such that $u_s \notin \{v_1, \ldots, v_t\}^*$ and $v_t \notin \{u_1, \ldots, u_{m-1}\}^*$.*

Proof. Let s and t be the minimal indices that satisfy the conditions in statement 1. Note that $s, t \geq 2$ and $u_s, v_t \notin u_1^+$ as $u_1 = v_1$. We will first argue, why the assumption $s \leq t$ is no restriction.

Let us consider the case where the conditions in statement 1 are satisfied, but the conditions in statement 2 are not satisfied. It is easy to see that $u_s \notin \{v_1, \ldots, v_t\}^*$ is satisfied anyway. By contradiction assume $s > t$. Due to Lemma 5,

$$\begin{aligned} v_t \notin \{u_1, \ldots, u_s\}^* \quad &\iff \quad v_t \notin \{u_1, \ldots, u_t\}^* \\ &\iff \quad v_t \notin \{u_1, \ldots, u_{m-1}\}^* . \end{aligned}$$

This satisfies the conditions of statement 2 and yields the contradiction. We conclude $s \leq t$.

Now, let us consider the case where the conditions of both statements are satisfied. Let s and t' be the minimal indices such that $u_s \notin \{v_1, \ldots, v_{n-1}\}^*$ and $v_{t'} \notin \{u_1, \ldots, u_{m-1}\}^*$. We may assume $s \leq t'$, by symmetry. Note that $v_{t'-1} \in \{u_1, \ldots, u_{m-1}\}^*$, by the minimality of t'. If $v_{t'-1} \in \{u_1, \ldots, v_s\}^*$, then we see that s and $t = t'$ are the minimal indices that satisfy the conditions in statement 1 and $s \leq t$. Otherwise, there is a factorization $v_{t'-1} = xu_iy$ where $x \in \{u_1, \ldots, u_s\}^*$, $s < i < m$, and $y \in \{u_1, \ldots, u_{m-1}\}^*$. Note that s and $t = \mathrm{ind}_\alpha(x) + s + 1$ (hence $v_t = xu_{s+1}$) are the minimal indices that satisfy the conditions of statement 1 and, obviously, $s \leq t$.

Observe that the minimality of s yields $u_1, \ldots, u_{s-1} \in \{v_1, \ldots, v_{n-1}\}^*$. If $x \in \{u_1, \ldots, u_{s-1}, v_1, \ldots, v_{n-1}\}$ was a suffix of u_s, then $u_s = u_{s-\mathrm{ind}_\alpha(x)}x \in \{v_1, \ldots, v_{n-1}\}^*$, due to Lemma 4. Hence, none of these words is a suffix of u_s. Symmetrically, none of the words $u_1, \ldots, u_s$, $v_1, \ldots, v_{t-1}$ is a suffix of v_t. This observation will become crucial later.

We will now define a regular language R and show that the intersection $\mathcal{H}^*_\alpha(w) \cap R$ is not regular and, therefore, $\mathcal{H}^*_\alpha(w)$ is not regular either. We let

$$R = u_s u_1^{\geq n} v_t w \overline{u_1}^{\geq n} \overline{u_s}$$

and we claim

$$\mathcal{H}^*_\alpha(w) \cap R = \left\{ u_s u_1^\ell v_t w \overline{u_1}^\ell \overline{u_s} \;\middle|\; \ell \geq n \right\} =: L,$$

which is obviously not regular. Note that for every $\ell \geq n$

$$w \to^*_{\mathcal{RH}} w\overline{u_1}^\ell \to_{\mathcal{RH}} w\overline{u_1}^\ell \overline{u_s} \to_{\mathcal{LH}} u_s u_1^\ell v_t w \overline{u_1}^\ell \overline{u_s}.$$

Hence, $\mathcal{H}^*_\alpha(w) \cap R \supseteq L$.

Let $z = u_s u_1^{\ell_1} v_t w \overline{u_1}^{\ell_2} \overline{u_s}$ for some $\ell_1, \ell_2 \geq n$, which is in R. We assume $z \in \mathcal{H}^*_\alpha(w)$ and prove that this assumption requires $\ell_1 = \ell_2$. Let z' be the *right-most* word in the derivation $w \to^*_{\mathcal{H}} z' \to^*_{\mathcal{H}} z$ such that $z' = xwy$ for some words x, y with $u_1^{\ell_1} v_t \geq_s x$ and $y \leq_p \overline{u_1}^{\ell_2}$; these conditions mean that x or y does not overlap with the prefix u_s or the suffix $\overline{u_s}$, respectively. By right-most we mean that either $z' \to_{\mathcal{LH}} x'z' \to^*_{\mathcal{H}} z$ where $x'x >_s u_1^{\ell_1} v_t$ or $z' \to_{\mathcal{RH}} z'y' \to^*_{\mathcal{H}} z$ where $\overline{u_1}^{\ell_2} <_p yy'$; this means x' or y' overlaps with the prefix u_s or the suffix $\overline{u_s}$, respectively. Obviously, $y \in \overline{u_1}^*$. Note that if $x \neq \varepsilon$, then x cannot be a proper suffix of v_t; otherwise a word from $u_1 = v_1, \ldots, v_{t-1}$ would be a suffix of v_t which was excluded. Hence, $x = \varepsilon$ or $x \in u_1^* v_t$.

First consider the case $z' \to_{\mathcal{LH}} x'z' \to^*_{\mathcal{H}} z$ where $x'x >_s u_1^{\ell_1} v_t$. We show that this case cannot occur. Let $u' \neq \varepsilon$ be the suffix of u_s such that $u' u_1^{\ell_1} v_t = x'x$. As $x' \in u_1^* \{v_1, \ldots, v_{n-1}\}$ and $u'\alpha \leq_p x'\alpha$, some word from $u_1 = v_1, \ldots, v_{n-1}$ would be a suffix of u_s.

Now consider the case $z' \to_{\mathcal{RH}} z'y' \to^*_{\mathcal{H}} z$ where $\overline{u_1}^{\ell_2} <_p yy'$. Again, let $u' \neq \varepsilon$ be the suffix of u_s such that $\overline{u_1}^{\ell_2} \overline{u'} = yy'$. As $u'\alpha$ is a prefix of $xu_{m-1}\alpha$ and none of the words $v_1, \ldots, v_t$, $u_1, \ldots, u_{s-1}$ is a suffix of u_s, we see that $u' = u_s$ and $x = \varepsilon$.

Thus, in order to generate z from w by iterated hairpin completion, the derivation process must be of the form

$$w \to^*_{\mathcal{RH}} w\overline{u_1}^{\ell_2} \overline{u_s} \to^*_{\mathcal{LH}} u_s u_1^{\ell_1} v_t w \overline{u_1}^{\ell_2} \overline{u_s} = z.$$

Let x be a (newly chosen) word such that $w\overline{u_1}^{\ell_2}\overline{u_s} \to_{\mathcal{LH}} xw\overline{u_1}^{\ell_2}\overline{u_s}$ is the first left hairpin completion in the derivation above. Therefore, $x\alpha$ is a prefix of $u_s u_1^{\ell_2} v_{n-1}\alpha$ and x is a suffix of $u_s u_1^{\ell_1} v_t$. In particular, every suffix y of x with $\mathrm{ind}_\alpha(y) \leq t$ is a suffix of v_t. Recall that $\mathrm{ind}_\alpha(u_s) = s \leq t$. If $x\alpha$ was a prefix of $u_s\alpha$, then some word from $u_1, \ldots, u_s$ would be a suffix of v_t which is impossible. Verify that $x \in u_s u_1^+$ and $x = u_s u_1^{\ell_2} v_j$ with $1 \leq j < t$ would also impose a forbidden suffix for v_t. Thus, we see that $x = u_s u_1^{\ell_2} v_j$ with $t \leq j < n$. The case $j > t$ is not possible as it implies $v_t\alpha <_p v_j\alpha = u_1^{j-t} v_t\alpha$ and a word from $u_1 = v_1, \ldots, v_{t-1}$ would be a suffix of v_t. Therefore, $x = u_s u_1^{\ell_2} v_t$ and since x is a suffix of $u_s u_1^{\ell_1} v_t$ and u_1 is not a suffix of u_s we deduce $u_s u_1^{\ell_2} v_t = u_s u_1^{\ell_1} v_t$.

Consequentially, $z \in \mathcal{H}^*_\alpha(w)$ if and only if $\ell_1 = \ell_2$. This completes the proof of $\mathcal{H}^*_\alpha(w) \cap R = L$. □

Combining Theorems 3 and 4, we conclude:

Corollary 4. *Let $w \in \alpha\Sigma^* \cap \Sigma^*\overline{\alpha}$ be a non-crossing (m,n)-word with $m,n \geq 2$, let $\mathrm{P}_\alpha(w) = \{u_0, \ldots, u_{m-1}\}$ such that $u_0 <_p \cdots <_p u_{m-1}$, let $\mathrm{S}_{\overline{\alpha}}(w) = \{\overline{v_0}, \ldots, \overline{v_{v-1}}\}$ such that $\overline{v_{n-1}} >_s \cdots >_s \overline{v_0}$, and let $u_1 = v_1$.*
$\mathcal{H}^*_\alpha(w)$ is regular if and only if

1. *for all $1 \leq s < m$ we have $u_s \in \overline{\mathrm{S}_{\overline{\alpha}}(w)}^*$ or $\overline{\mathrm{S}_{\overline{\alpha}}(w)} \subseteq \{u_1, \ldots, u_s\}^*$ and*
2. *for all $1 \leq t < n$ we have $v_t \in \mathrm{P}_\alpha(w)^*$ or $\mathrm{P}_\alpha(w) \subseteq \{v_1, \ldots, v_t\}^*$.*

Thus, we provided a necessary and sufficient condition for the regularity of a non-crossing (m,n)-word. As one can easily observe, this condition is decidable.

Corollary 5. *For a given non-α-crossing word w, it is decidable whether or not its iterated hairpin completion, $\mathcal{H}^*_\alpha(w)$, is regular.*

Furthermore, we can derive from the proof of Theorem 4 that if the iterated hairpin completion of w is not regular, then the intersection of $\mathcal{H}^*_\alpha(w)$ with $R = u_s u_1^{\geq n} v_t w \overline{u_1}^{\geq n} \overline{u_s}$ is not a regular language (for suitable s,t and in case $u_1 = v_1$). More precisely, we obtained the context-free language

$$\mathcal{H}^*_\alpha(w) \cap R = \left\{ u_s u_1^\ell v_t w \overline{u_1}^\ell \overline{u_s} \;\middle|\; \ell \geq n \right\}.$$

Consider we intersect $\mathcal{H}^*_\alpha(w)$ with $R' = (u_s u_1^{\geq n} v_t)^2 w \overline{u_1}^{\geq n} \overline{u_s}$. Using the same arguments as we did within the proof of Theorem 4, we can show that

$$\mathcal{H}^*_\alpha(w) \cap R' = \left\{ u_s u_1^\ell v_t u_s u_1^\ell v_t w \overline{u_1}^\ell \overline{u_s} \;\middle|\; \ell \geq n \right\},$$

which is a non-context-free language. Using this idea we can proof that if $\mathcal{H}^*_\alpha(w)$ is not regular, it is not context-free either. The details of this proof are left for the interested reader.

Corollary 6. *Let w be a non-α-crossing word. If its iterated hairpin completion $\mathcal{H}^*_\alpha(w)$ is not regular, then $\mathcal{H}^*_\alpha(w)$ is not context-free.*

Final Remarks

We prove that regularity of iterated hairpin completion a given of non-crossing word is decidable. The general case, including that of crossing words, remains to be explored.

References

1. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science 266(5187), 1021–1024 (1994)
2. Arita, M., Kobayashi, S.: DNA sequence design using templates. New Generation Computing 20, 263–277 (2002)

3. Cheptea, D., Martín-Vide, C., Mitrana, V.: A new operation on words suggested by DNA biochemistry: Hairpin completion. Transgressive Computing, 216–228 (2006)
4. Diekert, V., Kopecki, S.: Complexity Results and the Growths of Hairpin Completions of Regular Languages (Extended Abstract). In: Domaratzki, M., Salomaa, K. (eds.) CIAA 2010. LNCS, vol. 6482, pp. 105–114. Springer, Heidelberg (2011)
5. Hagiya, M., Arita, M., Kiga, D., Sakamoto, K., Yokoyama, S.: Towards parallel evaluation and learning of boolean μ-formulas with molecules. In: Second Annual Genetic Programming Conf., pp. 105–114 (1997)
6. Hopcroft, J.E., Ulman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
7. Jonoska, N., Kephart, D., Mahalingam, K.: Generating DNA codewords. Congressus Numerantium 156, 99–110 (2002)
8. Jonoska, N., Mahalingam, K.: Languages of DNA Based Code Words. In: Chen, J., Reif, J. (eds.) DNA 2003. LNCS, vol. 2943, pp. 61–73. Springer, Heidelberg (2004)
9. Kameda, A., Yamamoto, M., Ohuchi, A., Yaegashi, S., Hagiya, M.: Unravel four hairpins! Natural Computing 7, 287–298 (2008)
10. Kari, L., Konstantinidis, S., Losseva, E., Sosík, P., Thierrin, G.: A formal language analysis of DNA hairpin structures. Fundamenta Informaticae 71(4), 453–475 (2006)
11. Kari, L., Kopecki, S., Seki, S.: On the Regularity of Iterated Hairpin Completion of a Single Word. Fundamenta Informaticae 110(1-4), 201–215 (2011)
12. Kari, L., Kopecki, S., Seki, S.: Iterated Hairpin Completions of Non-crossing Words in arXiv:1110.0760
13. Kopecki, S.: On iterated hairpin completion. Theoretical Computer Science 412(29), 3629–3638 (2011)
14. Manea, F.: A series of algorithmic results related to the iterated hairpin completion. Theor. Comput. Sci. 411(48), 4162–4178 (2010)
15. Manea, F., Martín-Vide, C., Mitrana, V.: On some algorithmic problems regarding the hairpin completion. Discrete Applied Mathematics 157(9), 2143–2152 (2009)
16. Manea, F., Mitrana, V.: Hairpin Completion Versus Hairpin Reduction. In: Cooper, S.B., Löwe, B., Sorbi, A. (eds.) CiE 2007. LNCS, vol. 4497, pp. 532–541. Springer, Heidelberg (2007)
17. Manea, F., Mitrana, V., Yokomori, T.: Two complementary operations inspired by the DNA hairpin formation: Completion and reduction. Theor. Comput. Sci. 410(4-5), 417–425 (2009)
18. Manea, F., Mitrana, V., Yokomori, T.: Some remarks on the hairpin completion. Int. J. Found. Comput. Sci. 21(5), 859–872 (2010)
19. Păun, G., Rozenberg, G., Yokomori, T.: Hairpin languages. International Journal of Foundations of Computer Science 12(6), 837–847 (2001)
20. Sakamoto, K., Kiga, D., Komiya, K., Gouzu, H., Yokoyama, S., Ikeda, S., Hagiya, M.: State transitions by molecules (1998)
21. Takinoue, M., Suyama, A.: Molecular reactions for a molecular memory based on hairpin DNA. Chem-Bio Informatics Journal 4, 93–100 (2004)
22. Takinoue, M., Suyama, A.: Hairpin-DNA memory using molecular addressing. Small 2(11), 1244–1247 (2006)
23. Winfree, E.: Whiplash PCR for O(1) computing. pp. 175–188. University of Pennsylvania (1998)

On the Approximation Ratio of the Path Matching Christofides Algorithm*

Sacha Krug

Department of Computer Science, ETH Zurich, Switzerland
sacha.krug@inf.ethz.ch

Abstract. The traveling salesman problem (TSP) is one of the most fundamental optimization problems. We consider the β-metric traveling salesman problem (Δ_β-TSP), i.e., the TSP restricted to graphs satisfying the β-triangle inequality $c(\{v,w\}) \leq \beta(c(\{v,u\}) + c(u,w\}))$, for some cost function c and any three vertices u, v, w. The well-known path matching Christofides algorithm (PMCA) guarantees an approximation ratio of $\frac{3}{2}\beta^2$ and is the best known algorithm for the Δ_β-TSP, for $1 \leq \beta \leq 2$. We provide a complete analysis of the algorithm. First, we correct an error in the original implementation that may produce an invalid solution. Using a worst-case example, we then show that the algorithm cannot guarantee a better approximation ratio. The example can be reused for the PMCA variants for the Hamiltonian path problem with zero and one prespecified endpoints. For two prespecified endpoints, we cannot reuse the example, but we construct another worst-case example to show the optimality of the analysis also in this case.

Keywords: approximation algorithms, traveling salesman problem, Hamiltonian path problem

1 Introduction

The traveling salesman problem (TSP) is one of the most studied optimization problems. In its most general form, it is not approximable by any polynomial. Certain subsets of the TSP, however, allow for a constant-factor approximation. One such subset is the metric TSP (Δ-TSP), i.e., the TSP restricted to input graphs satisfying the triangle inequality. Christofides' algorithm [6] provides a $\frac{3}{2}$-approximation for the Δ-TSP. A very natural idea is to try to apply this algorithm to a wider set of input instances. This idea is captured by the concept of stability of approximation [3,5,4,11] that provides a formalism to express the changes of the approximation ratio of an algorithm when a different set of input instances is considered.

A natural generalization of the metric TSP is to consider the β-metric TSP (Δ_β-TSP), i.e., the TSP restricted to graphs (V, E) satisfying the β-triangle inequality $c(\{v,w\}) \leq \beta \cdot (c(\{v,u\}) + c(\{u,w\}))$, for some cost function $c : E \to \mathbb{Q}^+$

* This work was partially supported by SNF grant No. 200021-132510/1.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 349–360, 2012.

and any three vertices u, v, w. Böckenhauer et al. [4] showed that Christofides' algorithm, when applied to Δ_β-TSP instances, for some $\beta > 1$, no longer provides a constant approximation ratio of $\frac{3}{2}$, but an approximation ratio that depends on the size of the input graph. Therefore, the authors devised the path matching Christofides algorithm (PMCA) that provides an approximation ratio of $\frac{3}{2}\beta^2$, for any $\beta \geq 1$. Other algorithms for the Δ_β-TSP, for $\beta \geq 1$, are due to Andreae [1] and Bender and Chekuri [2] and provide approximation ratios of $\beta^2 + \beta$ and 4β, respectively. Forlizzi et al. [8] combined the path matching Christofides algorithm and Hoogeveen's approximation algorithm for the Hamiltonian path problem [10] to obtain an approximation algorithm for the β-metric Hamiltonian path problem with $l \in \{0, 1, 2\}$ prespecified endpoints (PMCA-HPP$_l$) that provides an approximation ratio of $\frac{3}{2}\beta^2$, for $l = 0, 1$, and of $\frac{5}{3}\beta^2$, for $l = 2$.

In this paper, we show that the four PMCA variants cannot provide better approximation ratios. In Section 2, we construct a TSP instance on which the PMCA cannot achieve an approximation ratio of $\frac{3}{2}\beta^2 - \varepsilon$, for any $\varepsilon > 0$. This instance can also be used to establish tight lower bounds for the PMCA-HPP$_0$ and the PMCA-HPP$_1$. For the PMCA-HPP$_2$, it is not possible to reuse this example, as we shall see. We therefore construct another worst-case instance in Section 3 to prove the optimality of the analysis also in this case.

We stick to the notation used in [4]. Formally, the *β-metric traveling salesman problem (Δ_β-TSP)* is the following optimization problem. Given a complete graph G with edge costs that satisfy the β-triangle inequality, find a cycle in G that visits every vertex exactly once and has minimum overall cost. A *path matching* for a vertex set V of even size is a set of $|V|/2$ edge-disjoint paths having the vertices in V as its disjoint endpoints. Let $p = (v_1, v_2, \ldots, v_k)$ be a path. A vertex v is internal to p if $v = v_i$, for some $1 < i < k$. A *bypass* in p is an edge $\{u, v\}$ replacing a sub-path $(u = v_i, v_{i+1}, \ldots, v_j = v)$, for $1 \leq i < i + 1 < j \leq k$. Also, we say that the vertices $v_{i+1}, v_{i+2}, \ldots, v_{j-1}$ are *bypassed*. A *conflict* in a set of paths is a vertex that occurs in more than one path. A *conflict* in an Eulerian cycle is a vertex that is visited more than once.

2 The PMCA for the Traveling Salesman Problem

In this section, we first briefly explain the PMCA (Algorithm 1) and then construct a worst-case example to prove that the approximation ratio of the algorithm cannot be improved.

The implementation of steps 1, 2, and 4 are well-known [4,7,9]. Note, however, that in order for the edges of each path in M' to remain consecutive in E, each such path has to be regarded as a single edge while computing the Eulerian cycle in step 4. In step 5, an arbitrary root vertex r is chosen. Then, in every path p_i, the vertex v closest to r in T is bypassed if v is internal to p_i and if v is adjacent to at least four edges from T. This last condition was not stated in [4], but is indeed necessary, as otherwise the algorithm might drop certain vertices, i.e., they might not appear in the end result, which would then by definition no longer be a Hamiltonian cycle.

Algorithm 1. Path Matching Christofides Algorithm [4]

Input: A complete β-metric graph G, for some $\beta \geq 1$.

1: Find a minimum spanning tree T in G. Let U be all odd-degree vertices in T.
2: Construct a minimum path matching M for U. {The matching is edge-disjoint as a direct consequence of its minimality.}
3: Resolve conflicts in M to obtain a vertex-disjoint path matching M'.
4: Construct an Eulerian cycle $E := (p_1, q_1, p_2, q_2, \ldots)$ on T and M' such that $p_1, p_2, \ldots$ are paths in T and $q_1, q_2, \ldots$ are paths in M'.
5: Transform $p_1, p_2, \ldots$ into $p'_1, p'_2, \ldots$ such that the forest T_f formed by $p'_1, p'_2, \ldots$ has maximum degree 3. Let $E' := (p'_1, q_1, p'_2, q_2, \ldots)$.
6: Resolve all remaining conflicts in E' to obtain a Hamiltonian cycle H.

Output: H.

Algorithm 2 shows the implementation of step 3. Roughly, it consists of two parts. First, a path with only one conflict is searched[1], and then this conflict is resolved. The last step is implemented as follows. First, bypass an arbitrary conflict x. If there are neighbors of x that are conflicts, bypass one of them. Else, bypass an arbitrary conflict. Repeat this until no conflicts are left.

Theorem 1. *For every $\beta \geq 1$, the PMCA provides an approximation ratio of $\frac{3}{2}\beta^2$, and there exists an infinite family of graphs satisfying the β-triangle inequality on which it cannot achieve a better appproximation ratio.*

Algorithm 2.

Input: An edge-disjoint, cycle-free path matching M for U in G.

while M has conflicts **do**
  pick an arbitrary path p that has at least one internal conflict or at least two conflicts
  while p has more than one conflict **do**
    let v, w be the first and the last conflict in p
    let p_v, p_w be paths that contain v resp. w
    pick as new p one of p_v, p_w that was formerly not picked
  end while
  let v be the only conflict in the finally chosen path p
  if v is internal to p **then**
    bypass v in p
  else
    bypass the unique edge incident to v in p together with one edge of the previously picked path as shown in Fig. 1
  end if
end while

Output: A vertex-disjoint path matching $M' = (q_1, q_2, \ldots)$.

[1] Such a path always exists because the graph formed by M is cycle-free due to the minimality of M.

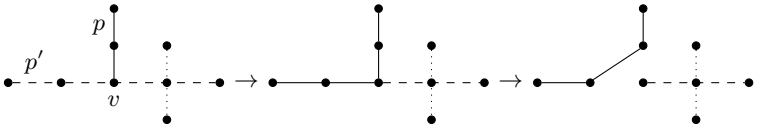


Fig. 1. Algorithm 2 first picks p' and then p. It takes two edges indicent to v, one from p and one from p', and replaces them with a bypass.

The first part was shown in [4]. To prove the second part, we first introduce a graph and show that it contains a Hamiltonian cycle of a certain cost. Then, we present one possible implementation of the PMCA on this graph in order to obtain the desired lower bound.

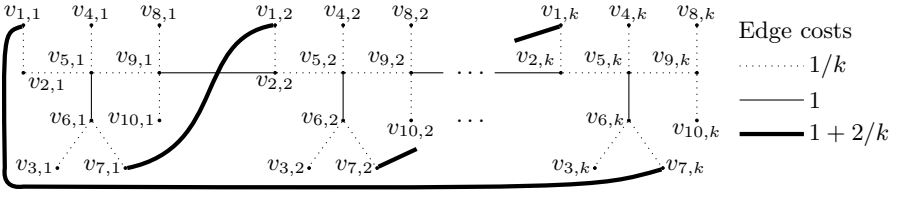


Fig. 2. The graph $G_{10,k}(\beta)$

Let $G_{10,k}(\beta)$ be the complete graph shown in Fig. 2, where all edges not shown have maximum possible cost such that the β-triangle inequality is satisfied.

Lemma 1. *The graph $G_{10,k}(\beta)$ satisfies the β-triangle inequality.*

Proof. The proof is omitted due to space limitations. It can be found in [12]. □

Fig. 3 shows a Hamiltonian cycle in $G_{10,k}(\beta)$ of cost $2k + 2\beta^2 + 7\beta + 6$. We now show one possible implementation of the PMCA that, on input $G_{10,k}(\beta)$, returns a Hamiltonian cycle of cost at least $3(k-1)\beta^2$.

In the first step, the PMCA computes the minimum spanning tree T shown in Fig. 4. The vertices in U are circled. It is easy to see that T is indeed a minimum spanning tree. Every edge in the graph has cost at least $1/k$, therefore the edges $\{v_{1,i}, v_{2,i}\}, \{v_{2,i}, v_{5,i}\}, \{v_{4,i}, v_{5,i}\}, \{v_{5,i}, v_{9,i}\}, \{v_{8,i}, v_{9,i}\}, \{v_{9,i}, v_{10,i}\}$, for $1 \leq i \leq k$, form minimum spanning trees for the respective upper subclusters. On the other hand, the edges $\{v_{3,i}, v_{6,i}\}, \{v_{6,i}, v_{7,i}\}$, for $1 \leq i \leq k$, form minimum spanning trees for the respective lower subclusters. All we need to do is add an edge for every component to construct a minimum spanning tree for the whole graph. All edges available for this have cost at least 1, so we can just take the edges $\{v_{5,i}, v_{6,i}\}$, for $1 \leq i \leq k$, and $\{v_{9,i}, v_{2,i+1}\}$, for $1 \leq i \leq k-1$.

In the second step, the PMCA computes the minimum path matching M for U shown in Fig. 5.

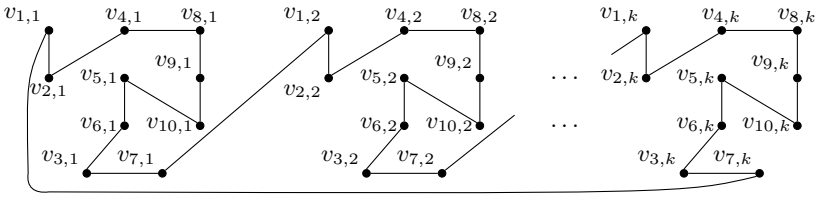


Fig. 3. A Hamiltonian cycle of cost $2k + 2\beta^2 + 7\beta + 6$ in $G_{10,k}(\beta)$

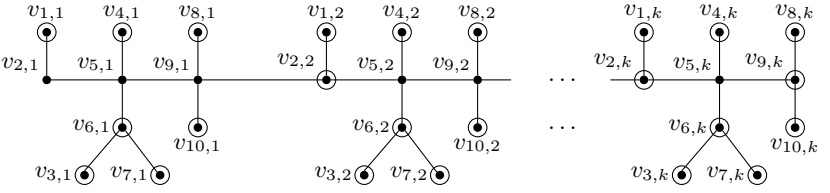


Fig. 4. A minimum spanning tree in $G_{10,k}(\beta)$. The odd-degree vertices are circled.

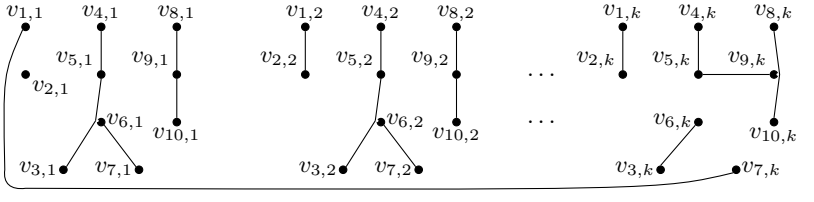


Fig. 5. A minimum path matching for U in $G_{10,k}(\beta)$

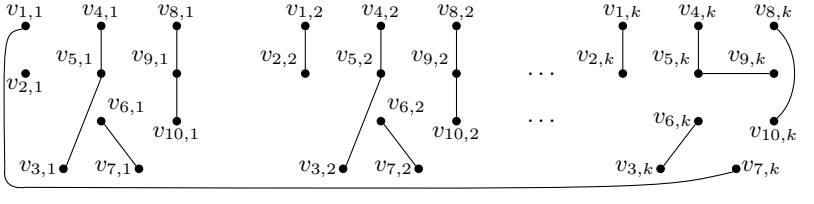


Fig. 6. A vertex-disjoint path matching for U in $G_{10,k}(\beta)$

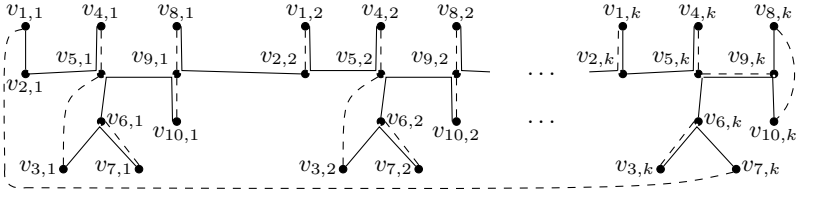


Fig. 7. The Eulerian cycle E. For clarity, the paths of M' are dashed.

Theorem 2. *M is a minimum path matching for U.*

Proof. The proof is omitted due to space limitations. It can be found in [12]. □

The goal of the third step is to resolve all conflicts in the minimum path matching M to obtain a vertex-disjoint path matching M'. The only problematic paths in M are $(v_{4,i}, v_{5,i}, v_{6,i}, v_{3,i})$ and $(v_{6,i}, v_{7,i})$, for $1 \leq i \leq k-1$, as well as $(v_{4,k}, v_{5,k}, v_{9,k})$ and $(v_{8,k}, v_{9,k}, v_{10,k})$. For each component in the first set, the PMCA may choose the path $(v_{4,i}, v_{5,i}, v_{6,i}, v_{3,i})$ as p and thus bypass $v_{6,i}$ in this path. For the second set, the PMCA may choose the path $(v_{8,k}, v_{9,k}, v_{10,k})$ as p and thus bypass $v_{9,k}$ in this path. In this step, the PMCA thus computes the vertex-disjoint path matching M' shown in Fig. 6.

Alternating between paths from T and paths from M', the PMCA computes in the fourth step the Eulerian cycle E shown in Fig. 7.

The goal of the fifth step is that every vertex is adjacent to at most three edges from T. The problematic vertices are thus all $v_{5,i}$ and all $v_{9,i}$ except $v_{9,k}$. Let $r := v_{1,1}$. Then, the PMCA bypasses the vertices $v_{5,i}$ between $v_{6,i}$ and $v_{9,i}$, for $1 \leq i \leq k-1$, the vertex $v_{5,k}$ between $v_{9,k}$ and $v_{6,k}$, and the vertices $v_{9,i}$ between $v_{8,i}$ and $v_{2,i+1}$, for $1 \leq i \leq k-1$, to obtain the modified Eulerian cycle E' shown in Fig. 8.

The goal of the last step is that every vertex has degree 2. The problematic vertices are thus all vertices $v_{2,i}$ except $v_{2,1}$ and all vertices $v_{5,i}, v_{6,i}, v_{9,i}$. The PMCA obtains the Hamiltonian cycle H shown in Fig. 9 by bypassing

- $v_{2,i}$ between $v_{8,i-1}$ and $v_{1,i}$, for $2 \leq i \leq k$,
- $v_{5,i}$ between $v_{4,i}$ and $v_{3,i}$, for $1 \leq i \leq k-1$, and $v_{5,k}$ between $v_{4,k}$ and $v_{9,k}$,
- $v_{6,i}$ between $v_{7,i}$ and $v_{9,i}$, for $1 \leq i \leq k-1$, and $v_{6,k}$ between $v_{3,k}$ and $v_{7,k}$,
- $v_{9,i}$ between $v_{10,i}$ and $v_{8,i}$, for $1 \leq i \leq k-1$, and $v_{9,k}$ between $v_{10,k}$ and $v_{6,k}$.

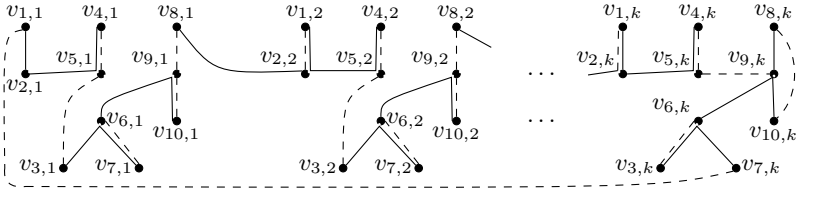


Fig. 8. The modified Eulerian cycle E'. The paths of M' are dashed.

Considering only the edges $\{v_{4,i}, v_{3,i}\}, \{v_{7,i}, v_{9,i}\}, \{v_{8,i}, v_{1,i+1}\}$, for $1 \leq i \leq k-1$, we obtain $\text{cost}(H) \geq 3(k-1)\beta^2$. We have thus shown that, for every $\beta \geq 1$ and arbitrarily small $\varepsilon > 0$, there exists an implementation I of the PMCA such that

$$\frac{\text{cost}(I(G_{10,k}(\beta)))}{\text{Opt}_{\Delta_\beta\text{-TSP}}(G_{10,k}(\beta))} \geq \frac{3(k-1)\beta^2}{2k + 2\beta^2 + 7\beta + 6} \geq \frac{3}{2}\beta^2 - \varepsilon, \tag{1}$$

for sufficiently large k, i.e., we have shown that the upper bound of $\frac{3}{2}\beta^2$ on the approximation ratio of the PMCA is tight.

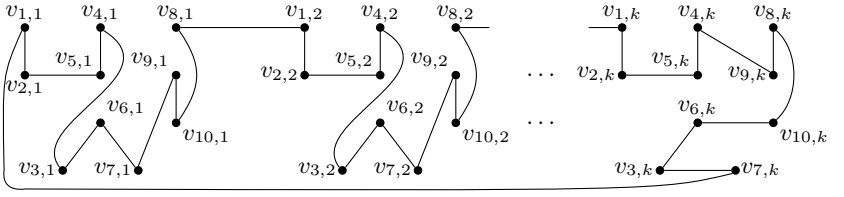


Fig. 9. The Hamiltonian cycle H

3 The PMCA for the Hamiltonian Path Problem

In this section, we analyze the PMCA variant for the β-metric Hamiltonian path problem with $l \in \{0, 1, 2\}$ prespecified endpoints (Δ_β-HPP$_l$). Formally, Δ_β-HPP$_l$ is the following optimization problem. Given a complete graph G with edge costs that satisfy the β-triangle inequality, find a path in G that visits every vertex exactly once, has minimum overall cost and starts and ends in the prespecified endpoints, if any.

We first briefly explain the PMCA-HPP$_l$ (Algorithm 3) and then construct a worst-case example to prove that the approximation ratio of the algorithm cannot be improved for $l = 2$.

The implementation of step 1 is well-known. Step 2 can again be implemented using the distance graph $d(G)$ of G. The algorithm adds $2 - l$ dummy vertices to $d(G)$ with all edges adjacent to them having cost 0 except the edge connecting them that has cost ∞. The algorithm computes a minimum perfect matching for U and the dummy vertices in $d(G)$, removes the edges adjacent to dummy vertices from the matching and maps it back to G by connecting two vertices v, w via a shortest path if they are matched in $d(G)$. After potentially removing an additional edge, the graph contains exactly two odd-degree vertices w, z.

The implementation of step 3 can be found in [8]. Essentially, it does the following. It resolves the conflicts in M for each tree in M separately. If the tree contains z, the algorithm ensures that z is still contained in the resulting forest.

In step 4, we need to distinguish two cases. Let y be the unique neighbor of w towards z in T. If there exists a path $p = (z, \ldots, w)$ in M', we first construct an Eulerian cycle $E = (w, u_1, \ldots, u_{h-1}, w)$ on T and $M' - \{p\}$. We concatenate p and E to obtain the Eulerian path $(z, \ldots, w, u_1, \ldots, u_{h-1}, w)$. If there exists no path $(z, \ldots, w)$, we essentially do the following. We look if there are (unique) paths $p = (z, \ldots, u)$ and $q = (u', \ldots, w)$, and if so, search for an Eulerian path P from u to u' and returns the concatenation of p, P, and q. The detailed implementation of this step can again be found in [8].

Step 5 is implemented in the same way as in the PMCA, except that we do not choose r arbitrarily, but set $r := z$. Step 6 is also implemented in the same way as in the PMCA, except that we first bypass w if it is a conflict.

Theorem 3. *For every $\beta \geq 1$, both the PMCA-HPP$_0$ and the PMCA-HPP$_1$ provide an approximation ratio of $\frac{3}{2}\beta^2$, and there exists an infinite family of*

graphs satisfying the β-triangle inequality on which they cannot achieve a better approximation ratio.

Theorem 4. *For every $\beta \geq 1$, the PMCA-HPP$_2$ provides an approximation ratio of $\frac{5}{3}\beta^2$, and there exists an infinite family of graphs satisfying the β-triangle inequality on which it cannot achieve a better approximation ratio.*

Algorithm 3. PMCA-HPP$_l$

Input: A complete β-metric graph $G = (V, E)$, for some $\beta \geq 1$, and a set $A \subseteq V$ of size l.

1: Find a minimum spanning tree T in G. Let U be the vertices in $V - A$ having odd degree in T plus the vertices in A having even degree in T.
2: Construct a minimum path matching M for U. If necessary, remove an edge from T such that the multigraph formed by T and M has two odd-degree vertices w, z.
3: Resolve conflicts in M in order to obtain a vertex-disjoint path matching M' in which z does not occur as an inner vertex.
4: Construct an Eulerian path $P = (p_1, q_1, p_2, q_2, \ldots)$ on T and M' from w to z such that $p_1, p_2, \ldots$ are paths in T and $q_1, q_2, \ldots$ are paths in M'.
5: Transform $p_1, p_2, \ldots$ into $p'_1, p'_2, \ldots$ such that the forest T_f formed by $p'_1, p'_2, \ldots$ has degree at most 3, w and z are the endpoints of $P' := (p'_1, q_1, p'_2, q_2, \ldots)$ and z is not a conflict in P'.
6: Resolve all remaining conflicts in P' to obtain a Hamiltonian path P''.

Output: P''.

The upper bounds were established in [8]. To prove the second part for $l = 0, 1$, the graph $G_{10,k}(\beta)$ can be reused. The proofs are for the most part quite similar to the one of Theorem 1. For the PMCA-HPP$_1$, the prespecified endpoint is $v_{1,1}$. Both proofs can be found in [12]. To prove the second part for $l = 2$, however, we cannot reuse the graph $G_{10,k}(\beta)$, as we shall see.

Observe that, for $\beta = 1$, there exists an implementation of the PMCA that is also an implementation of the Christofides algorithm. This implementation does not construct a path matching in the second step, but a normal matching.[2] Observe furthermore that, for $\beta = 1$, there exists an implementation of the PMCA-HPP$_2$ that is also an implementation of Hoogeveen's algorithm. As above, it is necessary always to construct a matching instead of a path matching.

Assume now that we could use the graph $G_{10,k}(\beta)$ to establish a lower bound of $\frac{5}{3}\beta^2 - \varepsilon$ on the approximation ratio of the PMCA-HPP$_2$. In particular, we could show an implementation that achieves an approximation ratio of 5/3 on the graph $G_{10,k}(1)$. This, in turn, would contradict the fact that the sets of worst-case instances for the metric TSP and the metric HPP$_2$ are disjoint [13].

To prove the lower bound of $\frac{5}{3}\beta^2 - \varepsilon$, we thus need to introduce a new graph. Let $G_{18,k}(\beta)$ be the complete graph shown in Fig. 10, where all edges not shown have maximum possible cost such that the β-triangle inequality is satisfied.

[2] The condition $\beta = 1$ ensures no path matching is shorter than a minimum matching.

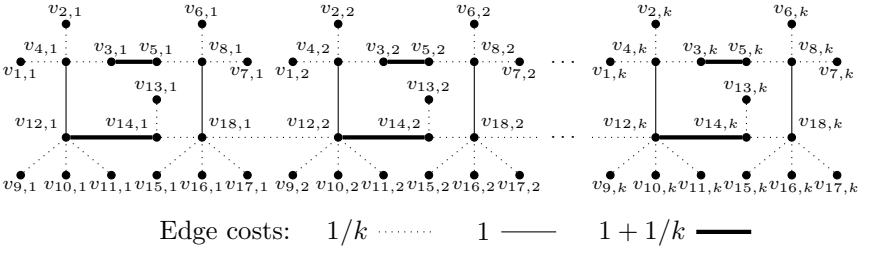


Fig. 10. The graph $G_{18,k}(\beta)$

Lemma 2. *The graph $G_{18,k}(\beta)$ satisfies the β-triangle inequality.*

Proof. The proof is omitted due to space limitations. It can be found in [12]. □

Fig. 11 shows a Hamiltonian path from $v_{9,1}$ to $v_{17,k}$ in $G_{18,k}(\beta)$ of cost $3k+2\beta^2+21\beta+5-(2\beta^2+\beta)/k$. We now show one possible implementation of the PMCA-HPP$_2$ that, on inputs $G_{18,k}(\beta)$ and $A := \{v_{9,1}, v_{17,k}\}$, returns a Hamiltonian path of cost at least $5(k-1)\beta^2$ from $v_{9,1}$ to $v_{17,k}$.

In the first step, the PMCA computes the minimum spanning tree shown in Fig. 12. The vertices in U are circled. In the second step, the PMCA computes the minimum path matching M for U shown in Fig. 13 and then sets $w := v_{17,k}, z := v_{9,1}$.

Theorem 5. *M is a minimum path matching for U.*

Proof. The proof is omitted due to space limitations. It can be found in [12]. □

The goal of the third step is to resolve all conflicts in the minimum path matching M in such a way that $z = v_{9,1}$ is still contained in the resulting vertex-disjoint path matching M'. The PMCA-HPP$_2$ does this for every connected component of M separately. The paths $\{(v_{13,i}, v_{14,i}) \mid 1 \le i \le k\}$ contain no conflicts. Let us therefore now look at the problematic paths of M, i.e.,

$$\begin{aligned}
&\{(v_{1,1}, v_{4,1}, v_{12,1}), (v_{2,1}, v_{4,1}, v_{3,1}), (v_{10,1}, v_{12,1}, v_{11,1})\} \cup \\
&\{(v_{5,i}, v_{8,i}, v_{6,i}), (v_{7,i}, v_{8,i}, v_{18,i}, v_{17,i}), (v_{15,i}, v_{18,i}, v_{16,i}) \mid 1 \le i \le k-1\} \cup \\
&\{(v_{1,i}, v_{4,i}, v_{12,i}, v_{9,i}), (v_{2,i}, v_{4,i}, v_{3,i}), (v_{10,i}, v_{12,i}, v_{11,i}) \mid 2 \le i \le k\} \cup \\
&\{(v_{5,k}, v_{8,k}, v_{6,k}), (v_{7,k}, v_{8,k}, v_{18,k}, v_{15,k}), (v_{16,k}, v_{18,k})\}.
\end{aligned} \tag{2}$$

In the first set, the PMCA-HPP$_2$ bypasses $v_{2,1}$ in the path $(v_{10,1}, v_{12,1}, v_{11,1})$ and $v_{4,1}$ in the path $(v_{1,1}, v_{4,1}, v_{12,1})$. In the second set, the PMCA-HPP$_2$ bypasses $v_{18,i}$ in the path $(v_{15,i}, v_{18,i}, v_{16,i})$ and $v_{8,i}$ in the path $(v_{7,i}, v_{8,i}, v_{18,i}, v_{17,i})$. In the third set, the PMCA-HPP$_2$ bypasses $v_{12,i}$ in the path $(v_{10,i}, v_{12,i}, v_{11,i})$ and $v_{4,i}$ in the path $(v_{1,i}, v_{4,i}, v_{12,i}, v_{9,i})$. In the fourth set, the PMCA-HPP$_2$ transform the two paths $(v_{7,k}, v_{8,k}, v_{18,k}, v_{15,k}), (v_{16,k}, v_{18,k})$ into the paths $(v_{15,k}, v_{16,k}), (v_{7,k}, v_{8,k}, v_{18,k})$ and bypasses $v_{8,k}$ in the latter path. The PMCA-HPP$_2$ thus computes the vertex-disjoint path matching M' shown in Fig. 14.

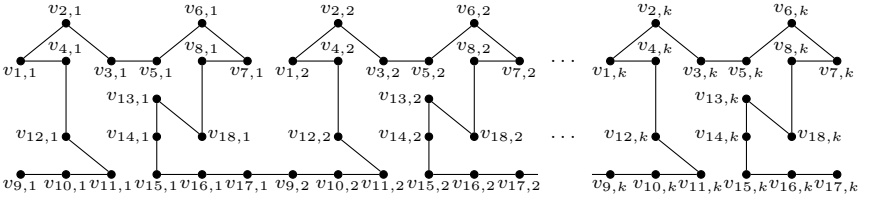


Fig. 11. A Hamiltonian path in $G_{18,k}(\beta)$ of cost $3k + 2\beta^2 + 21\beta + 5 - \frac{2\beta^2+\beta}{k}$

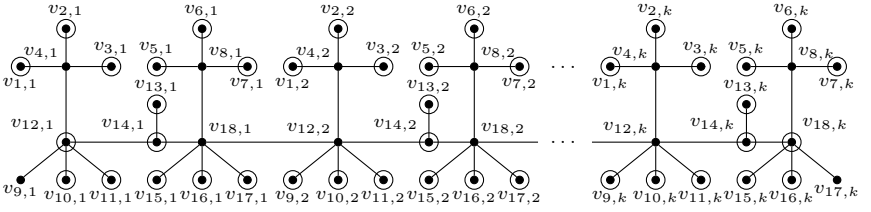


Fig. 12. A minimum spanning tree in $G_{18,k}(\beta)$. The odd-degree vertices are circled.

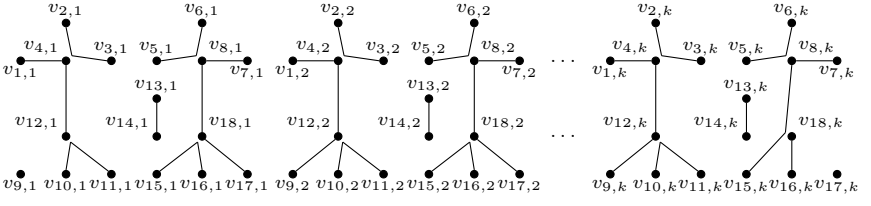


Fig. 13. A minimum path matching for U in $G_{18,k}(\beta)$

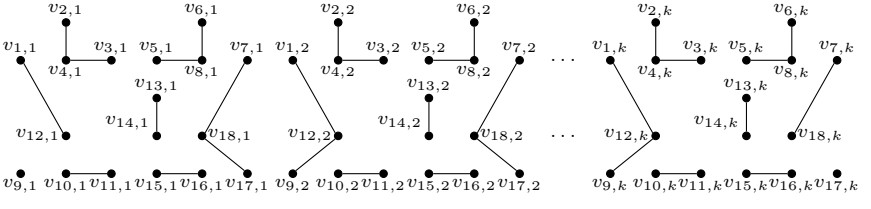


Fig. 14. A vertex-disjoint path matching for U in $G_{18,k}(\beta)$

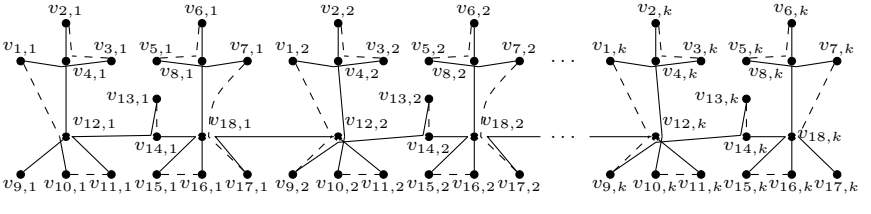


Fig. 15. The Eulerian path P. The paths of M' are dashed.

Now the PMCA-HPP$_2$ computes an Eulerian path from w to z. Because there are no paths in M' having w or z as an endpoint, the algorithm computes an Eulerian path from z to y in $T - \{y, w\}$ and M', where y is the neighbor of w towards z in T, i.e., $v_{18,k}$. Then, it appends the edge $\{y, w\}$ to this path to obtain the Eulerian path P from $v_{9,1}$ to $v_{17,k}$ shown in Fig. 15.

The goal of the fifth step is that every vertex is adjacent to at most three edges from T. The PMCA-HPP$_2$ achieves this by considering every path p in P consisting only of edges from T separately. If the vertex closest to $r := z = v_{9,1}$ in p is internal to p, it is bypassed. Therefore, it bypasses $v_{4,i}$ between $v_{3,i}$ and $v_{1,i}$ and $v_{8,i}$ between $v_{5,i}$ and $v_{7,i}$, for $1 \leq i \leq k, v_{12,1}$ between $v_{11,1}$ and $v_{14,1}, v_{12,i}$ between $v_{9,i}$ and $v_{14,i}$ and between $v_{10,i}$ and $v_{4,i}$, for $2 \leq i \leq k, v_{18,i}$ between $v_{16,i}$ and $v_{8,i}$ and between $v_{17,i}$ and $v_{12,i+1}$, for $1 \leq i \leq k-1$, and finally $v_{18,k}$ between $v_{16,k}$ and $v_{8,k}$. This results in the Eulerian path P' shown in Fig. 16.

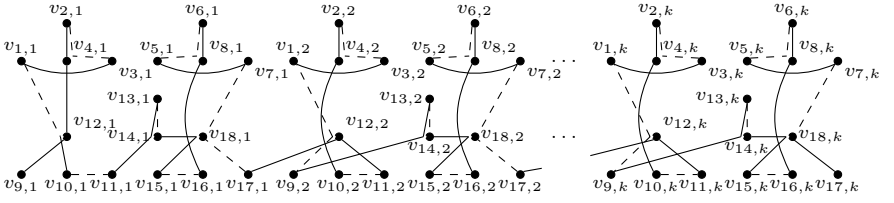


Fig. 16. The modified Eulerian path P'. The paths of M' are dashed.

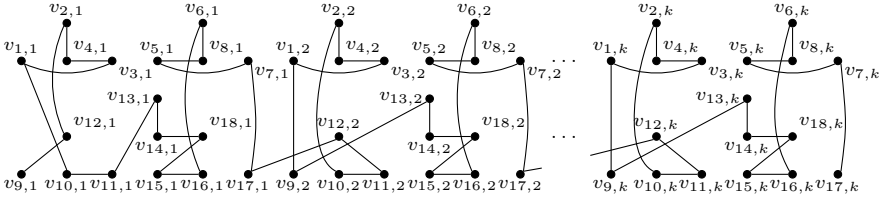


Fig. 17. The Hamiltonian path P''

The goal of the last step is that every vertex except $v_{9,1}$ and $v_{17,k}$ has degree 2. The problematic vertices are thus all the vertices $v_{4,i}, v_{8,i}, v_{12,i}, v_{14,i}, v_{18,i}$, for $1 \leq i \leq k$. Because $w = v_{17,k}$ is not a conflict in P', the PMCA-HPP$_2$ starts with the resolution of an arbitrary conflict. It obtains the Hamiltonian path P'' shown in Fig. 17 by bypassing

- $v_{4,1}$ between $v_{12,1}$ and $v_{2,1}$, and every other $v_{4,i}$ between $v_{10,i}$ and $v_{2,i}$,
- $v_{8,i}$ between $v_{16,i}$ and $v_{6,i}$, for $1 \leq i \leq k$,
- $v_{12,1}$ between $v_{1,1}$ and $v_{10,1}$, and every other $v_{12,i}$ between $v_{1,i}$ and $v_{9,i}$,
- $v_{14,1}$ between $v_{11,1}$ and $v_{13,1}$, and every other $v_{14,i}$ between $v_{9,i}$ and $v_{13,i}$,
- $v_{18,i}$ between $v_{7,i}$ and $v_{17,i}$, for $1 \leq i \leq k$.

Considering only the edges $\{v_{1,i}, v_{9,i}\}, \{v_{2,i}, v_{10,i}\}, \{v_{6,i}, v_{16,i}\}, \{v_{7,i}, v_{17,i}\}, \{v_{9,i}, v_{13,i}\}$, for $2 \leq i \leq k$, we obtain $\text{cost}(P'') \geq 5(k-1)\beta^2$. We have thus shown that, for every $\beta \geq 1$ and arbitrarily small $\varepsilon > 0$, there exists an implementation I of the PMCA-HPP$_2$ such that

$$\frac{\text{cost}(I(G_{18,k}(\beta)))}{\text{Opt}_{\Delta_\beta-\text{HPP}_2}(G_{18,k}(\beta))} \geq \frac{5(k-1)\beta^2}{3k + 2\beta^2 + 21\beta + 5 - \frac{2\beta^2+\beta}{k}} \geq \frac{5}{3}\beta^2 - \varepsilon, \qquad (3)$$

for sufficiently large k, i.e., we have shown that the upper bound of $\frac{5}{3}\beta^2$ on the approximation ratio of the PMCA-HPP$_2$ is tight.

References

1. Andreae, T.: On the Traveling Salesman Problem Restricted to Inputs Satisfying a Relaxed Triangle Inequality. Networks 38, 59–67 (2001)
2. Bender, M.A., Chekuri, C.: Performance guarantees for the TSP with a parameterized triangle inequality. Inf. Proc. Letters 73, 17–21 (2000)
3. Böckenhauer, H.-J., Hromkovič, J.: Stability of approximation algorithms or parameterization of the approximation ratio. In: Proceedings of the 9th International Symposium on Operations Research in Slovenia, pp. 23–28 (2007)
4. Böckenhauer, H.-J., Hromkovič, J., Klasing, R., Seibert, S., Unger, W.: Towards the notion of stability of approximation for hard optimization tasks and the traveling salesman problem. Theoretical Computer Science 285, 3–24 (2002)
5. Böckenhauer, H.-J., Hromkovič, J., Seibert, S.: Stability of Approximation. In: Gonzalez, T.F. (ed.) Handbook of Approximation Algorithms and Metaheuristics. Chapman & Hall, Boca Raton (2007)
6. Christofides, N.: Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388. Carnegie Mellon University, Graduate School of Industrial Administration (1976)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge (2009)
8. Forlizzi, L., Hromkovič, J., Proietti, G., Seibert, S.: On the Stability of Approximation for Hamiltonian Path Problems. Alg. Oper. Res. 1, 31–45 (2006)
9. Goodaire, E.G., Parmenter, M.M.: Discrete Mathematics with Graph Theory. Prentice Hall, Upper Saddle River (2005)
10. Hoogeveen, J.A.: Analysis of Christofides' heuristic: Some paths are more difficult than cycles. Oper. Res. Letters 10, 291–295 (1991)
11. Hromkovič, J.: Algorithmics for Hard Problems. Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics. Springer, Heidelberg (2004)
12. Krug, S.: Analysis of Approximation Algorithms for the Traveling Salesman Problem in Near-Metric Graphs. Master's thesis, ETH Zurich, Department of Computer Science (2011)
13. Mömke, T.: Structural Properties of Hard Metric TSP Inputs. In: Černá, I., Gyimóthy, T., Hromkovič, J., Jefferey, K.G., Králović, R., Vukolić, M., Wolf, S. (eds.) SOFSEM 2011. LNCS, vol. 6543, pp. 394–405. Springer, Heidelberg (2011)

Parikh's Theorem and Descriptional Complexity

Giovanna J. Lavado and Giovanni Pighizzini

Dipartimento di Informatica e Comunicazione
Università degli Studi di Milano
via Comelico 39, I-20135 Milano, Italy
giovanna.lavado@unimi.it,
pighizzini@dico.unimi.it

Abstract. It is well known that for each context-free language there exists a regular language with the same Parikh image. We investigate this result from a descriptional complexity point of view, by proving tight bounds for the size of deterministic automata accepting regular languages Parikh equivalent to some kinds of context-free languages. First, we prove that for each context-free grammar in Chomsky normal form with a fixed terminal alphabet and h variables, generating a bounded language L, there exists a deterministic automaton with at most $2^{h^{O(1)}}$ states accepting a regular language Parikh equivalent to L. This bound, which generalizes a previous result for languages defined over a one letter alphabet, is optimal. Subsequently, we consider the case of arbitrary context-free languages defined over a two letter alphabet. Even in this case we are able to obtain a similar bound. For alphabets of at least three letters the best known upper bound is a double exponential in h.

Keywords: finite automata, formal languages, context-free languages, descriptional complexity, Parikh's theorem, bounded languages.

1 Introduction

Parikh's Theorem is a classical result in formal language theory [1]. With each string x over an alphabet of m symbols, the m integer vector counting the number of occurrences in x of each alphabet symbol (the *Parikh image* of x) is associated. The *Parikh image of a language* L is the set of all Parikh images of strings in L. Parikh's Theorem states that the Parikh image of a context-free language L is a semilinear set or, equivalently, that there exists a regular language R with the same Parikh image of L (L and R will be also said to be Parikh equivalent).

This classical result has been extensively investigated in the literature (e.g., [2,3,4]) even for the connections of semilinear sets with other fields of investigation (e.g., Presburger Arithmetics [5], Petri Nets [6], logical formulas [7]).

Two recent papers present new interesting contributions to these researches, in particular by investigating complexity aspects of Parikh's Theorem.

In [8] the authors obtain normal form theorems for Parikh images of regular and context-free languages. In particular, in the case of regular languages and in the case of context-free languages over a two letter alphabet they provide tight

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 361–372, 2012.

bounds on the size of the semilinear representations of Parikh images. They also present some applications of these results in different fields.

In [9] the authors give a new proof of Parikh's Theorem, which provides a construction of a *nondeterministic* automaton A accepting a language Parikh equivalent to the language specified by a context-free grammar G. This construction is also interesting because the number of the states of the resulting automaton A is exponential in the size of the grammar G, thus significantly improving the upper bounds that can be obtained from the classical constructions [1,2]. Furthermore, this bound cannot be further improved.

It is natural to extend these investigations in order to discover how many states a *deterministic* automaton accepting a regular language Parikh equivalent to the language generated by a context-free grammar of size n needs. Applying the classical subset construction to the nondeterministic automaton obtained in [9], an upper bound which is double exponential in n can be immediately obtained. However, we do not know if this is optimal.

In the case of languages defined over a one letter alphabet, also called *unary languages*, this problem has been solved in [10]. (We remind the reader that unary context-free languages are regular [11].) The authors proved that for each context-free grammar in Chomsky normal form with h variables generating a unary language there exists an equivalent nondeterministic automaton with at most $2^{2h-1}+1$ states and an equivalent deterministic automaton with less than 2^{h^2} states. Both these bounds are optimal.

The main result of this paper is an extension of these bounds to letter bounded context-free languages, i.e., subsets of $a_1^* a_2^* \cdots a_m^*$, where $a_1, a_2, \ldots, a_m$ are pairwise different symbols [13,14]. Fixed an alphabet of m symbols, we show that given a letter bounded context-free language described by a grammar of size n we can obtain a deterministic automaton with a number of states exponential in a polynomial in n, accepting a Parikh equivalent language. As a consequence of the results in the unary case, this bound cannot be improved.

We also attack the problem in the general case (i.e., we remove the restriction to bounded languages). We prove a similar result in the case of languages defined over a binary alphabet, while we leave as an open problem the generalization to alphabets of cardinality greater than two.

2 Preliminaries

Given a set S, $\#S$ denotes its cardinality and 2^S the family of all its subsets. Let Σ^* denote the set of all strings over the finite alphabet Σ, with the empty string denoted by ε. For the sake of simplicity, all the languages we consider do not contain ε. Given a string $w \in \Sigma^*$, $|w|$ denotes its length and $|w|_a$ the number of occurrences in w of the symbol $a \in \Sigma$.

Given the alphabet $\Sigma = \{a_1, a_2, \ldots, a_m\}$, the *Parikh map* $\psi : \Sigma^* \to \mathbb{N}^m$ associates with each word $w \in \Sigma^*$ the vector $\psi(w) = (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_m})$ and with each language $L \subseteq \Sigma^*$ the set $\psi(L) = \{\psi(w) \mid w \in L\}$ (the *Parikh image* of w and L, respectively). Two strings $w', w'' \in \Sigma^*$ (languages $L', L'' \subseteq$

Σ^*, resp.) are said to be *Parikh equivalent* when $\psi(w') = \psi(w'')$ ($\psi(L') = \psi(L'')$, resp.). In this case we write $w' =_\pi w''$ ($L' =_\pi L''$, resp.).

For the notions of deterministic and nondeterministic automaton (DFA and NFA, resp.), context-free grammar and context-free language (CFG and CFL), context-free grammar in Chomsky normal form (CNFG), derivation, derivation tree, and for the corresponding notations, we refer the reader to [15]. As in [10] and according to the discussion in [12], we use the number of variables of CNFGs as a "reasonable" measure of descriptional complexity for CFLs. With some abuse of language, given a node of a derivation tree labeled with a variable A, we write "the node A" instead of "the node with label A", if this does not introduce any ambiguity. The notation $A \stackrel{\star}{\Rightarrow}_\pi w$, where $w \in \Sigma^*$, indicates that from A it is possible to derive a string which is Parikh equivalent to w.

Parikh's theorem [1] states that the Parikh image of each context-free language is a semilinear set. The following result is an immediate consequence.

Theorem 1. *Each context-free language is Parikh equivalent to a regular language.*

A language L is said to be *letter bounded* (bounded, for short) if $L \subseteq a_1^* a_2^* \cdots a_m^*$, where $a_1, a_2, \ldots, a_m$ are pairwise different symbols [13].

Given a CNFG $G = (V, \Sigma, P, S)$ generating a subset of $a_1^* \cdots a_m^*$, without loss of generality we can suppose that each variable of G is *useful*, i.e., for each $A \in V$, there are terminal strings u, v, w, such that $S \stackrel{\star}{\Rightarrow} uAw \stackrel{\star}{\Rightarrow} uvw$. According to the discussion in [14], it is not difficult to prove that with each variable $A \in V$, a pair of indices l_A, r_A, $1 \leq l_A \leq r_A \leq m$, can be associated in such a way that $A \stackrel{\star}{\Rightarrow} w \in \Sigma^*$ implies that $w \in a_{l_A}^* \cdots a_{r_A}^*$ and $(A \to BC) \in P$ implies that $l_A \leq l_B \leq r_B \leq l_C \leq r_C \leq r_A$. If $A \stackrel{\star}{\Rightarrow} w$ also implies that the first symbol of w is a_{l_A} and the last symbol of w is a_{r_A}, i.e., $w \in a_{l_A}\Sigma^* \cap \Sigma^* a_{r_A}$, then G is said to be *strongly bounded*.

In a strongly bounded CNFG G, a variable A is said to be *unary* if $l_A = r_A$, otherwise A is *nonunary*. We notice that if there is a production of the form $A \to a$, $a \in \Sigma$, then the variable A is unary and, on the other hand, if there is a production $A \to BC$ then $l_A = l_B$ and $r_C = r_A$. If G is not strongly bounded, then we can express the language $L(G)$ as the union of the $m(m+1)/2$ languages $L_{ij} = L(G) \cap a_i\Sigma^* \cap \Sigma^* a_j$, $1 \leq i \leq j \leq m$. We now observe that each of these languages is generated by a strongly bounded CNFG and, furthermore, the total number of variables used by such grammars is $hm(m+1)/2$. To this aim, we consider the set of variables $V' = \{(A, i, j) \mid A \in V, 1 \leq i \leq j \leq m\}$ and the set of productions P' containing:

- $(A, i, j) \to (B, i, k')(C, k'', j)$, for each production $A \to BC$ in P, and $1 \leq i \leq k' \leq k'' \leq j \leq m$,
- $(A, i, i) \to a_i$, for each production $A \to a_i$ in P.

It is possible to verify that using the set P' of productions, $(A, i, j) \stackrel{\star}{\Rightarrow} w$ if and only if $A \stackrel{\star}{\Rightarrow} w$ in G and $w \in a_i\Sigma^* \cap \Sigma^* a_j$. Furthermore, since the original grammar G defines a bounded language, $w \in a_i^* \cdots a_j^*$. As a consequence, the grammar $G_{ij} = (V', \Sigma, P', (S, i, j))$, $1 \leq i \leq j \leq n$, generates the language L_{ij}.

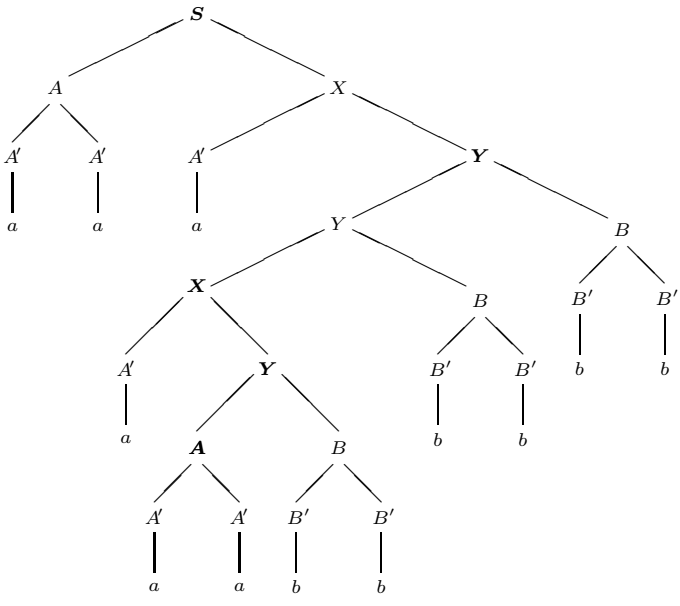


Fig. 1. Example with $\Sigma = \{a, b\}$

3 The Bounded Case

In this section we present our main result, namely we prove that for each bounded CFL L specified by a CNFG with h variables there exists a regular language R which is Parikh equivalent to L and it is accepted by a DFA M with $2^{h^{O(1)}}$ states.

Since the proof involves many technical details, we firstly give an informal explanation. We describe how the language R is chosen and how the DFA M accepting R operates.

Let us start by considering the case of a strongly bounded grammar with a terminal alphabet consisting of two symbols $a_1 = a$ and $a_2 = b$. Given a production $A \to BC$, if A is nonunary then at most one variable between B and C can be nonunary. Furthermore, each sentential form derived from A can contain at most one nonunary variable. On the other hand, if A is unary then both B and C must be unary.

Suppose the grammar produces the tree in Figure 1. The variables A, A', B, B' are unary, while S, X, Y are nonunary. The tree derives the string $w = a^6b^6$.

The DFA M given by our construction will simulate the derivation process, expanding unary variables as soon as possible, and verifying the matching between terminal symbols so derived and the symbols in the input string. According to this strategy, the first part of the derivation described by the tree in Figure 1 is

$$S \Rightarrow AX \stackrel{\star}{\Rightarrow} a^2X \Rightarrow a^2A'Y \Rightarrow a^3Y \Rightarrow a^3YB \stackrel{\star}{\Rightarrow} a^3Yb^2 \Rightarrow \dots$$

Hence, the string recognized in this way starts with a^3b^2. In particular, the DFA M will accept the string $w_\pi = a^3b^4ab^2a^2$ which is clearly Parikh equivalent to w.

However, the derivation process is strictly nondeterministic, while our purpose is to obtain a deterministic automaton. To achieve this goal, the recognition process is "driven" by the structure of the input string. In particular, the computation of M is a sequence of phases, each one of them scanning a longest input factor consisting of occurrences of a same symbol. During a phase, M simulates a suitable finite automaton.

The DFA M keeps in each state q a set V_q of variables of the grammar. Suppose that M from q starts a phase having a^t, $t > 0$, as next longest input factor consisting of the same symbol. Then M computes the set of variables that can be "reached" from variables in V_q by reading a^t, namely $V_{q'} = \{Y \in V \mid X \stackrel{\star}{\Rightarrow} a^tY, X \in V_q\}$. This set will be stored in the new state q'. Since unary CFLs are regular [11], this computation can be performed using a finite state control. In a similar way the automaton M can manage factors consisting of occurrences only of the letter b.

In the initial state M remembers only the variable S. Then it starts to operate on the longest input prefix consisting of a same symbol. In our example with input $w_\pi = a^3b^4ab^2a^2$, this prefix is a^3. M simulates a DFA M_S that on input a^t computes the set $\Lambda_S(a^t)$ consisting of all variables Z such that $S \stackrel{\star}{\Rightarrow} a^tZ$. According to the tree in the picture, we can observe that $Y \in \Lambda_S(a^3)$. Now, the next input symbol is b. The automaton M applies a similar procedure for the factor b^4, in particular, simulating a DFA M_Y, associated with the variable Y, it computes the set of variables $\Lambda_Y(b^4) = \{Z \mid Y \stackrel{\star}{\Rightarrow} Zb^4\}$. Using this procedure on the string $w_\pi = a^3b^4ab^2a^2$, the following steps lead to the acceptance:

$$Y \in \Lambda_S(a^3), X \in \Lambda_Y(b^4), Y \in \Lambda_X(a), A \in \Lambda_Y(b^2), \text{ and finally } \varepsilon \in \Lambda_A(a^2).$$

Notice that the last step starts from the unary variable A. In this case the function $\Lambda_A(a^t)$ returns ε if and only if a^t can be derived from A, reaching in this way the end of the derivation. The corresponding steps in the derivation are as follows (in Figure 1 the variables considered in these steps are represented in boldface):

$$S \stackrel{\star}{\Rightarrow} a^3Y \stackrel{\star}{\Rightarrow} a^3Xb^4 \stackrel{\star}{\Rightarrow} a^4Yb^4 \stackrel{\star}{\Rightarrow} a^4Ab^6 \stackrel{\star}{\Rightarrow} a^6b^b.$$

In the example we present only a successful path, but the automaton M will examine in parallel different paths, by keeping in its state a set of variables.

Why we choose $w_\pi = a^3b^4ab^2a^2$ as representative of w and not another string, as for instance $w' = a^3b^4a^3b^2$?

In the derivation process, eventually a nonunary variable whose both sons in the tree are unary will be found (in the previous derivation the variable Y in the sentential form a^4Yb^4). In this situation *we always firstly choose to expand the variable on the right*, remembering the other one in the state, to be expanded later. This choice turns out to be useful for alphabets with more than two letters.

Consider now the tree (generated by a different strongly bounded grammar) in Figure 2. In this case the terminal alphabet is $\{a, b, c\}$. It is possible to observe

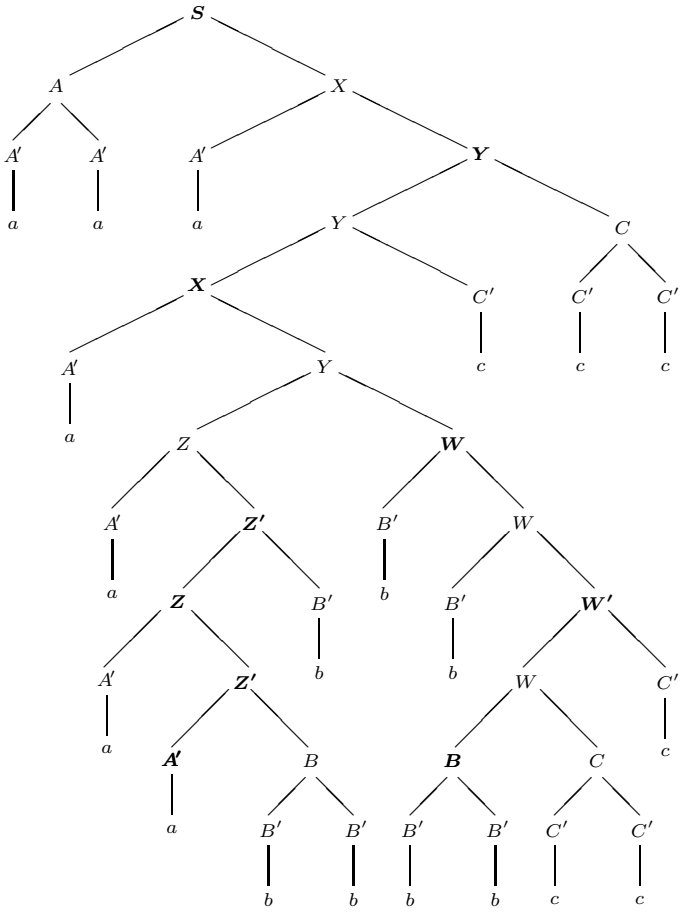


Fig. 2. Example with $\Sigma = \{a, b, c\}$

that, with this alphabet, each sentential form which is derivable from a nonunary variable can contain at most 2 nonunary variables.

The string produced by the tree is $w = a^7b^7c^6$. The Parikh equivalent string recognized by our automaton M is $w_\pi = a^3c^3a^2bab^2ab^2c^3b^2$. In this case:

$$Y \in \Lambda_S(a^3), X \in \Lambda_Y(c^3), Z'W \in \Lambda_X(a^2), Z \in \Lambda_{Z'}(b), Z' \in \Lambda_Z(a),$$
$$A' \in \Lambda_{Z'}(b^2), \varepsilon \in \Lambda_{A'}(a), W' \in \Lambda_W(b^2), B \in \Lambda_{W'}(c^3), \varepsilon \in \Lambda_B(b^2).$$

Notice that $\Lambda_X(a^2)$ gives a sequence of two nonunary variables due the fact that in the right son of the node with label X (the second node with label X from the top) the production $Y \to ZW$ is applied. With this production, the nonunary variable Y, generating a subset of $a^*b^*c^*$ is "split" in Z and W that can generate, respectively, subsets of a^*b^* and b^*c^*.

The two subtrees produce the strings $z' = a^3b^3$ and $z'' = b^4c^3$, whose concatenation is $z = a^3b^7c^3$. Reading z as input, a finite automaton cannot have any information about which prefix and suffix should be associated to the left and to the right subtree. Hence, the factor b^7 should be split, introducing some kind of nondeterminism, between the two subtrees. This problem is avoided using the strategy above described. The sons of the node with label Z' closest to the leaves are unary variables: the factor produced by the rightmost one is firstly chosen. In particular, with our strategy the strings corresponding to the subtrees rooted Z and W we will consider are $z'_\pi = abab^2a =_\pi a^3b^3$ and $z''_\pi = b^2c^3b^2 =_\pi b^4c^3$. In this way, the last symbol of z'_π cannot occur in the string z''_π and, in general, an input factor consisting of a same symbol cannot be split over two independent subtrees. This allows us to avoid of dealing with a possible source of nondeterminism.

Now, we start to present the proof of our main result. To this aim, from now on we fix a strongly bounded CNFG G with h variables, generating a subset of $a_1^* \cdots a_m^*$. We need some preliminary notions and lemmas.

A sequence of variables $A_1 \cdots A_k \in V^*$ is said to be *valid* when:

- $l_{A_1} \leq r_{A_1} \leq l_{A_2} \leq r_{A_2} \leq \cdots \leq l_{A_k} \leq r_{A_k}$, and
- if $1 \leq j < k$ and A_j is unary then $r_{A_j} < l_{A_{j+1}}$.

The set of all valid sequences will be denoted as Υ. The set of valid sequences *without unary variables* is denoted by Υ_0. The empty sequence from Υ is denoted as the empty word by ε. Notice that the length of a valid sequence is at most m. Hence, the cardinality of Υ is a polynomial in h.

Lemma 1. *Let $A \stackrel{\star}{\Rightarrow} w$ be a derivation in G of a string $w \in \Sigma^*$ from a* nonunary *variable A. Then:*

(i) $A \stackrel{\star}{\Rightarrow} a_{l_A}^t A_1 \cdots A_k \stackrel{\star}{\Rightarrow} w$, for some $t \geq 0$, $k > 0$, $A_1 \cdots A_k \in \Upsilon_0$, such that:
- *either the right son of A_1 is unary,*
- *or $t > 0$ and $l_A < l_{A_1}$.*

(ii) $A \stackrel{\star}{\Rightarrow} A_1 \cdots A_k a_{r_A}^t \stackrel{\star}{\Rightarrow} w$, for some $t \geq 0$, $k > 0$, $A_1 \cdots A_k \in \Upsilon$, such that:
- *if $k > 1$ then $A_1, \ldots, A_{k-1}$ are nonunary,*
- *either A_k and its right son are nonunary,*
- *or $t > 0$ and $r_{A_k} < r_A$.*

For each variable $A \in V$, we now introduce a function $\Lambda_A : a_{l_A}^* \cup a_{r_A}^* \to 2^\Upsilon$ such that for each $t \geq 0$:

- if A is unary, i.e., $l_A = r_A$, then $\Lambda_A(a_{l_A}^t) = \begin{cases} \{\varepsilon\} & \text{if } A \stackrel{\star}{\Rightarrow} a_{l_A}^t \\ \emptyset & \text{otherwise} \end{cases}$
- if A is nonunary then:
  - $A_1 \cdots A_k \in \Lambda_A(a_{l_A}^t)$ if and only if $A_1 \cdots A_k \in \Upsilon_0$ and $A \stackrel{\star}{\Rightarrow} a_{l_A}^t A_1 \cdots A_k$,
  - $A_1 \cdots A_k \in \Lambda_A(a_{r_A}^t)$ if and only if $A_1 \cdots A_k \in \Upsilon$ and $A \stackrel{\star}{\Rightarrow} A_1 \cdots A_k a_{r_A}^t$ and either A_k is nonunary or $r_{A_k} < r_A$.

In the following, we say that a function $f : \Sigma^* \to \Gamma$, where Γ is a finite set, is *computable by an automaton*, if there exists a DFA M such that all the strings leading from the initial state to a same state have the same image by f. In other words, we can associate an output with each state of M (as in classical Moore's machines) such that the value of $f(x)$ is the output associated with the state reached on x.

Lemma 2. *The function Λ_A can be computed by a* DFA *M_A with $2^{h^{O(1)}}$ states.*

Proof. (outline) In [10, Theorem 6] it is proved that for each unary CNFG with $h \geq 2$ variables there exists an equivalent DFA with less than 2^{h^2} states. Given a unary variable A, the DFA M_A can be obtained by applying that construction to the grammar obtained from G by replacing the start symbol with the variable A and by restricting to the symbols which are reachable from A.

The same argument can be modified to manage the case of a nonunary variable A. To this aim, it is useful an extension of the Ogden Lemma [16,17]: sufficiently long strings can be pumped after fixing some "excluded" positions that are not used to pump. This allows us to generalize Lemma 2 from [10], by considering all derivations of the form $A \stackrel{\star}{\Rightarrow} a^t_{l_A} A_1 \cdots A_k$ or $A \stackrel{\star}{\Rightarrow} A_1 \cdots A_k a^t_{r_A}$, *for fixed variables* $A_1 \cdots A_k$.

With such a change, it is possible to extend Theorem 6 from [10], by proving the existence of a DFA $M_{A,A_1\cdots A_k}$ with $2^{h^{O(1)}}$ states which accepts $a^t_{l_A}$ if and only if $A \stackrel{\star}{\Rightarrow} a^t_{l_A} A_1 \cdots A_k$ and which accepts $a^t_{r_A}$ if and only if $A \stackrel{\star}{\Rightarrow} A_1 \cdots A_k a^t_{r_A}$.

The DFA M_A simulates in parallel the DFAs $M_{A,A_1\cdots A_k}$, for all $A_1 \cdots A_k \in \Upsilon$. Hence M_A can be implemented using $(2^{h^{O(1)}})^{\#\Upsilon}$ many states. Since $\#\Upsilon$ is bounded by a polynomial in h, the number of states of M_A is $2^{h^{O(1)}}$. □

We now extend the function Λ in order to consider valid sequences of variables and unary strings. For $\alpha \in \Upsilon$, $t \geq 0$, $i = 1, \ldots, m$, we define:

$$\Lambda(\alpha, a^t_i) = \{B_1 \cdots B_{j-1} \gamma B_{j+1} \cdots B_s \mid \alpha \stackrel{\star}{\Rightarrow} B_1 \cdots B_s \in \Upsilon, \\ l_{B_j} = i \text{ or } r_{B_j} = i, j > 1 \text{ implies } r_{B_{j-1}} < i, \text{and } \gamma \in \Lambda_{B_j}(a^t_i)\}.$$

The function Λ describes some sentential forms containing variables and the factor a^t_i and which are derivable from the valid sequence α.

- First, from α we can derive another valid sequence $B_1 \cdots B_s$, without generating any terminal symbol.
- Then, we try to derive the string a^t_i from one of the variables in the sequence, using the following rules:
  - First we check if there is a position j where we can try to derive the string a^t_i with a *rightmost derivation* of the form $B_j \stackrel{\star}{\Rightarrow} A_1 \cdots A_k a^t_i$, namely we check if $r_{B_j} = i$. If this is the case, then we compute the possible sequences $\gamma = A_1 \cdots A_k$ by using the function Λ_{B_j} (i.e., the DFA M_{B_j}).
  - Otherwise ($r_{B_{j'}} \neq i$ for $j' = 1, \ldots, s$) we check if there is a position j where the string a^t_i could be derived with a *leftmost derivation* of the

form $B_j \stackrel{\star}{\Rightarrow} a_i^t A_1 \cdots A_k$, namely we check if $l_{B_j} = i$. We notice that $j > 1$ implies that $r_{B_{j-1}} < i$, otherwise the previous case could be applied starting from the variable B_{j-1}. Even in this case we then use the function Λ_{B_j}.

- If even in the last case the answer is negative, i.e., i is different from all l_{B_j}'s and r_{B_j}'s, then from $B_1 \cdots B_s$ we do not add any contribution to $\Lambda(\alpha, a_i^t)$.

Lemma 3. *Let $\alpha \in \Upsilon$, $t \geq 0$, $i = 1, \ldots, m$. Then $\Lambda(\alpha, a_i^t)$ is a subset of Υ.*

We now extend the domain of Λ to consider strings over Σ^*, by defining, for $\alpha \in \Upsilon$, $w \in \Sigma^*$:

$$\Lambda(\alpha, w) = \begin{cases} \alpha & \text{if } w = \varepsilon \\ \bigcup_{\beta \in \Lambda(\alpha, w')} \Lambda(\beta, a_i^t) & \text{if } w = w' a_i^t \text{ and } a_i^t \text{ is the longest suffix} \\ & \quad \text{of } w \text{ consisting of a same symbol.} \end{cases}$$

Lemma 4. *Let $\alpha, \alpha', \beta \in \Upsilon$, $A, A' \in V$, and $w, w' \in \Sigma^*$, such that the last symbol of w and the first symbol of w' are different.*

(i) If $\beta \in \Lambda(\alpha, w)$ and $\alpha\alpha' \in \Upsilon$ then $\beta\alpha' \in \Lambda(\alpha\alpha', w)$.
(ii) If $A'\alpha \in \Lambda(A, w)$ and $\varepsilon \in \Lambda(A', w')$, then $\alpha \in \Lambda(A, ww')$.
(iii) If $\varepsilon \in \Lambda(A, w)$ then $A \stackrel{\star}{\Rightarrow}_\pi w$.

The next lemma is fundamental to obtain our main result. It states that for each string w derivable from a variable A, we can find a Parikh equivalent string w_π, which, in some sense, can be "recognized" by making use of the function Λ.

Lemma 5. *Let $A \stackrel{\star}{\Rightarrow} w$ be a derivation in G. Then there exists $w_\pi \in \Sigma^*$ such that $w_\pi =_\pi w$ and $\varepsilon \in \Lambda(A, w_\pi)$. Furthemore:*

- *If A is nonunary then the last symbol of w_π is different from a_{r_A}.*
- *If both A and its right son are nonunary then even the first symbol of w_π is different from a_{r_A}.*
- *If A is nonunary and its right son is unary then the first symbol of w_π is different from a_{l_A}.*

At this point we are able to define the regular language we are interested in, and to prove that it can be accepted by a DFA with a number of states exponential in a polynomial in h, the number of variables of the grammar G. The language is defined as:

$$R(G) = \{x \in \Sigma^* \mid \varepsilon \in \Lambda(S, x)\}.$$

As an immediate consequence of Lemma 4(iii) and of Lemma 5, we get:

Theorem 2. $L(G) =_\pi R(G)$.

Now we give an outline of the construction of a DFA M with $2^{h^{O(1)}}$ states accepting the language $R = R(G)$.

The main idea is to remember in each state a subset of valid sequences in such a way that the subset associated with the state reached from the initial state by reading a string w is $\Lambda(S, w)$. Hence, in the light of the definition and of the properties of Λ, a string w belongs to R if and only if the subset of Υ associated with the state reached on w contains the empty sequence. So:

- The subset of Υ associated with the initial state is $\{S\}$.
- A state is final if and only if the subset associated with it contains ε.

Now, we describe how the transition function works. Consider a state q and let $\mathcal{A} \subseteq \Upsilon$ be the subset associated with it. Suppose that M in q starts to read an input factor a_i^t consisting of occurrences of a same input symbol and that either the symbol which has been read to reach q is other than a_i, or q is the initial state.

Starting from q and reading a_i^t, our goal is to reach a state q' having as associated subset $\mathcal{A}' = \bigcup_{\alpha \in \mathcal{A}} \Lambda(\alpha, a_i^t)$. To do that:

- For each $\alpha \in \mathcal{A}$, M considers all $B_1 \cdots B_s \in \Upsilon$ such that $\alpha \stackrel{\star}{\Rightarrow} B_1 \cdots B_s$. Notice that given α the possible $B_1 \cdots B_s$ are fixed.
- For each $B_1 \cdots B_s$, M selects a suitable B_j, if any, according to the definition of $\Lambda(\alpha, a_i^t)$. Given $B_1 \cdots B_s$ this choice depends only on i, hence it can be encoded in the transition function of M. (If a suitable B_j does not exist $B_1 \cdots B_s$ does not contribute to $\Lambda(\alpha, a_i^t)$, for each $t > 0$, hence it can be forgotten.)
- M, by keeping this information, simulates the DFA M_{B_j}. In this way, after reading a_i^t, knowing $B_1 \cdots B_s$ and the state reached by M_{B_j}, M can reconstruct the contribution to $\mathcal{A}'$ derived from $B_1 \cdots B_s$.

This strategy can be implemented by remembering in each state of M a set $\mathcal{A}$ of valid sequences with states of some of M_{B_j}'s. Since different $\alpha \in \mathcal{A}$ and $B_1 \cdots B_s$ must be considered, we may need to simulate in parallel different DFAs M_{B_j}. In the worst case we have to simulate one DFA for each variable, hence h DFAs. By Lemma 2 each one of them has $2^{h^{O(1)}}$ states. Hence, running all of them in parallel can be done within $2^{h^{O(1)}}$ states.

Furthermore, the cardinality of Υ is bounded by a polynomial in h, hence the number of possible subsets of Υ is $2^{h^{O(1)}}$. This permit us to conclude that the DFA M can be implemented with $2^{h^{O(1)}}$ states.

We can now prove our main result.

Theorem 3. *Let $\Sigma = \{a_1, a_2, \ldots, a_m\}$. For each* CNFG *with h variables generating a bounded language $L \subseteq a_1^* a_2^* \cdots a_m^*$, there exists a* DFA *with at most $2^{h^{O(1)}}$ states accepting a regular language Parikh equivalent to L.*

Proof. If G is strictly bounded then the result is an immediate consequence of Theorem 2 and of the above outlined construction. Otherwise, we can apply the construction given in Section 2, to get from G a set V' of $O(h)$ variables, a set P' of productions and $m(m+1)/2$ grammars using V' and P' and differing only

in the start symbol, such that the union of the languages generated by them is $L(G)$. We can construct the DFA M, as above explained, starting from the sets V' and P' and by choosing as initial state the set of all start symbols of those grammars. □

In [10], it is proven that for infinitely many integers h there exists a unary language which is generated by a CNFG with h variables, such that each equivalent DFA requires at least 2^{ch^2} states. Hence, the upper bound given in Theorem 3 cannot be reduced.

We leave as an open problem to give a tight estimation of the exponent of h in the upper bound.

4 The General Case

In this section we discuss the case of general CFLs, hence by removing the restriction to the bounded case. We start by extending the result stated in Theorem 3 to all CFLs defined over a binary alphabet. The proof mainly relies on a result presented in [8, Theorem 11], which can be formulated as follows:

Theorem 4. *Let $G = (V, \Sigma, P, S)$ be a* CNFG *with h variables and a terminal alphabet of two letters, and $L = L(G)$. Then $\psi(L) = \bigcup_{i \in I} Z_i$, where $\#I = O(h^2)$ and $Z_i \subseteq \mathbb{N}^2$ has the form*

$$Z_i = \bigcup_{\alpha_0 \in W_i} \{\alpha_0 + \alpha_{1,i} n + \alpha_{2,i} m \mid n, m \geq 0\},$$

where W_i is a finite subset of $\mathbb{N}^2$ and the integers in the vectors of W_i and in $\alpha_{1,i}$ and $\alpha_{2,i}$ do not exceed 2^{h^c}, for a constant $c > 0$.

Using Theorem 4 we are able to prove:

Theorem 5. *For each* CNFG *G with h variables and a terminal alphabet of two letters there exists a* DFA *with at most $2^{h^{O(1)}}$ states accepting a regular language Parikh equivalent to $L(G)$.*

Proof. (outline) First, we can express $\psi(L)$ as the union of a polynomial number of sets Z_i, according to Theorem 4. We can give a construction to build for each $i \in I$ a DFA M_i with $2^{h^{O(1)}}$ states accepting a language R_i such that $\psi(R_i) = Z_i$. Finally, we can define a DFA M which simulates in parallel all M_i's, in order to recognize $R = \bigcup_{i \in I} R_i$. Clearly $L =_\pi R$. Furthemore, from $\#I = O(h^2)$ we can conclude that the number of states of M is $2^{h^{O(1)}}$. □

We do not know whether or not the result stated in Theorem 5 can be extended to larger alphabets. In [8] a language L showing that Theorem 4 cannot be extended to an alphabet of three letters is presented. However, the counterexample L is of the form $(L'c)^*$, where $L' \subseteq \{a, b\}^*$ is a CFL. We can easily see that the proof of Theorem 5 can be extended to languages of that form. So L cannot be used as counterexample to show that Theorem 5 does not hold in the three letter case.

Hence, we conclude the paper by leaving as open problem the extension of Theorem 5 to languages defined over alphabet with at least three letter.

References

1. Parikh, R.J.: On context-free languages. J. ACM 13(4), 570–581 (1966)
2. Goldstine, J.: A simplified proof of Parikh's theorem. Discrete Mathematics 19(3), 235–239 (1977)
3. Huynh, D.T.: The Complexity of Semilinear Sets. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 324–337. Springer, Heidelberg (1980)
4. Aceto, L., Ésik, Z., Ingólfsdóttir, A.: A fully equational proof of Parikh's theorem. RAIRO - Theoretical Informatics and Applications 36(2), 129–153 (2002)
5. Ginsburg, S., Spanier, E.H.: Semigroups, Presburger formulas, and languages. Pacific J. Math. 16(2), 285–296 (1966)
6. Esparza, J.: Petri nets, commutative context-free grammars, and basic parallel processes. Fundam. Inform. 31(1), 13–25 (1997)
7. Verma, K.N., Seidl, H., Schwentick, T.: On the Complexity of Equational Horn Clauses. In: Nieuwenhuis, R. (ed.) CADE 2005. LNCS (LNAI), vol. 3632, pp. 337–352. Springer, Heidelberg (2005)
8. Kopczyński, E., To, A.W.: Parikh images of grammars: Complexity and applications. In: Symposium on Logic in Computer Science, pp. 80–89 (2010)
9. Esparza, J., Ganty, P., Kiefer, S., Luttenberger, M.: Parikh's theorem: A simple and direct automaton construction. Information Processing Letters 111(12), 614–619 (2011)
10. Pighizzini, G., Shallit, J., Wang, M.: Unary context-free grammars and pushdown automata, descriptional complexity and auxiliary space lower bounds. Journal of Computer and System Sciences 65(2), 393–414 (2002)
11. Ginsburg, S., Rice, H.G.: Two families of languages related to ALGOL. J. ACM 9, 350–371 (1962)
12. Gruska, J.: Descriptional complexity of context-free languages. In: MFCS, Mathematical Institute of the Slovak Academy of Sciences, pp. 71–83 (1973)
13. Ginsburg, S., Spanier, E.H.: Bounded Algol-like languages. Transactions of the American Mathematical Society 113(2), 333–368 (1964)
14. Malcher, A., Pighizzini, G.: Descriptional Complexity of Bounded Context-Free Languages. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 312–323. Springer, Heidelberg (2007)
15. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
16. Bader, C., Moura, A.: A generalization of Ogden's lemma. J. ACM 29, 404–407 (1982)
17. Dömösi, P., Kudlek, M.: Strong Iteration Lemmata for Regular, Linear, Context-Free, and Linear Indexed Languages. In: Ciobanu, G., Păun, G. (eds.) FCT 1999. LNCS, vol. 1684, pp. 226–233. Springer, Heidelberg (1999)

A Combinatorial Algorithm for All-Pairs Shortest Paths in Directed Vertex-Weighted Graphs with Applications to Disc Graphs

Andrzej Lingas[1] and Dzmitry Sledneu[2]

[1] Department of Computer Science, Lund University, 22100 Lund, Sweden
Andrzej.Lingas@cs.lth.se
[2] The Centre for Mathematical Sciences, Lund University, 22100 Lund, Sweden
Dzmitry.Sledneu@math.lu.se

Abstract. We consider the problem of computing all-pairs shortest paths in a directed graph with non-negative real weights assigned to vertices.

For an $n \times n$ $0-1$ matrix C, let K_C be the complete weighted graph on the rows of C where the weight of an edge between two rows is equal to their Hamming distance. Let $MWT(C)$ be the weight of a minimum weight spanning tree of K_C.

We show that the all-pairs shortest path problem for a directed graph G on n vertices with non-negative real weights and adjacency matrix A_G can be solved by a combinatorial randomized algorithm in time[1]

$$\widetilde{O}(n^2\sqrt{n+\min\{MWT(A_G), MWT(A_G^t)\}})$$

As a corollary, we conclude that the transitive closure of a directed graph G can be computed by a combinatorial randomized algorithm in the aforementioned time.

We also conclude that the all-pairs shortest path problem for vertex-weighted uniform disk graphs induced by point sets of bounded density within a unit square can be solved in time $\widetilde{O}(n^{2.75})$.

1 Introduction

The problems of finding shortest paths and determining their lengths are fundamental in algorithms. They have been extensively studied in algorithmic graph theory. A central open question in this area is if there is a substantially subcubic in the number of vertices algorithm for the all-pairs shortest path problem for directed graphs with real edge weights (APSP) in the addition-comparison model [24,27]. For several special cases of weights and/or graphs substantially subcubic algorithms for the APSP problem are known [3,8,23,25,26,27]. However, in the general case the fastest known algorithm due to Chan [8] (see also

[1] The notation $\widetilde{O}(\ )$ suppresses polylogarithmic factors and B^t stands for the transposed matrix B.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 373–384, 2012.

[9]) runs in time $O(n^3 \log^3 \log n / \log^2 n)$, achieving solely a moderate polylogarithmic improvement over the $O(n^3)$ bound yielded by Floyd-Warshall and Johnson's algorithms [1,27].

The situation is different for directed graphs with real vertex weights. Recently, Chan has shown that the APSP problem for the aforementioned graphs can be solved in time $O(n^{2.844})$ [8] and Yuster has slightly improved the latter bound to $O(n^{2.842})$ by using an improved bound on rectangular multiplication [25].

The basic tool in achieving substantially subcubic upper bounds on the running time for the APSP for directed graphs with constrained edge weights or real vertex weights are the fast algorithms for arithmetic square and rectangular matrix multiplication [10,14]. One typically exploits here the close relationship between the APSP problem and the so called distance or $(\min, +)$ product [3,23,24,25,27,26].

Unfortunately, these fast algorithms for matrix multiplication, yielding equally fast algorithms for Boolean matrix product, are based on recursive algebraic approaches over a ring difficult to implement. Thus, another central question in this area is whether or not there is a substantially subcubic *combinatorial* (i.e., not relaying on ring algebra) algorithm for the Boolean product of two $n \times n$ Boolean matrices [4,22,24]. Again, the fastest known combinatorial algorithm for Boolean matrix product due to Bansal and Williams [4] running in time $O(n^3 \log^2 \log n / \log^{9/4} n)$ achieves solely a moderate polylogarithmic improvement over the trivial $O(n^3)$ bound. On the other hand, several special cases of Boolean matrix product admit substantially subcubic combinatorial algorithms [5,13,20].

In particular, Björklund et al. [5] provided a combinatorial randomized algorithm for Boolean matrix product which is substantially subcubic in case the rows of the first $n \times n$ matrix or the columns of the second one are highly clustered, i.e., their minimum spanning tree in the Hamming metric has low cost. More exactly, their algorithm runs in time $\tilde{O}(n(n+c))$, where c is the minimum of the costs of the minimum spanning trees for the rows and the columns, respectively, in the Hamming metric. It relies on the fast Monte Carlo methods for computing an approximate minimum spanning tree in the L_1 and L_2 metrics given in [16,17].

The assumption that the input directed graph is highly clustered in the sense that the minimum spanning tree of the rows or columns of its adjacency matrix in the Hamming metric has a subquadratic cost does not yield any direct applications of the algorithm of Björklund et al. [5] to shortest path problems, not even to the transitive closure. The reason is that the cost of the analogous minimum spanning tree can grow dramatically in the power graphs[2] of the input graph. In particular, we cannot obtain directly an upper time-bound on the transitive closure of Boolean matrix corresponding to that for the Boolean matrix product from [5] by applying the asymptotic equality between the time complexity of

[2] In the i-th power graph there is an edge from v to u if there is a path composed of at most i edges from v to u in the input graph.

matrix product over a closed semi-ring and that of its transitive closure over the semi-ring due to Munro [21]. The reason is the dependence of the upper bound from [5] on the cost of the minimum spanning tree.

In this paper, we extend the idea of the method from [5] to include a mixed product of a real matrix with a Boolean one. We combine the aforementioned extension with the ideas used in the design of subcubic algorithms for important variants of the APSP problem [3,26], in particular those for directed graphs with vertex weights [8,25], to obtain not only a substantially subcubic combinatorial algorithm for the transitive closure but also for the APSP problem in highly clustered directed graphs with real vertex weights.

For an $n \times n$ $0-1$ matrix C, let K_C be the complete weighted graph on the rows of C where the weight of an edge between two rows is equal to their Hamming distance. Let $MWT(C)$ be the weight of a minimum weight spanning tree of K_C. We show that the all-pairs shortest path problem for a directed graph G on n vertices with non-negative real weights and an adjacency matrix A_G can be solved by a combinatorial randomized algorithm in $\widetilde{O}(n^2\sqrt{n + \min\{MWT(A_G), MWT(A_G^t)\}})$ time. It follows in particular that the transitive closure of a directed graph G can be computed by a combinatorial randomized algorithm in the aforementioned time.

Our algorithms are of Monte Carlo type and by increasing the polylogarithmic factor at the time bounds, the probability that they return a correct output within the bounds can be amplified to $1 - \frac{1}{n^\alpha}$, where $\alpha \geq 1$.

Since there are no practical or combinatorial substantially subcubic-time algorithms not only for the APSP problem but even for the transitive closure problem for arbitrary directed graphs at present, our simple adaptive method might be a potentially interesting alternative for a number of graph classes.

As an example of an application of our method, we consider the APSP problem for vertex-weighted uniform disk graphs induced by point sets of bounded density within a unit square. We obtain a combinatorial algorithm for this problem running in time $O(\sqrt{r}n^{2.75})$, where r is the radius of the disks around the vertices in a unit square.

The recent interest in disk graphs, in particular uniform disk graphs, stems from their applications in wireless networks. In this context, the restriction to point sets of bounded density is quite natural. In [11], Fürer and Kasiviswanathan provided a roughly $O(n^{2.5})$-time preprocessing for *approximate* $O(\sqrt{n})$-time distance queries in arbitrary disk graphs.

Our paper is structured as follows. In the next section, we show a reduction of the APSP problem for directed graphs with real vertex-weights to a mixed matrix product of a distance matrix over reals with the $0-1$ adjacency matrix. In Section 3, we present an algorithm for such a mixed product which generalizes that for the Boolean matrix product from [5] and runs in subcubic time if the input $0-1$ matrix is highly clustered. By combining the results of Sections 2,3, we can derive our main results in Section 4. In the next section, we present the application of our method to uniform disk graphs induced by point sets of bounded density. We conclude with final remarks.

2 A Reduction of APSP to Mixed Matrix Products

2.1 The APSP Problem

Formally, the All-Pairs Shortest Paths problem (APSP) in a directed graph $G = (V, E)$ with real weights $w(v)$ associated to vertices $v \in V$ is to compute the $|V| \times |V|$ distance matrix D_G such that $D_G(v, u)$ is the distance $\delta_G(v, u)$ from v to u in G, i.e., the minimum total weight of vertices on a path from v to u in G. An additional goal of the APSP problem is to compute a concise data structure representing the shortest paths.

Note that $\delta_G(v, u)$ is equal to the minimum total weight of inner vertices on a path from v to u in G increased by the weights of v and u.

We shall assume $|V| = n$ throughout the paper.

For $i = 0, 1, ..., n-1$, let $\delta^i_G(v, u)$ be the distance from v to u on paths consisting of at most i edges, i.e., the minimum total weight of vertices on a path from v to u having at most i edges in G. Next, let D^i_G be the $|V| \times |V|$ matrix such that $D^i_G[v, u]$ is equal to $\delta^i_G(v, u)$.

For convention, we assume $\delta^0_G(v, v) = 0$ and $\delta^0_G(v, u) = +\infty$ for $v \neq u$. Hence, D^0_G has zeros on the diagonal and $+\infty$ otherwise. In D^1_G, all the entries $D^1_G[v, u]$ where $(v, u) \in E$ are set to $w(v) + w(u)$ instead of $+\infty$. Thus, both D^0_G and D^1_G can be easily computed in time $O(n^2)$.

2.2 Mixed Matrix Products

Let A be an $n \times n$ matrix over $R \cup \{+\infty\}$, and let B be an $n \times n$ matrix with entries in $\{0, 1\}$. The *mixed right product* C of A and B is defined by

$$C[i, j] = \min\{A[i, k] | 1 \leq k \leq n \ \& \ B[k, j] = 1\} \cup \{+\infty\}.$$

If $C[i, j] \neq +\infty$ then the index k such that $C[i, j] = A[i, k]$ (and thus $B[k, j] = 1$) is called a witness for $C[i, j]$. Analogously, the *mixed left product* C' of B and A is defined by

$$C'[i, j] = \min\{A[k, j] | 1 \leq k \leq n \ \& \ B[i, k] = 1\} \cup \{+\infty\},$$

and if $C'[i, j] \neq +\infty$ then the index k such that $C'[i, j] = A[k, j]$ is called a witness for $C'[i, j]$.

An $n \times n$ matrix W such that whenever $C[i, j] \neq +\infty$ then $W[i, j]$ is a witnesses for $C[i, j]$ is called a witness matrix for the right mixed product of A and B. Analogously, we define a witness matrix for the left mixed product of B and A.

2.3 The Reduction

Let A_G denote the $n \times n$ adjacency matrix of $G = (V, E)$, i.e., $A_G[v, u] = 1$ iff $(v, u) \in E$.

Lemma 1. *For an arbitrary $i \in \{0, 1, ..., n-2\}$, D^{i+1}_G can be computed on the basis of D^i_G and the right mixed product of D^i_G with A_G or D^i_G and the left mixed product of A_G with D^i_G in time $O(n^2)$.*

Proof. It is sufficient to observe that for any pair $v,\ u$ of vertices in G, $D_G^{i+1}[v,u]$ is equal to

$$\min\{D_G^i[v,u], \min\{D_G^i[v,x] + w(u) | 1 \leq x \leq n \ \&\ A_G[x,u] = 1\} \cup \{+\infty\}\}$$

Symmetrically $D_G^{i+1}[v,u]$ is equal to

$$\min\{D_G^i[v,u], \min\{D_G^i[x,u] + w(v) | 1 \leq x \leq n \ \&\ A_G[v,x] = 1\} \cup \{+\infty\}\}$$

□

The following lemma follows the general strategy used to prove Theorem 3.4 in [8].

Lemma 2. *Let G be a directed graph G on n vertices with non-negative real vertex weights. Suppose that the right (or left) mixed product of an $n \times n$ matrix over $R \cup \{+\infty\}$ with the adjacency matrix A_G of G along with the witness matrix can be computed in time $T_{mix}(n) = \Omega(n^2)$. The APSP problem for G can be solved in time $\widetilde{O}(n^{1.5}\sqrt{T_{mix}(n)})$.*

Proof. We begin by computing D_G^{t-1} for some $t \in [2, ..., n]$ which will be specified later. By Lemma 1 this computation takes time $O(tT_{mix}(n))$.

It remains to determine distances between pairs of vertices where any shortest path consists of at least t edges. For this purpose, we determine a subset B of V, the so called bridging set [26], hitting all the aforementioned long paths. We apply the following fact to $l = t$ and sets of t vertices on shortest consisting of exactly $t-1$ edges, similarly as in [3,8,25,26].

Fact 1. *Given a collection of N subsets of $\{1, ..., n\}$, where each subset has size exactly l, we can find a subset B of size $O((n/l)\log n)$ that hits all subsets in the collection in time $O(Nl)$.*

Since our application of Fact 1 is analogous to those in [3,8,25,26], we solely sketch it referring the reader for details to the aforementioned papers.

Note that for each pair $v,\ u$, of vertices for which any shortest path has at least t edges there is a pair v', u' of vertices on a shortest path from v to u such that any shortest path from v' to u' has exactly $t-1$ edges. For all such pairs v', u', we can find a shortest path on $t-1$ edges, and thus on t vertices, by backtracking on the computation of D_G^{t-1} and using witnesses for the mixed products. In total, we generate $O(n^2)$ such paths on t vertices in time $O(tn^2)$. The application of Fact 1 also takes time $O(tn^2)$.

Next, we run Dijkstra's single-source shortest path algorithm [1] for all vertices in the bridging set B in the input graph G and in the graph resulting from reversing the direction of edges in G. In this way, we determine $D_G[v,u]$ for all pairs $(v,u) \in (B \times V) \cup (V \times B)$.

Now, it is sufficient for all remaining pairs (v,u) in $V \times V$ to set

$$D_G[v,u] = \min\{D_G^{t-1}(v,u), \min_{b \in B}\{D_G[v,b] + D_G[b,u] - w(b)\}$$

in order to determine the whole D_G.

The computation of D_G^{t-1} takes $O(tT_{mix}(n))$ time which asymptotically is not less than the $O(tn^2)$ time taken by the construction of the bridging set. The runs of Dijkstra's algorithm and the final computation of D_G require $\widetilde{O}(\frac{n}{t}n^2)$ time. By setting $t = \sqrt{\frac{n^3}{T_{mix}(n)}}$, we obtain the lemma. □

3 Fast Computation of the Mixed Products for Clustered Data

Our algorithm for the right (or, left) mixed product relies on computation of an approximate minimum spanning tree of the columns (or rows, respectively) of the Boolean input matrix in the Hamming metric.

3.1 Approximate Minimum Spanning Tree in High Dimensional Space

For $c \geq 1$ and a finite set S of points in a metric space, a *c-approximate minimum spanning tree for S* is a spanning tree in the complete weighted graph on S, with edge weights equal to the distances between the endpoints, whose total weight is at most c times the minimum.

In [16] (section 4.3) and [15] (section 3), Indyk and Motwani in particular considered the bichromatic ϵ-approximate closest pair problem for n points in R^d with integer coordinates in $O(1)$ under the L_p metric, $p \in \{1, 2\}$. They showed that there is a dynamic data structure for this problem which supports insertions, deletions and queries in time $O(dn^{1/(1+\epsilon)})$ and requires $O(dn + n^{1+1/(1+\epsilon)})$-time preprocessing. In consequence, by a simulation of Kruskal's algorithm they deduced the following fact.

Fact 2. *For $\epsilon > 0$, a $1+\epsilon$-approximate minimum spanning tree for a set of n points in R^d with integer coordinates in $O(1)$ under the L_1 or L_2 metric can be computed by a Monte Carlo algorithm in time $O(dn^{1+1/(1+\epsilon)})$.*

In [17] Indyk, Schmidt and Thorup reported even slightly more efficient (by a poly-log factor) reduction of the problem of finding a $1+\epsilon$-approximate minimum spanning tree to the bichromatic ϵ-approximate closest pair problem via an easy simulation of Prim's algorithm.

Note that the L_1 metric for points in R^n with 0, 1-coordinates coincides with the n-dimensional Hamming metric. Hence, Fact 2 immediately yields the following corollary.

Corollary 1. *For $\epsilon > 0$, a $1+\epsilon$-approximate minimum spanning tree for a set of n $0-1$ strings of length n under the Hamming metric can be computed by a Monte Carlo algorithm in time $O(n^{2+1/(1+\epsilon)})$.*

3.2 The Algorithm for Mixed Matrix Product

The idea of our combinatorial algorithm for the right mixed product C of A with B and its witness matrix is a generalization of that from [5]. Let $P(r, v)$

denote a priority queue (implemented as a heap) on the entries $A[r,k]$ such that $B[k,v]=1$ ordered by their values in nondecreasing order.

First, we compute an approximate minimum spanning tree of the columns of B in the Hamming metric. Then, we fix a traversal of the tree. Next, for each row r of A, we traverse the tree, construct $P(r,start)$ where *start* is the first column of B in the tree traversal and then maintain $P(r,v)$ for the currently traversed v by updating $P(r,u)$ where u is the predecessor of v in the traversal. A minimum element in $P(r,v)$ yields a witness for $C[r,v]$. The cost of the updates in a single traversal of the tree is proportional to the cost of the tree modulo a logarithmic factor.

Algorithm 1.
Input: $n \times n$ matrix A over $R \cup \{+\infty\}$ and an $n \times n$ Boolean matrix B;
Output: A witness matrix W for the right mixed product C of A and B.
Comment: $P(r,v)$ stands for a priority queue on the entries $A[r,k]$ s.t. $B[k,v]=1$ ordered by their values in nondecreasing order.

1. Compute an $O(\log n)$-approximate minimum spanning tree T_B of the columns of B in the Hamming metric;
2. Fix a traversal of the tree T_B linear in its size;
3. Set *start* to the first node of the traversal;
4. For each pair of consecutive neighboring columns v, u in the traversal, precompute the set $D_{v,u}$ of positions where 1s occur in v but not in u and the set $D_{u,v}$ of positions where 1s occur in u but not in v;
5. For each row r of A do
   - Construct the priority queue $P(r,start)$ and if $P(r,start) \neq \emptyset$ set $W[r,start]$ to the index k where $A[r,k]$ is the minimum element in $P(r,start)$;
   - Traverse the tree T_B and for each node v different from *start* compute the priority queue $P(r,v)$ from the priority queue $P(r,u)$, where u is the predecessor of v in the traversal, by utilizing $D_{v,u}$ and $D_{u,v}$. If $P(r,v) \neq \emptyset$ set $W[r,v]$ to the index k where $A[r,k]$ is the minimum element in $P(r,v)$.

Lemma 3. *Algorithm 1 is correct, i.e., it outputs the witnesses matrix for the right mixed product of matrices A and B.*

For an $n \times n$ Boolean matrix C, let K_C be the complete weighted graph on the rows of C where the weight of an edge between two rows is equal to their Hamming distance. Next, let $MWT(C)$ be the weight of a minimum weight spanning tree of K_C.

Lemma 4. *Algorithm 1 can be implemented in time $\widetilde{O}(n(n+MWT(B^t)))+t(n)$, where $t(n)$ is the time taken by the construction of the $O(\log n)$-approximate minimum weight spanning tree in step 1.*

Proof. Step 1 can be implemented in time $t(n)$ while steps 2,3 take time $O(n)$. Step 4 takes $O(n^2)$ time. The block in Step 5 is iterated n times.

The first step in the block, i.e., the construction of $P(r,start)$ takes $O(n \log n)$ time. The update of $P(r,u)$ to $P(r,v)$ takes $O(\log n(|D_{v,u}|+|D_{u,v}|))$ time. Note

that $|D_{v,u}| + |D_{u,v}|$ is precisely the Hamming distance between the columns v and u. It follows by the $O(\log n)$ approximation factor of T_B that the total time taken by these updates is $O(MWT(B^t)\log^2 n)$.

We conclude that Step 5 can be implemented in time $\widetilde{O}(nMWT(B^t))$. □

Theorem 1. *The right mixed product of two $n \times n$ matrices A over $R \cup \{+\infty\}$ and B over $\{0,1\}$ can be computed by a combinatorial randomized algorithm in time $\widetilde{O}(n(n+MWT(B^t)))$. Analogously, the left mixed product of B and A can be computed by a combinatorial randomized algorithm in time $\widetilde{O}(n(n+MWT(B)))$.*

Proof. By Corollary 1, an $\Theta(\log n)$-approximate minimum spanning tree can be constructed by a Monte Carlo algorithm in time $\widetilde{O}(n^2)$ (observe that $n^{1/f} = O(1)$ if $f = \Omega(\log n)$). Hence, by Lemmata 3, 4, we obtain the theorem for the right mixed product. The upper bound on the time required to compute the left mixed product follows symmetrically. □

4 Main Results

Lemma 2 combined with Theorem 1 yield our main result.

Theorem 2. *Let G a directed graph G on n vertices with non-negative real vertex weights. The all-pairs shortest path problem for G can be solved by a combinatorial randomized algorithm in time $\widetilde{O}(n^2\sqrt{n + \min\{MWT(A_G), MWT(A_G^t)\}})$.*

By setting vertex weights, say, to zero, we obtain immediately the following corollary.

Corollary 2. *The transitive closure of a directed graph G on n vertices can be computed by a combinatorial randomized algorithm in time $\widetilde{O}(n^2\sqrt{n + \min\{MWT(A_G), MWT(A_G^t)\}})$.*

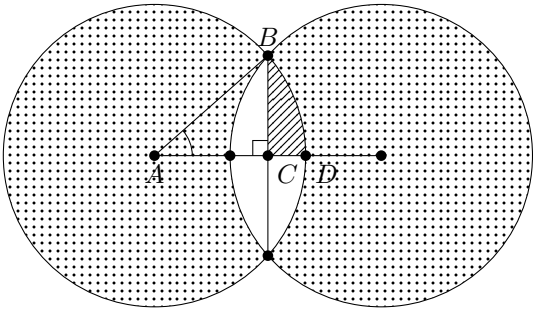
Equivalently, we can formulate Corollary 2 as follows.

Corollary 3. *The transitive closure of an $n \times n$ Boolean matrix B (over the Boolean semi-ring) can be computed by a combinatorial randomized algorithm in time $\widetilde{O}(n^2\sqrt{n + \min\{MWT(B), MWT(B^t)\}})$.*

5 APSP in Vertex-Weighted Uniform Disk Graphs of Bounded Density

In this section, we consider uniform disk graphs that are induced by a set P of n points in a unit square in the plane that are $b(n)$-dense, where $b : N \to N$. Formally, we say that P is $b(n)$-dense iff each cell of the regular $\sqrt{n} \times \sqrt{n}$ grid within the unit square contains at most $b(n)$ points. The vertices of such an induced disk graph are the points in P, and two vertices are adjacent in the graph iff their Euclidean distance is at most r, where r is a positive constant not exceeding 1. We shall term the aforementioned graphs as *uniform disk graphs induced by $b(n)$-dense point sets.*

Lemma 5. *Given two intersecting disks on the plane of the same radius r with the distance d between centers, the area of the symmetric difference is $O(rd)$.*



Proof. $AC = \frac{d}{2}, AB = r$.

The area of the triangle ABC is

$$Area_{ABC} = \frac{1}{2}ACBC = \frac{1}{2}\frac{d}{2}\sqrt{r^2 - \frac{d^2}{4}} = \frac{1}{8}d\sqrt{4r^2 - d^2}.$$

The area of the circular sector ABD is

$$Area_{ABD} = \frac{1}{2}r^2 \angle BAC = \frac{1}{2}r^2 \arccos\left(\frac{d}{2r}\right).$$

The area of BCD is $Area_{BCD} = Area_{ABD} - Area_{ABC}$.

The area of the symmetric difference

$$Area = 2(\pi r^2 - 4Area_{BCD}) = 2\pi r^2 - 4r^2 \arccos\left(\frac{d}{2r}\right) + d\sqrt{4r^2 - d^2}.$$

Finally, by using Taylor series expansion

$$4r^2 \arccos\left(\frac{d}{2r}\right) = 4r^2\left(\frac{\pi}{2} - \frac{d}{2r} + O\left(\left(\frac{d}{2r}\right)^2\right)\right) =$$
$$= 2\pi r^2 - 2dr + O(d^2) = 2\pi r^2 - 2dr + O(rd)$$

and $\sqrt{4r^2 - d^2} \leq 2r$ we get $Area = O(rd)$. □

Lemma 6. *Let G be a uniform disk graph induced by a $b(n)$-dense point set. For each edge (v, u) of G, the number of vertices in G that are a neighbor of exactly one of the vertices v, u, i.e., the Hamming distance between the two rows in the adjacency matrix of G corresponding to v and u, respectively, is $O(r \times b(n)(dist(v, u) \times n + \sqrt{n}))$.*

Proof. The number of vertices of G that are a neighbor of exactly one of the vertices v and u is at most the minimum number of cells of the regular $\sqrt{n} \times \sqrt{n}$

grid within the unit square that cover the symmetric difference $S(v,u)$ between the disks centered at v and u, respectively, multiplied by $b(n)$. The aforementioned number of cells is easily seen to be at most the area $A(v,u)$ of $S(v,u)$ divided by the area of the grid cell, i.e., $A(v,u) \times n$, plus the number of cells of the grid intersected by the perimeter of $S(v,u)$, i.e., $O(r\sqrt{n})$. By Lemma 5, we have $A(v,u) = O(dist(v,u) \times r)$. Hence, the aforementioned number of cells is $O(r(dist(v,u) \times n + \sqrt{n}))$. □

The following lemma is a folklore (e.g., it follows directly from the upper bound on the length of closed path through a set of points in a d-dimensional cube given in Lemma 2 in [18]).

Lemma 7. *The minimum Euclidean spanning tree of any set of n points in a unit square in the plane has total length $O(\sqrt{n})$.*

Combining Lemmata 6, 7, we obtain the following one.

Lemma 8. *For a uniform disk graph G induced by a $b(n)$-dense n-point set, a spanning tree of the rows (or, columns) of the adjacency matrix of G in the Hamming metric having cost $O(rn^{3/2})$ can be found in time $O(n^2)$.*

Proof. Construct a minimum Euclidean spanning tree of the n points forming the vertex set of G. It takes time $O(n \log n)$ and the resulting tree T has total length $O(\sqrt{n})$ by Lemma 7. Form a spanning tree U of the rows (or, columns) of the adjacency matrix of G by connecting by edge the rows corresponding to v and u iff $(v,u) \in T$. By Lemma 6 and the $O(\sqrt{n})$ length of T, the total cost of U is $O(rn^{3/2}b(n))$. □

By plugging Lemma 8 into Theorem 2, we obtain our main result in this section.

Theorem 3. *Let G be a uniform disk graph with non-negative real vertex weights induced by a $b(n)$-dense n-point set. The all-pairs shortest path problem for G can be solved by a combinatorial algorithm in time $\widetilde{O}(\sqrt{r}n^{2.75}\sqrt{b(n)})$.*

In the application of the method of Theorem 2 yielding Theorem 3, we can use the deterministic algorithm of Lemma 8 to find a spanning tree of the rows or columns of the adjacency matrix of G instead of the randomized approximation algorithm from Fact 2.

By straightforward calculations, our upper time-bound for APSP in vertex-weighted uniform disk graphs induced by $O(1)$-dense point sets subsumes that for APSP in sparse graphs based on Dijkstra's single-source shortest-path algorithm, running in time $\widetilde{O}(nm)$, where m is the number of edges, for $r >> n^{-1/6}$.

Finally, we can also easily extend Theorem 3 to include uniform ball graphs in a d-dimensional Euclidean space. In the extension, the term $\sqrt{r}$ in the upper time-bound generalizes to $\sqrt{r^{d-1}}$.

6 Final Remarks

We can easily extend our main result to include solving the APSP problem for vertex and edge weighted directed graphs in which the number of different edge

weights is bounded, say by q. This can be simply achieved by decomposing the adjacency matrix A_G into the union of up to q matrices $A_1, A_2, ...A_l$ in one-to-one correspondence with the distinct edge weights and consequently replacing each mixed product with l such products in Lemmata 1, 2. In the final upper bound, $MWT(A_G)$ and $MWT(A_G^t)$ are replaced by $\sum_{i=1}^{l} MWT(A_i)$ and $\sum_{i=1}^{l} MWT(A_i^t)$, respectively.

It is an interesting problem to determine if there are other natural graph classes where $MWT(A_G)$ or $MWT(A_G^t)$ are substantially subquadratic in the number of vertices.

It follows from the existence of the so called Hadamard matrices [7] that there is an infinite sequence of graphs with $n_i \times n_i$ adjacency matrices A_i such that $\min\{MWT(A_i), MWT(A_i^t)\} = \Omega((n_i)^2)$ holds.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
2. Alon, N., Naor, M.: Derandomization, Witnesses for Boolean Matrix Multiplication and Construction of Perfect hash functions. Algorithmica 16, 434–449 (1996)
3. Alon, N., Galil, Z., Margalit, O.: On the exponent of all pairs shortest path problem. J. Comput. System Sci. 54, 25–51 (1997)
4. Bansal, N., Williams, R.: Regularity Lemmas and Combinatorial Algorithms. In: Proc. of 50th IEEE Symposium on Foundations on Computer Science, Atlanta (2009)
5. Björklund, A., Lingas, A.: Fast Boolean Matrix Multiplication for Highly Clustered Data. In: Dehne, F., Sack, J.-R., Tamassia, R. (eds.) WADS 2001. LNCS, vol. 2125, pp. 258–263. Springer, Heidelberg (2001)
6. Borodin, A., Ostrovsky, R., Rabani, Y.: Subquadratic Approximation Algorithms For Clustering Problems in High Dimensional Spaces. In: Proceedings of the 31st ACM Symposium on Theory of Computing (1999)
7. Cameron, P.J.: Combinatorics. Cambridge University Press (1994)
8. Chan, T.M.: More algorithms for all-pairs shortest paths in weighted graphs. SIAM J. Comput. 39(5), 2075–2089; preliminary version in proc. STOC 2007, pp. 590–598 (2007)
9. Chan, T.M.: All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. Algorithmica 41, 330–337 (2008)
10. Coppersmith, D., Winograd, S.: Matrix Multiplication via Arithmetic Progressions. J. of Symbolic Computation 9, 251–280 (1990)
11. Fürer, M., Kasiviswanathan, S.P.: Approximate Distance Queries in Disk Graphs. In: Erlebach, T., Kaklamanis, C. (eds.) WAOA 2006. LNCS, vol. 4368, pp. 174–187. Springer, Heidelberg (2007)
12. Galil, Z., Margalit, O.: Witnesses for Boolean Matrix Multiplication and Shortest Paths. Journal of Complexity, 417–426 (1993)
13. Gąsieniec, L., Lingas, A.: An Improved Bound on Boolean Matrix Multiplication for Highly Clustered Data. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) WADS 2003. LNCS, vol. 2748, pp. 329–339. Springer, Heidelberg (2003)
14. Huang, X., Pan, V.Y.: Fast rectangular matrix multiplications and applications. Journal of Complexity 14(2), 257–299 (1998)

15. Indyk, P.: High-dimensional computational geometry. PhD dissertation, Standford University (2000)
16. Indyk, P., Motwani, R.: Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In: Proceedings of the 30th ACM Symposium on Theory of Computing (1998)
17. Indyk, P., Schmidt, S.E., Thorup, M.: On reducing approximate mst to closest pair problems in high dimensions (1999) (manuscript)
18. Karp, R.M., Steele, J.M.: Probabilistic analysis of heuristics. In: Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (eds.) The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, ch. 6, pp. 181–205. John Wiley & Sons Ltd. (1985)
19. Kushilevitz, E., Ostrovsky, E., Rabani, Y.: Efficient search for approximate nearest neighbor in high dimensional spaces. SIAM J. Comput. 30(2), 457–474; Preliminary version in Proc. 30th STOC (1989)
20. Lingas, A.: A Geometric Approach to Boolean Matrix Multiplication. In: Bose, P., Morin, P. (eds.) ISAAC 2002. LNCS, vol. 2518, pp. 501–510. Springer, Heidelberg (2002)
21. Munro, J.I.: Efficient determination of the transitive closure of a directed graph. Information Processing Letters 1(2), 56–58 (1971)
22. Rytter, W.: Fast recognition of pushdown automaton and context-free languages. Information and Control 67(1-3), 12–22 (1985)
23. Seidel, R.: On the All-Pairs-Shortest-Path Problem. In: Proc. 24th ACM STOC, pp. 745–749 (1992)
24. Vassilevska, V., Williams, R.: Subcubic Equivalences Between Path, Matrix, and Triangle Problems. In: Proceedings 51st Annual IEEE Symposium on Foundations of Computer Science, FOCS (2010)
25. Yuster, R.: Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In: Proc. of the 20th ACM-SIAM Symposium on Discrete Algorithms, pp. 950–957 (2009)
26. Zwick, U.: All pairs shortest paths using bridging rectangular matrix multiplication. Journal of the ACM 49(3), 289–317 (2002)
27. Zwick, U.: Exact and Approximate Distances in Graphs - A survey. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 33–48. Springer, Heidelberg (2001)

The Complexity of Small Universal Turing Machines: A Survey*

Turlough Neary[1] and Damien Woods[2]

[1] School of Computer Science & Informatics, University College Dublin, Ireland
turlough.neary@ucd.ie
[2] Division of Engineering & Applied Science, California Institute of Technology, Pasadena, CA 91125, USA
woods@caltech.edu

Abstract. We survey some work concerned with small universal Turing machines, cellular automata, tag systems, and other simple models of computation. For example it has been an open question for some time as to whether the smallest known universal Turing machines of Minsky, Rogozhin, Baiocchi and Kudlek are efficient (polynomial time) simulators of Turing machines. These are some of the most intuitively simple computational devices and previously the best known simulations were exponentially slow. We discuss recent work that shows that these machines are indeed efficient simulators. In addition, another related result shows that Rule 110, a well-known elementary cellular automaton, is efficiently universal. We also discuss some old and new universal program size results, including the smallest known universal Turing machines. We finish the survey with results on generalised and restricted Turing machine models including machines with a periodic background on the tape (instead of a blank symbol), multiple tapes, multiple dimensions, and machines that never write to their tape. We then discuss some ideas for future work.

1 Introduction

In this survey we explore results related to the time and program size complexity of universal Turing machines, and other models of computation. We also discuss results for variants on the Turing machine model to give an idea of the many strands of work in the area. Of course the choice of topics is incomplete and reflects the authors' interests, and there are other related surveys that may interest the reader [32,38,41,57].

In 1956 Shannon [95] considered the question of finding the smallest possible universal Turing machine [99], where size is the number of states and symbols.

* This paper is extended and updated from [110]. T. Neary is supported by Science Foundation Ireland, Grant Number 09/RFP/CMS2212. D. Woods is supported by National Science Foundation Grant 0832824, the Molecular Programming Project. We thank Astrid Haberleitner for her tireless work in translating a number of highly technical papers from German to English, and Beverley Henley for her support.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 385–405, 2012.

In the early Sixties, Minsky and Watanabe had a running competition to see who could find the smallest universal Turing machine [51,54,103,104]. Early attempts [23,104] gave small universal Turing machines that efficiently (in polynomial time) simulated Turing machines. In 1962, Minsky [54] found a small 7-state, 4-symbol universal machine. Minsky's machine worked by simulating 2-tag systems, which were shown to be universal by Cocke and Minsky [8,55]. Rogozhin [88] extended Minsky's technique of 2-tag simulation and found small machines with a number of state-symbol pairs. Subsequently, some of Rogozhin's machines were reduced in size or improved by Robinson [86,91], Kudlek and Rogozhin [27], and Baiocchi [4]. All of the smallest known 2-tag simulators are plotted as circles in Figure 1. Also, Table 1 lists a number of these machines.

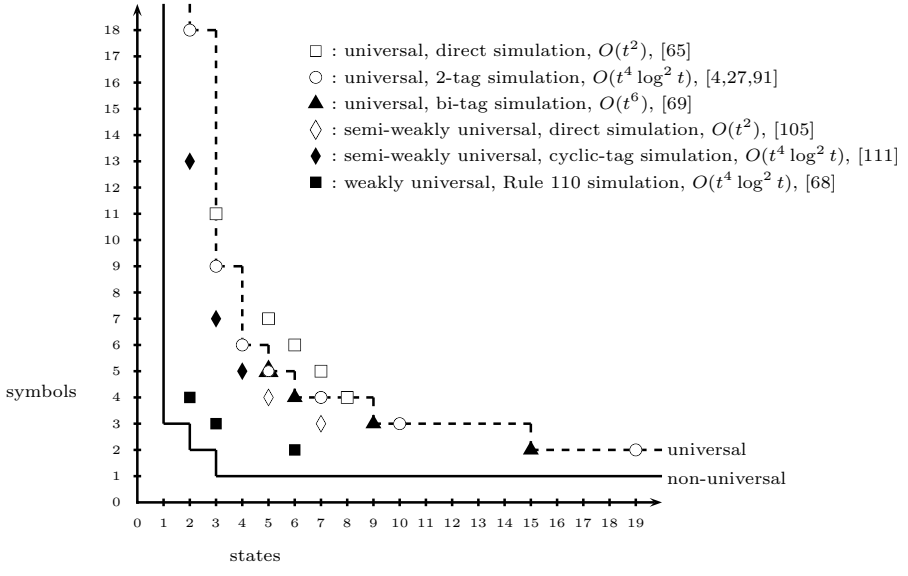


Fig. 1. State-symbol plot of small universal Turing machines. The type of simulation is given for each group of machines. Simulation time overheads are given in terms of simulating a single-tape deterministic Turing machine that runs in time t.

Unfortunately, Cocke and Minsky's 2-tag simulation of Turing machines was exponentially slow. The exponential slowdown was essentially caused by the use of a unary encoding of Turing machine tape contents. Therefore, for many years it was entirely plausible that there was an exponential trade-off between program size complexity on the one hand, and time/space complexity on the other: the smallest universal Turing machines seemed to be exponentially slow.

Figure 1 shows a non-universal curve. This curve is a lower bound that gives the state-symbol pairs for which it is known that the halting problem is decidable. The 1-symbol case is trivial and Shannon [95] claimed that 1-state Turing machines are non-universal. However, both Fischer [12] and Nozaki [70] noted that Shannon's definition of universal Turing machine is too strict and so his proof is not sufficiently general. Later, the 1-state case was shown by Hermann [19].

Table 1. Small standard universal Turing machines, ordered by date and then by state-symbol product. If there are multiple machines with the same state-symbol pair, the machine with the smallest number of instructions is denoted *.

states	symbols	state-symbol product	author
m	2	$2m$	Shannon [95]
2	n	$2n$	Shannon [95]
12	6	72	Takahashi [98] (mentioned in [104])
10	6	60	Ikeno [23] (also appears in [51])
8	6	48	Watanabe [103] (mentioned in [54])
7	6	42	Minsky [51]
8	5	40	Watanabe [104]
9	4	36	Tritter (mentioned in [54])
25	2	50	Minsky [55]
6	6	36	Minsky [54]
7	4	28	Minsky [54]
24	2	48	Rogozhin [87,88,91]
2	21	42	Rogozhin [87,88]
11	3	33	Rogozhin [87,88]
3	10	30	Rogozhin [87,88]
7	4	28	Rogozhin [87,88,91]
5	5	25	Rogozhin [87,88,91]
4	6	24	Rogozhin [87,88,91]
2	18	36	Rogozhin [91]
10	3	30	Rogozhin [89,91]
3	10	30	Rogozhin [90,91]*
22	2	44	Rogozhin [92]
19	2	38	Baiocchi [4]
7	4	28	Baiocchi [4]*
3	9	27	Kudlek & Rogozhin [27]
18	2	36	Neary & Woods [66]
9	3	27	Neary & Woods [69]
5	5	25	Neary & Woods [69]*
6	4	24	Neary & Woods [69]
15	2	30	Neary & Woods [69]

Pavlotskaya [75] and, via another method, Kudlek [26] have shown that there are no universal 2-state, 2-symbol machines, where one transition rule is reserved for halting. Pavlotskaya [77] has also shown that there are no universal 3-state, 2-symbol machines, and also claimed [75], without publishing a proof, that there are no universal machines for the 2-state, 3-symbol case. Again, both of these cases assume that a transition rule is reserved for halting.

2 Time and Size Efficiency of Universal Machines

As mentioned above, some of the very earliest small Turing machines were polynomial time simulators. Subsequently, attention turned to the smaller, but exponentially slower, 2-tag simulators given by Minsky, Rogozhin and others.

Recently [65] we have given small machines that are efficient polynomial time simulators. More precisely, if M is a deterministic single-tape Turing machine that runs in time t and space s, then there are machines, with state-symbol pairs given by the squares in Figure 1, that directly simulate M in polynomial time $O(t^2)$ and linear space $O(s)$. These machines define a $O(t^2)$ curve. They are currently the smallest known universal Turing machines that simulate Turing machines in $O(t^2)$ time. Their $O(s)$ space usage is also extremely efficient, more efficient than the other machines in Figure 1, all of which use space that is up to square root of their simulation time.

Despite the existence of these efficient $O(t^2)$ simulators, it still remained the case that the smallest universal machines were exponentially slow. However, we have recently shown that the smallest machines are in fact efficient simulators of Turing machines, by showing that 2-tag systems are efficient [108]. Tag systems are one of a number of rewriting systems invented in the 1920s by Post, although published somewhat later [79]. Post wanted to prove the decidability of various properties of tag systems, but found that even very simple examples had extremely complicated behaviour. Forty years later, Minsky showed that tag systems [53] are in fact computationally universal, and then Cocke and Minsky [8,55] showed universality for a particularly simple form called 2-tag systems. Minsky [54,55] saw that one could find very small universal Turing machines by simulating 2-tag systems, and since then 2-tag systems have been at the core of many results in the field.

A 2-tag system acts on a dataword, which is a string of symbols taken from a finite alphabet Σ. There is a fixed set of rules $R : \Sigma \to \Sigma^*$. In a single timestep, the leftmost symbol σ_j of the dataword is read, if there is a rule $\sigma_j \to \alpha_j$ then the string α_j is appended to the right of the dataword and the leftmost *two* dataword symbols are deleted. This process is iterated until a suitable halting condition is reached (i.e. there is no rule for the read symbol, the dataword has length less than 2, or the 2-tag system enters a repeating loop). Part of the reason why it was presumed that 2-tag systems were exponentially slow is that it is not obvious how to locate a specific symbol based solely on its position relative to other symbols in the dataword (one might want to do this to simulate the local action of a Turing machine tape head). The main result of [108] uses an algorithm that solves this problem, and does so efficiently.

More precisely, given a deterministic single-tape Turing machine M that runs in time t, there is a 2-tag system that simulates M and runs in polynomial time $O(t^4 \log^2 t)$. The small machines of Minsky, Rogozhin, and others have a quadratic time overhead when simulating 2-tag systems, hence by the result in [108] they simulate Turing machines in time $O(t^8 \log^4 t)$. It turns out that the time overhead can be improved [63] to $O(t^4 \log^2 t)$, giving the $O(t^4 \log^2 t)$ time overhead for the machines shown in Figure 1 as hollow circles. Thus, there is currently little evidence for the claim of an exponential trade-off between program size complexity, and time/space complexity.

From the point of view of program size, Neary and Woods [63,69] have recently given four Turing machines that are presently the smallest known (standard)

machines with 2, 3, 4 and 5 symbols. The 5-symbol machine improves on the 5-symbol machine of Rogozhin [91] by one transition rule. The remainder of these machines improve on the 2- and 4-symbol machines of Baiocchi [4], and the 3-symbol machine of Rogozhin [91]. These small machines simulate Turing machines in polynomial time $O(t^6)$ and are illustrated as triangles in Figure 1. They were proven universal via simulation of our universal variant of tag systems called *bi-tag systems* [69]. Bi-tag systems are essentially 1-tag systems (and so they read and delete one symbol per timestep) augmented with additional context sensitive rules that read, and delete, two symbols per timestep. Bi-tag systems are a restriction of Post's normal systems [79]. On the one hand bi-tag systems are universal, while on the other hand they are sufficiently 'simple' to be simulated by such small machines.

Exponentially improving the time efficiency of 2-tag systems has implications for a number of models of computation, besides small universal Turing machines. Following our result, the simulation efficiency of many biologically inspired models of computation, including neural networks, H systems and P systems, has been improved from exponential to polynomial. For example, Siegelmann and Margenstern [96] give a neural network that uses only nine high-order neurons to simulate 2-tag systems. Taking each synchronous update of the nine neurons as a single parallel timestep, their neural network simulates 2-tag systems in linear time. They note that "tag systems suffer a significant slow-down ... and thus our result proves only Turing universality and should not be interpreted complexity-wise as a Turing equivalent." Now we know that their neural network is in fact efficiently universal. Rogozhin and Verlan [93] give a tissue P system with eight rules that simulates 2-tag systems in linear time, and thus we have improved its simulation time overhead from exponential to polynomial. This system uses splicing rules (from H systems) with membranes (from P systems) and is non-deterministic. Harju and Margenstern [18] gave an extended H-system with 280 rules that generates recursively enumerable sets using Rogozhin's 7-state, 4-symbol universal Turing machine. Using our result from 2-tag systems, the time efficiency of their construction is improved from exponential to polynomial, with a possible small constant increase in the number of rules. The efficiency of Hooper's [22] small 2-tape universal Turing machine is also improved from exponential to polynomial, as is Rothemund's [94] restriction enzyme implementation of Minsky's 7-state, 4-symbol UTM. The technique of simulation via 2-tag systems is at the core of many of the universality proofs in Margenstern's survey [41]. Our work exponentially improves the time overheads in these simulations, such as Lindgren and Nordahl's cellular automata [31], Margenstern's non-erasing Turing machines [34,36], and Robinson's tiling [85].

3 Non-standard Universal Turing Machines: Time Efficiency and Program Size

So far we have been discussing results for universal Turing machines that have one tape, one tape head, and are deterministic (we often refer to this setup as

the *standard* model). Of course one can consider results for other variants of the model. There are many generalised models, for example allowing multiple tapes, multiple dimensions, or even coupling the Turing machine with a finite automaton. Restricted models include non-printing, non-erasing and reversible Turing machines, and machines with restricted instructions. In this section we explore program size and time complexity results for a number of generalised and restricted models. Table 2 contains program size results for a number of such non-standard machines.

Table 2. Small non-standard universal Turing machines. Semi-weak machines are denoted by †, weak machines by ‡, machines coupled with a finite automaton by ⋆, and a machine with 2 tape heads by △.

states	symbols	dimensions	tape	author
15	2	1	3	Moore [60]†
6	5	1	1	Watanabe [104]†
1	2	1	4	Hooper [21,22]†
2	3	1	2	Hooper [21,22]
7	3	1	1	Watanabe (mentioned in [105,70])†
5	4	1	1	Watanabe [105]†
8	4	2	1	Wagner [100]
2	7	2	1	Ottmann [73]‡
10	2	2	1	Ottmann [74,25]‡
6	3	2	1	Ottmann [74,25]‡
4	4	2	1	Ottmann [74,25]‡
2	6	2	1	Kleine-Büning & Ottmann [25]‡
2	5	2	1	Kleine-Büning & Ottmann [25]‡
2	3	2	1	Kleine-Büning & Ottmann [25]‡
1	7	3	1	Kleine-Büning & Ottmann [25]‡
4	5	2	1	Kleine-Büning & Ottmann [25]
3	6	2	1	Kleine-Büning & Ottmann [25]
10	2	2	1	Kleine-Büning [24]
2	5	2	1	Kleine-Büning [24]
2	4	2	1	Priese [82]
4	2	2	1	Gajardo et al. [15]
2	2	2	1	Priese [82]△
2	5	1	1	Margenstern & Pavlotskaya [45]⋆
4	7	1	1	Pavlotskaya [78]⋆
2	3	1	1	Margenstern & Pavlotskaya [46]⋆
7	2	1	1	Eppstein (published by Cook [9])‡
4	3	1	1	Cook [9] & Wolfram [107]‡
3	4	1	1	Cook [9] & Wolfram [107]‡
2	5	1	1	Cook [9] & Wolfram [107]‡
6	2	1	1	Neary & Woods [68]‡
3	3	1	1	Neary & Woods [68]‡
2	4	1	1	Neary & Woods [68]‡
3	7	1	1	Woods & Neary [111]†
4	5	1	1	Woods & Neary [111]†
2	13	1	1	Woods & Neary [111]†

3.1 Weak Universality and Rule 110

An interesting generalisation occurs when we stick to the standard conventions, but we allow the blank portion of the tape to contain a word, that is constant (independent of the input), and is repeated infinitely often in one direction, say to the left of the input. We say that such Turing machines are *semi-weakly universal*. Some of the earliest small universal Turing machines were semi-weak [104,105]. Sometimes another word is also repeated infinitely often to the right. Universal machines that use this setup are called *weakly universal* [43].

It is not difficult to see how this generalisation can help to reduce program size. For example, it is typical of small universal Turing machine simulations that the program being simulated is stored on the tape. When reading an instruction we often mark certain symbols. At a later time we then restore marked symbols to their original values. If the simulated program is repeated infinitely often, say to the left of the input, things may be much easier as we can simply skip the 'restore' phase of our algorithm and access a new copy of the program when simulating the next instruction, thus reducing the universal program's size.

This was the strategy used by Watanabe [104,105] to find the semi-weak, direct Turing machine simulators shown as hollow diamonds in Figure 1. Recently [111] we have given three new semi-weakly universal machines and these are shown as solid diamonds in Figure 1. These machines simulate cyclic tag systems [9]. It is interesting to note that two of our machines are symmetric with those of Watanabe (around the line where states = symbols), despite the fact that we use a different simulation technique. Our 4-state, 5-symbol machine has only 17 transition rules, making it the smallest known semi-weakly universal machine (Watanabe's 5-state, 4-symbol machine has 18 transition rules, and his 7-state, 3-symbol machine has 21 rules [105])[1]. The time overhead for these machines is polynomial. More precisely, if M is a single-tape deterministic Turing machine that runs in time t, then M is simulated by either of our semi-weak machines in time $O(t^4 \log^2 t)$. Watanabe's semi-weak machines also ran in polynomial time, with a very efficient time overhead of $O(t^2)$.

Cook, Eppstein, and Wolfram [9,107] gave weakly universal Turing machines that were significantly smaller than the existing semi-weak machines. These were improved upon by Neary and Woods [68] to give the smallest known weakly universal machines. In (states, symbols) notation their sizes are $(2, 4)$, $(3, 3)$ and $(6, 2)$, and they are illustrated in Figure 1. These machines work by simulating Rule 110, a very simple kind of cellular automaton. Rule 110 is an elementary cellular automaton, which means that it is a one-dimensional, nearest neighbour, binary cellular automaton [106]. More precisely, it is composed of a sequence of cells $\ldots p_{-1}p_0p_1 \ldots$ where each cell has a binary state $p_i \in \{0, 1\}$. At timestep $t + 1$ the value of cell $p_{i,t+1} = F(p_{i-1,t}, p_{i,t}, p_{i+1,t})$ is given by the synchronous local update function F

[1] Watanabe mentions that he found a $(7, 3)$ universal machine with 21 transition rules in reference [105]. We have not found the details of this machine, however the most reasonable inference from the literature is that it is semi-weakly universal.

$$\begin{aligned} F(0,0,0) &= 0 & F(1,0,0) &= 0 \\ F(0,0,1) &= 1 & F(1,0,1) &= 1 \\ F(0,1,0) &= 1 & F(1,1,0) &= 1 \\ F(0,1,1) &= 1 & F(1,1,1) &= 0 \end{aligned}$$

Rule 110 was shown to be universal via an impressive and detailed simulation of cyclic tag systems, the result is stated and described in [107] and the full proof is given in [9]. In the proof, the Rule 110 instance has a special (constant) word repeated infinitely to the left of the input, and another to the right. Rule 110 has a very simple update rule which facilitates the writing of very small weak Turing machines to simulate it.

As noted, Rule 110 was shown to be universal by simulating cyclic tag systems, which in turn simulate 2-tag systems. The chain of simulations included the exponentially slow 2-tag algorithm of Cocke and Minsky, thus Rule 110, and the weakly universal machines that simulate it, were exponentially slow. In a recent paper [64] we have improved their simulation time overhead to polynomial by showing that cyclic tag systems are efficient simulators of Turing machines. In doing so, we solved what Cook [10] has called the "geometry problem of cyclic-state tape processors." The difficult in overcoming this problem is that there is no obvious way for the system to efficiently determine which symbols or objects are adjacent to each other. Previous works used unary encodings as it was not obvious how to determine the relative positions of adjacent digits in a sequence. Our main result was in providing an efficient solution to this problem.

Our result has interesting implications for Rule 110. For example, given an initial configuration of Rule 110, and a value t in unary, predicting t timesteps of a Rule 110 computation is P-complete. Therefore, unless P = NC, which is widely believed to be false, we cannot hope to quickly (in polylogarithmic time) predict the evolution of this simple cellular automaton even if we have a polynomial amount of parallel hardware. Rule 110 is the simplest (one-dimensional, nearest neighbour) cellular automaton that has been shown to have a P-complete prediction problem. In particular, Ollinger's [71] intrinsic universality result already shows that prediction for one dimensional nearest neighbour cellular automata is P-complete for 6 states (later improved to 4 states by Richard and Ollinger [84,72]), and our result improves this to 2 states. The question of whether Rule 110 prediction is P-complete has been asked, directly or indirectly, in a number of previous works (for example [2,58,59]).

It is currently unknown whether all of the lower bounds in Figure 1 hold for weak machines. For example, the non-universality results of Pavlotskaya were proven for the case where one transition rule is reserved for halting, however the smallest weak machines do not halt.

3.2 Other Non-standard Universal Turing Machines

Weakness has not been the only generalisation on the standard model in the search for ever smaller universal machines. We give some notable examples here, many others are to be found in Table 2.

Before Shannon's famous paper, Moore [60] observed that 2-symbol machines were universal as any Turing machine could be converted into a 2-symbol machine by the (now) usual encoding. In the same paper Moore used this observation to give a universal 3-tape machine with 15 states and 2 symbols. Moore's machine uses only 57 instructions, each instruction being a sextuple that either moves one of its tape heads or prints a single symbol to one of its tapes. One of the tapes in Moore's 3-tape machine is circular and contains the simulated program, therefore his machine also operates correctly if the circular tape is replaced with a one-way infinite tape with a periodic background (i.e. semi-weak). Moore's result has been largely ignored in the literature despite being the first published small universal Turing machine. Interestingly, Moore's paper cites unpublished work by Shannon on the universality of non-erasing machines.

Hooper [21,22] gave universal machines with 2 states, 3 symbols and 2 tapes, and with 1 state, 2 symbols and 4 tapes. One of the tapes in Hooper's 4-tape machine is circular and contains the simulated program, and so could be replaced by a one-way infinite tape with a periodic background (i.e. semi-weak). Priese [82] gave a 2-state, 4-symbol machine with a 2-dimensional tape, and a 2-state, 2-symbol machine with 2 tape heads and a 2-dimensional tape. Margenstern and Pavlotskaya [46,45] gave a 2-state, 3-symbol Turing machine that uses only 5 instructions and is universal when coupled with a finite automaton. They also showed that the halting problem is decidable for such machines with 4 instructions [46].

3.3 Restricted Universal Turing Machines

If we suitably restrict the standard Turing machine model the problem of finding universal machines with small state-symbol products becomes more difficult. Over the years, a number of authors have looked at non-erasing Turing machines, that is machines that are permitted to overwrite blank symbols only. Moore [60] mentions that Shannon had proved that such non-erasing Turing machines simulate arbitrary Turing machines, however Shannon's work was never published. Shortly after, Shannon published a proof that 2-symbol Turing machines are universal, and Wang [101] proved that 2-symbol non-erasing Turing machines are universal. Later, Minsky proved the same result as Wang, but using the technique of simulation via non-writing Turing machines, yet another (universal) restriction [53].

Margenstern has examined the universality of 2-symbol Turing machines for a number of different restrictions. One such restriction is the number of colours of a machine, defined as the number of distinct triples (α, D, δ), where α is the read symbol, D is the move direction, and δ is the write symbol of a transition rule. Pavlotskaya [75,76] has shown that there are standard universal Turing machines with 3 colours and no standard universal Turing machines with 2 colours. Margenstern [34] has shown that there are non-erasing universal Turing machines with 5 colours and no non-erasing universal Turing machines with 4 colours.

Laterality number is another property examined by Margenstern. The laterality number of a Turing machine is defined as the minimum of the number of left move instructions and the number of right move instructions. Margenstern and Pavlotskaya [75,44] have shown that there are universal Turing machines with laterality number 2 and no universal machines with laterality number 1. Margenstern [36,39] has shown that there are universal non-erasing Turing machines with laterality number 3 and no universal non-erasing machines with laterality number 2. For more on these results see [33,34,35,36,37,42].

Fischer [12] gives a number of universality results for Turing machines that use restricted forms of transition rules. In one result he proves that 3-state Post machines are universal (Post machines [80] are like Turing machines, except that in a single timestep they can move or write, but not both). Interestingly, Aanderaa and Fischer [1] show that the halting problem for 2-state Post machines is decidable.

Bennett [5] has shown that 3-tape reversible Turing machines are universal. Morita and others have since shown universality results for reversible Turing machines with 1 tape and 2 symbols [61], and 17 states and 5 symbols [62].

3.4 Universal Turing Machines with Multidimensional Tapes: Time Efficiency and Program Size

During the 1970s a number of authors [82,25,100] were interested in finding small universal Turing machines with multidimensional tapes. The machines of these authors have not, to our knowledge, been analysed from the perspective of time/space complexity. We discuss this topic here.

Lutz Priese [82] gives a 2D machine with 2 states and 4 symbols that is universal on finite initial conditions (i.e. all except a finite number of symbols are initially blank), and another 2D machine with 2 states, 2 symbols and 2 tape heads that is derived from this 4-symbol machine. Priese's machines simulate counter machines (also called register machines), via a sequence of reductions. Given a counter machine that runs in time τ, Priese's machines simulate its computation in time $O(\tau^2)$ and space $O(\tau)$. Due to the unary encoding used by counter machines [13], both of Priese's machines simulate Turing machines with an exponential time overhead. Priese's machines do not end their computation in the conventional manner of halting on a state-symbol pair that has no transition rule: instead there is a choice, via the initial input encoding, of ending a computation either by entering a sequence of 6 repeating configurations or by halting when an attempt is made to move off the edge of the 2D tape.

Langton's ant [29] is usually described as an ant that lives on a 2D grid of binary-valued cells. The ant chooses which adjacent cell to move to based on (a) the current cell's binary value and (b) the ant's current orientation. The ant flips the current cell's bit as it moves away. So Langton's ant is a 2D Turing machine with 2 symbols and 4 states (North, South, East and West). Gajardo et al. [15] showed that predicting the behaviour of the ant is P-hard, by simulating

Boolean circuits in polynomial time. By then showing how the ant can simulate an infinite sized circuit (with a simple repeating structure), which in turn can simulate the space-time diagram of a cellular automata (CA), they prove that Langton's ant is weakly universal in 2D.

It is worth pausing to describe a form of weak universality in 2D, where the tape has a background that is ultimately periodic in both dimensions of single quadrant. A one-way infinite sequence is ultimately periodic [14] if it is of the form $s_1 s_2^\omega$ where $s_2^\omega = s^2 s^2 s^2 \ldots$, and s_1 and s_2 are finite sequences. We say that a $\mathbb{N} \times \mathbb{N}$ pattern is ultimately periodic in the x direction if for each $y \in \mathbb{N}$ the infinite sequence of symbols at the coordinates $(0, y), (1, y), (2, y), \ldots$ is ultimately periodic. This is defined analogously for the y direction.

Kleine-Büning and Ottmann [25] give universal Turing machines which have a single multidimensional tape, a number of which are weakly universal. Remarkably, their 2D, 2-state, 3-symbol machine does not even print to the tape! The two counter values of a simulated 2-counter machine are encoded by the (x, y) position of the tape head on the 2D tape. Testing for zero amounts to detecting one of the axes. It is well-known that 2-counter machines are universal [56]. However, using known algorithms, 2-counter machines suffer from a doubly-exponential slowdown when simulating Turing machines [97], and so the 2-state, 3-symbol machine of Kleine-Büning and Ottmann also suffers from a doubly-exponential slowdown when simulating Turing machines. We give a brief overview of this machine's computation.

The 2D tape uses only the upper-right quadrant of the plane and so each tape cell may be indexed by a coordinate of the form $(x, y) \in \mathbb{N} \times \mathbb{N}$. The quadrant is filled using 4, infinitely repeated, finite square blocks (of tape symbols) which we will call A, B, C, and D. The infinite pattern on the 2D tape given by the arrangement of these blocks is ultimately periodic in both the x and y directions. Each block is of size $O(r^2)$ where r is the number of instructions in the 2-counter machine being simulated. The block at the origin of the quadrant is of type A. Types B and C are repeated along along the x-axis and y-axis, respectively, and the remainder of the quadrant is tiled by blocks of type D. Each block encodes the entire program of the 2-counter machine being simulated. The current counter machine instruction being simulated is given by the position of the tape head within a block. If the counters have values x_1 and y_1 respectively, then the tape head will be in the x_1^{th} block from the y-axis and the y_1^{th} block from the x-axis. The blocks contain specially defined paths that the tape head follows to (a) arrive at the next counter machine instruction and (b) move to one of the adjacent blocks if a change in the value of a counter is being simulated. A, B and C blocks lie along the axes and so are used to simulate any instruction where one or more counters have value zero, and in particular they contain special paths that simulate a positive test for zero.

Kleine-Büning and Ottmann adapt their technique to give a non-printing 1-state, 7-symbol universal machine with a 3D tape. Only 2 planes in the third dimension are used, giving tape cells that are indexed by coordinates (x, y, z),

where $x, y \in \mathbb{N}$ and $z \in \{0, 1\}$. The pattern defined by the symbols on each of the infinite 2D planes given by $(x, y, 0)$ and $(x, y, 1)$ is ultimately periodic in both the x and y directions. The technique used to simulate 2-counter machines by the 1-state, 7-symbol machine is, in essence, the same as the technique use by the 2-state, 3-symbol machine. The 2D machine uses 2 states to remember which path it is following when two different paths cross (the tape head follows paths that encode instructions of the counter machine being simulated). With the introduction of a third dimension it is no longer necessary for paths to cross and so it is possible to give a universal Turing machine with only 1 state. Finally, we note that an immediate corollary of this machine's design is the existence of a non-halting universal machine with only 6 symbols, as the only purpose of one of the 7 symbols is to provide an undefined transition rule for halting.

It is a fairly straightforward matter to show that for each Turing machine with a single, ultimately periodic, 2D tape and no print instructions there is a 2-counter machine that simulates it in linear time. It immediately follows that improvement on the doubly-exponential time overhead when simulating Turing machines with such non-printing 2D machines is not possible unless such an improvement is also possible for 2-counter machines. Thus, it could be interesting to see if the simulation time overhead for such machines can be reduced to singly-exponential when a slightly more complicated background is permitted on the tape.

Wagner [100] shows that the halting problem for Turing machines with a single kD tape ($k \in \mathbb{N}$), 2 symbols and 2 states is decidable[2]. Specifically, he shows that if such machines halt then they do so in space $O(n)$, where n is input length. It is not difficult to give relevant decidability results (such as predicating looping or halting) for machines with a single 1D tape and non-printing instructions, even when an ultimately periodic background is permitted. Regarding kD machines, it can be shown for some classes of these machines that only weaker forms of universality are possible. For the case of kD non-printing machines, it is not difficult to give relevant decidability results when the initial tape contains only a finite number of non-blank symbols. Herman has shown that the halting problem is decidable for 1-state kD printing machines when all but a finite number of tape cells are blank at the start of each computation [20].

Though lacking in formal rigour, a comparison between the three 2D machines we discussed in this section poses some interesting questions about the possible trade-offs for different 2D models. For example, out of the three machines the 2-state, 3-symbol weak machine has the smallest state-symbol product, is the only non-writing machine, and the only machine that can halt. The 2-state, 4-symbol machine of Priese is the only machine of the three that does not use a periodic (weak) encoding, and the 4-state, 2-symbol machine of Gajardo et al. (Langton's ant) is the only machine of the three that simulates Turing machines in polynomial time. The best we can hope for with non-printing 2D machines is

[2] Machines using Wagner's definition end their computation with a simple loop: repeatedly executing a special transition rule that does not change the configuration. This is equivalent to executing a halting transition rule.

a singly exponential time overhead, but achieving even this bound would seem to be very tricky. It is interesting to note that the only non-weak 2D machine of the three, that of Priese, has an exponential time overhead when simulating Turing machines. This is not the case for the smallest non-weak 1D machines. It begs the question, is there a non-weak 2D machine with the same number of states and symbols as Priese's machine that is universal with a polynomial time overhead?

3.5 Termination of a Computation

As we hope has been made clear so far, it is vitally important to clearly specify the computational model one is using when trying to find small universal programs or give lower bounds on universal program size. In the absence of a clear model description and matching lower bounds, one can never claim to have found the "smallest" universal program. Throughout this work we have described results on upper bounds and lower bounds on universal program size and we have described how both change when the model definition changes. In this section we focus on one such issue: computation termination.

A number of authors have given universal Turing machines where successful computations do not end in a halt state. Many of the machines given in Table 2 are non-halting. What about the problem of proving relevant non-universality results for these models? Such non-universality results are not achievable by proving the halting problem decidable. Before we attempt such an endeavour we must agree on a clear definition of universal Turing machine. For example, instead of specifying the end of a computation by a single halting (or terminal) configuration, a computation could end with a specific sequence of configurations. We refer to this as a terminal configuration sequence. The output of the simulated Turing machine is retrieved by applying a recursive decoding function to the entire computation (also a configuration sequence). There are many ways to define terminal configuration sequence, some examples are:

- a configuration sequence that goes through a specified sequence of states,
- a configuration sequence that contains two identical configurations,
- a configuration that contains a specific subword.

Given a definition of a terminal configuration sequence we may prove that the terminal sequence problem (will a machine execute a terminal configuration sequence) is decidable. This gives non-universal lower bounds that are relevant to universal machines that end their computation with such a sequence. However, this result may not hold as a proof of non-universality if we subsequently alter our definition of terminal configuration sequence. One more general approach is to prove that the terminal sequence problem for all possible terminal sequences, of a machine or set of machines, is decidable. In any case, it is important to specify these details when giving upper and lower bounds on program size.

4 Busy Beavers

Besides small universal Turing machines, one finds small, yet complicated, programs in the busy beaver literature. The term busy beaver was introduced by Rado [83] who put forward a game where the goal for a given $k \in \mathbb{N}$ is to find, out of all the k-state, 2-symbol Turing machines, the machine that prints the most 1s and then halts when started on a blank tape. The busy beaver function $\Sigma : \mathbb{N} \to \mathbb{N}$ is then defined by letting $\Sigma(k)$ be the maximum number of 1's printed by any halting k-state, 2-symbol Turing machine. Busy beavers essentially adhere to the standard Turing machine model described in previous sections (one tape, one head, usual blank symbol, deterministic). It is known that $\Sigma(1) = 1$ (trivial), $\Sigma(2) = 4$ [83], $\Sigma(3) = 6$ [30], and $\Sigma(4) = 13$ [6]. However for 5 states or more the best we currently have are lower bounds. For example, Michel [50] cites $\Sigma(5) \geq 4098$ to Marxen and Buntrock [47], and $\Sigma(6) \geq 3.5 \times 10^{18267}$ to Pavel Kropitz. $S(k)$, the step-counting analogue of $\Sigma(k)$, is also considered. In fact, both Σ and S grow faster than any computable function [83]. Green [16] has given a lower bound on the growth of the function Σ.

The busy beaver problem has been generalised to machines with $\ell \geq 2$ symbols [7], where $\Sigma(k, \ell)$ is the largest number of non-zeros written by any k-state, ℓ-symbol Turing machine. It has been shown [7,28] that $\Sigma(2, 3) = 9$. Terry Ligocki and Shawn Ligocki have shown that $\Sigma(2, 4) \geq 2,050$ and $\Sigma(3, 3) \geq 374,676,383$, and have given lower bounds on a number of other state-symbol pairs. See Michel's survey [50] for more results.

Although finding busy beavers is somewhat orthogonal to the goal of finding small universal Turing machines, there are potential connections between the two fields. On the one hand, when designing small universal programs one often has to reuse instructions in many different contexts, something which busy beavers might also do, so perhaps small instruction sets from one field might be useful for the other. On the other hand, proving lower bounds on universal program size, and upper bounds on values for the busy beaver function, both involve hefty case analyses so once again techniques developed in one field could potentially be useful for the other. In particular, the search for busy beavers has produced small programs with very complicated behaviour, which lend weight to the idea that proving non-universality of such program classes might be difficult.

5 Further Work

There are many avenues for further work, here we highlight a few examples.

Applying computational complexity theory to the area of small universal Turing machines allows us to ask a number of questions that are more subtle than the usual questions about program size. As we move towards the origin in Figure 1, the universal machines have larger (but polynomial) time overheads. Can the time overheads in Figure 1 be further improved (lowered)? Can we prove lower bounds on the simulation time of machines with a given state-symbol pair? Proving non-trivial simulation time lower bounds seems like a difficult problem.

Such results could be used to prove that there is a polynomial trade-off between simulation time and universal program size.

As we move away from the origin, the non-universal machines seem to have more power. For example Kudlek's classification of 2-state, 2-symbol machines shows that the sets accepted by these machines are regular, with the exception of one context free language ($a^n b^n$). Can we hope to fully characterise the sets accepted by non-universal machines (e.g. in terms of complexity or automata theoretic classes) with given state-symbol pairs or other program restrictions?

When discussing the complexity of small machines the issue of encodings becomes very important. For example, when proving that the prediction problem for a small machine is P-complete [17], the relevant encodings should be in logspace, and this is the case for all of the polynomial time machines in Figure 1.

Of course there are many models of computation that we have not mentioned where researchers have focused on finding small universal programs. Post's [79] tag systems are an interesting example. Minsky [52,53] showed that tag systems are universal with deletion number 6. Cocke and Minsky lowered the deletion number to 2, by showing that 2-tag systems were universal. They used productions (appendants) of length at most 4. Wang [102] further lowered the production length to 3. Recently, De Mol [11] has given a lower bound by showing that the reachability (and thus halting) problems are decidable for 2-tag systems with 2 symbols; a problem which Post claimed [81] to have solved but never published. It would be interesting to find the smallest universal tag systems in terms of number of symbols, deletion length, and production length.

The space between the non-universal curve and the smallest non-weakly universal machines in Figure 1 contains some complicated beasts. These lend weight to the feeling that finding new lower bounds on universal program size is tricky. Most noteworthy are the weakly and semi-weakly universal machines discussed earlier. Table 2 highlights that the existence of general models that provably have less states and symbols than the standard universal machines can have (for example the machines with (states, symbols, dimensions, tapes) of (2,3,2,1) [25], (1,7,3,1) [25], and (1,2,1,4) [22]). Also of importance are the busy beavers [50] and small machines of Margenstern [40,41], Baiocchi [3], and Michel [48,49] that live in this region and simulate iterations of the $3x + 1$ problem and other Collatz-like functions. So it seems that there are plenty of animals yet to be tamed.

References

1. Aanderaa, S., Fischer, P.C.: The solvability of the halting problem for 2-state post machines. Journal of the Association for Computing Machinery 14(4), 677–682 (1967)
2. Aaronson, S.: Book review: A new kind of science. Quantum Information and Computation 2(5), 410–423 (2002)
3. Baiocchi, C.: 3N+1, UTM e tag-system. Technical Report Pubblicazione 98/38, Dipartimento di Matematico, Università di Roma (1998) (in Italian)

4. Baiocchi, C.: Three Small Universal Turing Machines. In: Margenstern, M., Rogozhin, Y. (eds.) MCU 2001. LNCS, vol. 2055, pp. 1–10. Springer, Heidelberg (2001)
5. Bennett, C.H.: Logical reversibility of computation. IBM Journal of Research and Development 17(6), 525–532 (1973)
6. Brady, A.H.: The determination of the value of Rado's noncomputable function $\Sigma(k)$ for four-state Turing machines. Mathematics of Computation 40(163), 647–665 (1983)
7. Brady, A.H.: The busy beaver game and the meaning of life. In: Herken, R. (ed.) The Universal Turing Machine: A Half-Century Survey, pp. 259–277. Oxford University Press (1988)
8. Cocke, J., Minsky, M.: Universality of tag systems with $P = 2$. Journal of the Association for Computing Machinery 11(1), 15–20 (1964)
9. Cook, M.: Universality in elementary cellular automata. Complex Systems 15(1), 1–40 (2004)
10. Cook, M.: A Concrete View of Rule 110 Computation. Electronic Proceedings in Theoretical Computer Science 1, 31–55 (2009)
11. De Mol, L.: Study of Limits of Solvability in Tag Systems. In: Durand-Lose, J., Margenstern, M. (eds.) MCU 2007. LNCS, vol. 4664, pp. 170–181. Springer, Heidelberg (2007)
12. Fischer, P.C.: On formalisms for Turing machines. Journal of the Association for Computing Machinery 12(4), 570–580 (1965)
13. Fischer, P.C., Meyer, A.R., Rosenberg, A.L.: Counter machines and counter languages. Mathematical Systems Theory 2(3), 265–283 (1968)
14. Friedman, J.: A decision procedure for computations of finite automata. Journal of the ACM 9(3), 315–323 (1962)
15. Gajardo, A., Moreira, A., Goles, E.: Complexity of Langton's ant. Discrete Applied Mathematics 117, 41–50 (2002)
16. Green, M.W.: A lower bound on Rado's sigma function for binary Turing machines. In: Proceedings of the 5th IEEE Annual Symposium on Switching Circuit Theory and Logical Design, pp. 91–94 (1964)
17. Greenlaw, R., Hoover, H.J., Ruzzo, W.L.: Limits to parallel computation: P-completeness theory. Oxford university Press, Oxford (1995)
18. Harju, T., Margenstern, M.: Splicing Systems for Universal Turing Machines. In: Ferretti, C., Mauri, G., Zandron, C. (eds.) DNA 2004. LNCS, vol. 3384, pp. 149–158. Springer, Heidelberg (2005)
19. Hermann, G.T.: The uniform halting problem for generalized one state Turing machines. In: Proceedings of the Ninth Annual Symposium on Switching and Automata Theory (FOCS), pp. 368–372. IEEE Computer Society Press, Schenectady (1968)
20. Hermann, G.T.: The halting problem of one state Turing machines with n-dimensional tape. Mathematical Logic Quarterly 14(7-12), 185–191 (1968)
21. Hooper, P.: Some small, multitape universal Turing machines. Technical report, Computation Laboratory, Harvard University, Cambridge, Massachusetts (1963)
22. Hooper, P.: Some small, multitape universal Turing machines. Information Sciences 1(2), 205–215 (1969)
23. Ikeno, N.: A 6-symbol 10-state universal Turing machine. In: Proceedings, Institute of Electrical Communications, Tokyo (1958)
24. Kleine-Büning, H.: Über probleme bei homogener Parkettierung von $\mathbb{Z} \times \mathbb{Z}$ durch Mealy-automaten bei normierter verwendung. PhD thesis, Institut für Mathematische Logik, Münster (1977)

25. Kleine-Büning, H., Ottmann, T.: Kleine universelle mehrdimensionale Turingmaschinen. Elektronische Informationsverarbeitung und Kybernetik 13(4-5), 179–201 (1977) (in German)
26. Kudlek, M.: Small deterministic Turing machines. Theoretical Computer Science 168(2), 241–255 (1996)
27. Kudlek, M., Rogozhin, Y.: A Universal Turing Machine with 3 States and 9 Symbols. In: Kuich, W., Rozenberg, G., Salomaa, A. (eds.) DLT 2001. LNCS, vol. 2295, pp. 311–318. Springer, Heidelberg (2002)
28. Lafitte, G., Papazian, C.: The fabric of small Turing machines. In: Computation and Logic in the Real World, Third Conference on Computability in Europe, CiE 2007, Local Proceedings, pp. 219–227 (2007)
29. Langton, C.: Studying artificial life with cellular automata. Physica D 2(1-3), 120–149 (1986)
30. Lin, S., Rado, T.: Computer studies of Turing machine problems. Journal of the ACM 12(2), 196–212 (1965)
31. Lindgren, K., Nordahl, M.G.: Universal computation in simple one-dimensional cellular automata. Complex Systems 4(3), 299–318 (1990)
32. Margenstern, M.: Surprising areas in the quest for small universal devices. Electronic Notes in Theoretical computer Science 225, 201–220 (2009)
33. Margenstern, M.: Sur la frontière entre machines de Turing á arrêt décidable et machines de Turing universelles. Technical Report 92-83, LITP Institut Blaise Pascal (1992)
34. Margenstern, M.: Non-erasing Turing Machines: A Frontier Between a Decidable Halting Problem and Universality. In: Ésik, Z. (ed.) FCT 1993. LNCS, vol. 710, pp. 375–385. Springer, Heidelberg (1993)
35. Margenstern, M.: Une machine de Turing universelle sur {0,1}, non-effaçante et à trois instructions gauches. Technical Report 94-08, LITP Institut Blaise Pascal (1994)
36. Margenstern, M.: Non-Erasing Turing Machines: A New Frontier Between a Decidable Halting Problem and Universality. In: Baeza-Yates, R.A., Poblete, P.V., Goles, E. (eds.) LATIN 1995. LNCS, vol. 911, pp. 386–397. Springer, Heidelberg (1995)
37. Margenstern, M.: Une machine de Turing universelle non-effaçante à exactement trois instructions gauches. C. R. Acad. Sci. Paris, Série I 320, 101–106 (1995)
38. Margenstern, M.: Decidability and Undecidability of the Halting Problem on Turing Machines, a Survey. In: Adian, S., Nerode, A. (eds.) LFCS 1997. LNCS, vol. 1234, pp. 226–236. Springer, Heidelberg (1997)
39. Margenstern, M.: The laterality problem for non-erasing Turing machines on $\{0, 1\}$ is completely solved. Theoretical Informatics and Applications 31(2), 159–204 (1997)
40. Margenstern, M.: Frontier between decidability and undecidability: a survey. In: Margenstern, M. (ed.) Machines, Computations and Universality (MCU), France, IUT, Metz., vol. 1, pp. 141–177 (1998)
41. Margenstern, M.: Frontier between decidability and undecidability: a survey. Theoretical Computer Science 231(2), 217–251 (2000)
42. Margenstern, M.: On quasi-unilateral universal Turing machines. Theoretical Computer Science 257(1-2), 153–166 (2001)
43. Margenstern, M.: An algorithm for building intrinsically universal cellular automata in hyperbolic spaces. In: Proceedings of the 2006 International Conference on Foundations of Computer Science (FCS), Las Vegas, NV, pp. 3–9. CSREA Press (2006)

44. Margenstern, M., Pavlotskaya, L.: Deux machines de Turing universelles á au plus deux instructions gauches. C. R. Acad. Sci. Paris, Série I 320, 1395–1400 (1995)
45. Margenstern, M., Pavlotskaya, L.: Vers ue nouvelle approche de l'universalité concernant les machines de Turing. Technical Report 95-58, LITP Institut Blaise Pascal (1995)
46. Margenstern, M., Pavlotskaya, L.: On the optimal number of instructions for universality of Turing machines connected with a finite automaton. International Journal of Algebra and Computation 13(2), 133–202 (2003)
47. Marxen, H., Buntrock, J.: Attacking the Busy Beaver 5. Bulletin of the EATCS 40, 247–251 (1990)
48. Michel, P.: Busy beaver competition and Collatz-like problems. Archive Mathematical Logic 32(5), 351–367 (1993)
49. Michel, P.: Small Turing machines and generalized busy beaver competition. Theoretical Computer Science 326, 45–56 (2004)
50. Michel, P.: The busy beaver competition: a historical survey, arXiv:0906.3749v2 (2010), http://www.logique.jussieu.fr/~michel/ha.html
51. Minsky, M.: A 6-symbol 7-state universal Turing machines. Technical Report 54-G-027, MIT (1960)
52. Minsky, M.: Recursive unsolvability of Post's tag problem. Technical Report 54-G-023, Massachusetts Institute of Technology (1960)
53. Minsky, M.: Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. Annals of Mathematics 74(3), 437–455 (1961)
54. Minsky, M.: Size and structure of universal Turing machines using tag systems. In: Recursive Function Theory: Proceedings, Symposium in Pure Mathematics, Provelence, vol. 5, pp. 229–238 (1962)
55. Minsky, M.: Universality of (p=2) tag systems and a 4 symbol 7 state universal Turing machine. In: AIM-33, A.I. memo 33, Computer Science and Artificial Intelligence Laboratory. MIT, Cambridge (1962)
56. Minsky, M.: Computation, finite and infinite machines. Prentice-Hall, Englewood Cliffs (1967)
57. Moore, C., Mertens, S.: The Nature of Computation. Oxford University Press (2011)
58. Moore, C.: Quasi-linear cellular automata. Physica D 103, 100–132 (1997)
59. Moore, C.: Predicting non-linear cellular automata quickly by decomposing them into linear ones. Physica D 111, 27–41 (1998)
60. Moore, E.F.: A simplified universal Turing machine. In: ACM National Meeting, Toronto, Canada, pp. 50–54. ACM Press (1952)
61. Morita, K., Shirasaki, A., Gono, Y.: A 1-tape 2-symbol reversible Turing machine. The Transactions of the IEICE Japan E72(3), 223–228 (1989)
62. Morita, K., Yamaguchi, Y.: A Universal Reversible Turing Machine. In: Durand-Lose, J., Margenstern, M. (eds.) MCU 2007. LNCS, vol. 4664, pp. 90–98. Springer, Heidelberg (2007)
63. Neary, T.: Small universal Turing machines. PhD thesis, National University of Ireland, Maynooth (2008)
64. Neary, T., Woods, D.: P-Completeness of Cellular Automaton Rule 110. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006, Part I. LNCS, vol. 4051, pp. 132–143. Springer, Heidelberg (2006)
65. Neary, T., Woods, D.: Small fast universal Turing machines. Theoretical Computer Science 362(1-3), 171–195 (2006)

66. Neary, T., Woods, D.: Four Small Universal Turing Machines. In: Durand-Lose, J., Margenstern, M. (eds.) MCU 2007. LNCS, vol. 4664, pp. 242–254. Springer, Heidelberg (2007)
67. Neary, T., Woods, D.: Small weakly universal Turing machines, arXiv:0707.4489v1 [cs.CC] (2007)
68. Neary, T., Woods, D.: Small Weakly Universal Turing Machines. In: Kutyłowski, M., Charatonik, W., Gębala, M. (eds.) FCT 2009. LNCS, vol. 5699, pp. 262–273. Springer, Heidelberg (2009)
69. Neary, T., Woods, D.: Four small universal Turing machines. Fundamenta Informaticae 91(1), 123–144 (2009)
70. Nozaki, A.: On the notion of universality of Turing machine. Kybernetika Academia Praha 5(1), 29–43 (1969) (English translated version)
71. Ollinger, N.: The Quest for Small Universal Cellular Automata. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 318–329. Springer, Heidelberg (2002)
72. Ollinger, N., Richard, G.: Four states are enough! Theoretical Computer Science 412(1-2), 22–32 (2011)
73. Ottmann, T.: Eine universelle Turingmaschine mit zweidimensionalem band. Elektronische Informationsverarbeitung und Kybernetik 11(1-2), 27–38 (1975) (in German)
74. Ottmann, T.: Einfache universelle mehrdimensionale Turingmaschinen. Habilitationsschrift, Karlsruhe (1975)
75. Pavlotskaya, L.: Solvability of the halting problem for certain classes of Turing machines. Mathematical Notes 13(6), 537–541; Translated from Matematicheskie Zametki, 13(6), 899–909 (1973)
76. Pavlotskaya, L.: O minimal'nom chisle razlichnykh kodov vershin v grafe universal'noj mashiny T'juringa. Disketnyj Analiz, Sbornik Trudov Instituta Matematiki SO AN SSSR 27, 52–60 (1975); On the minimal number of distinct codes for the vertices of the graph of a universal Turing machine (in Russian)
77. Pavlotskaya, L.: Dostatochnye uslovija razreshimosti problemy ostanovki dlja mashin T'juring. Avtomaty i Mashiny, 91–118 (1978); Sufficient conditions for the halting problem decidability of Turing machines (in Russian)
78. Pavlotskaya, L.: On machines, universal by extensions. Theoretical Computer Science 168(2), 257–266 (1996)
79. Post, E.L.: Formal reductions of the general combinatorial decision problem. American Journal of Mathmatics 65(2), 197–215 (1943)
80. Post, E.L.: Recursive unsolvability of a problem of Thue. Journal of Symbolic Logic 12(1), 1–11 (1947)
81. Post, E.L.: Absolutely unsolvable problems and relatively undecidable propositions – account of an anticipation. In: Davis, M. (ed.) The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions, pp. 340–406. Raven Press, New York (1965); Corrected republication. Dover publications, New York (2004)
82. Priese, L.: Towards a precise characterization of the complexity of universal and nonuniversal Turing machines. SIAM J. Comput. 8(4), 508–523 (1979)
83. Rado, T.: On non-computable functions. Bell System Technical Journal 41(3), 877–884 (1962)
84. Richard, G.: A particular universal cellular automaton, HAL research report (oai:hal.archives-ouvertes.fr:hal-00095821_v1) (2006)
85. Robinson, R.M.: Undecidability and nonperiodicity for tilings of the plane. Inventiones Mathematicae 12(3), 177–209 (1971)

86. Robinson, R.M.: Minsky's small universal Turing machine. International Journal of Mathematics 2(5), 551–562 (1991)
87. Rogozhin, Y.: Sem' universal'nykh mashin T'juringa. In: Fifth All Union Conference on Mathematical Logic, Akad. Naul SSSR. Otdel. Inst. Mat., Novosibirsk, p. 27 (1979); Seven universal Turing machines (in Russian)
88. Rogozhin, Y.: Sem' universal'nykh mashin T'juringa. Systems and Theoretical Programming, Mat. Issled 69, 76–90 (1982); Seven universal Turing machines (in Russian)
89. Rogozhin, Y.: Universal'naja mashina T'juringa s 10 sostojanijami i 3 simvolami. Izvestiya Akademii Nauk Respubliki Moldova, Matematika 4(10), 80–82 (1992); A universal Turing machine with 10 states and 3 symbols (in Russian)
90. Rogozhin, Y.: About Shannon's problem for Turing machines. Computer Science Journal of Moldova 1(3), 108–111 (1993)
91. Rogozhin, Y.: Small universal Turing machines. Theoretical Computer Science 168(2), 215–240 (1996)
92. Rogozhin, Y.: A universal Turing machine with 22 states and 2 symbols. Romanian Journal of Information Science and Technology 1(3), 259–265 (1998) (in Russian)
93. Rogozhin, Y., Verlan, S.: On the Rule Complexity of Universal Tissue P Systems. In: Freund, R., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2005. LNCS, vol. 3850, pp. 356–362. Springer, Heidelberg (2006)
94. Rothemund, P.W.K.: A DNA and restriction enzyme implementation of Turing Machines. In: Lipton, R.J., Baum, E.B. (eds.) DNA Based Computers: Proceeding of a DIMACS Workshop. DIMACS, vol. 2055, pp. 75–119. Princeton University, AMS (1996)
95. Shannon, C.E.: A universal Turing machine with two internal states. Automata Studies, Annals of Mathematics Studies 34, 157–165 (1956)
96. Siegelmann, H.T., Margenstern, M.: Nine switch-affine neurons suffice for Turing universality. Neural Networks 12(4-5), 593–600 (1999)
97. Schroeppel, R.: A two counter machine cannot calculate 2^n. Technical Report AIM-257, A.I. memo 257, Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA (1972)
98. Takahashi, H.: Keisankikai II. Iwanami, Tokyo (1958); Computing machine II (in Japanese)
99. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society 2(42), 230–265 (1936)
100. Wagner, K.: Universelle Turingmaschinen mit n-dimensionale band. Elektronische Informationsverarbeitung und Kybernetik 9(7-8), 423–431 (1973); Universal Turing machines with n-dimensional tapes (in German)
101. Wang, H.: A variant to Turing's theory of computing machines. Journal of the Association for Computing Machinery 4(1), 63–92 (1957)
102. Wang, H.: Tag systems and lag systems. Mathematical Annals 152(4), 65–74 (1963)
103. Watanabe, S.: On a minimal universal Turing machine. Technical report, MCB Report, Tokyo (1960)
104. Watanabe, S.: 5-symbol 8-state and 5-symbol 6-state universal Turing machines. Journal of the ACM 8(4), 476–483 (1961)
105. Watanabe, S.: 4-symbol 5-state universal Turing machine. Information Processing Society of Japan Magazine 13(9), 588–592 (1972)

106. Wolfram, S.: Statistical mechanics of cellular automata. Reviews of Modern Physics 55(3), 601–644 (1983)
107. Wolfram, S.: A new kind of science. Wolfram Media, Inc. (2002)
108. Woods, D., Neary, T.: On the time complexity of 2-tag systems and small universal Turing machines. In: 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 439–446. IEEE, Berkeley (2006)
109. Woods, D., Neary, T.: Small Semi-Weakly Universal Turing Machines. In: Durand-Lose, J., Margenstern, M. (eds.) MCU 2007. LNCS, vol. 4664, pp. 303–315. Springer, Heidelberg (2007)
110. Woods, D., Neary, T.: The complexity of small universal Turing machines: A survey. Theoretical Computer Science 410(4-5), 443–450 (2009)
111. Woods, D., Neary, T.: Small semi-weakly universal Turing machines. Fundamenta Informaticae 91(1), 179–195 (2009)

A Sufficient Condition for Sets Hitting the Class of Read-Once Branching Programs of Width 3 (Extended Abstract)*

Jiří Šíma and Stanislav Žák

Institute of Computer Science, Academy of Sciences of the Czech Republic,
P. O. Box 5, 18207 Prague 8, Czech Republic
{sima,stan}@cs.cas.cz

Abstract. We characterize the hitting sets for read-once branching programs of width 3 by a so-called richness condition which is independent of a rather technical definition of branching programs. The richness property proves to be (in certain sense) necessary and sufficient condition for such hitting sets. In particular, we show that any rich set extended with all strings within Hamming distance of 3 is a hitting set for width-3 read-once branching programs. Applying this result to an example of an efficiently constructible rich set from our previous work we achieve an explicit polynomial time construction of an ε-hitting set for read-once branching programs of width 3 with acceptance probability $\varepsilon > 11/12$.

1 Introduction

An *ε-hitting set* for a class of Boolean functions of n variables is a set $H \subseteq \{0,1\}^n$ such that for every function f in the class, if a random input is accepted by f with probability at least ε, then there is also an input in H that is accepted by f. Looking for polynomial time constructions of hitting sets for functions of polynomial complexity in different models such as circuits, formulas, branching programs which would have consequences for the relationship between respective deterministic and probabilistic computations belongs to the hardest problems in computer science, and hence, restricted models are investigated. An efficiently constructible sequence of hitting sets for increasing n is a straightforward generalization of the *hitting set generator* introduced in [9], which is a weaker (one-sided error) version of *pseudorandom generator* [13].

We consider *read-once branching (1-branching) programs* of polynomial size, which is a restricted model of space-bounded computations [17] for which pseudorandom generators with seed length $O(\log^2 n)$ have been known for a long time through Nisan's result [12]. Recently, considerable attention has been paid to improving this to $O(\log n)$ in the constant-width case, which is a fundamental problem with many applications in circuit lower bounds and derandomization [11].

* This research was partially supported by projects GA ČR P202/10/1333, MŠMT ČR 1M0545, and AV0Z10300504.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 406–418, 2012.

The problem has been resolved for width 2 but already for width 3 the issue was reported to be widely open as the known techniques provably fail [3,5,6,8,11].

In the case of width 3, we do not know of any significant improvement over Nisan's result except for severely restricted so-called regular or permutation oblivious 1-branching programs. Recall that an *oblivious* branching program queries the input variables in a fixed order, which represents a provably weaker model [2]. For constant-width *regular* oblivious 1-branching programs which have the in-degree of all nodes equal to 2 (or 0), pseudorandom generators have recently been constructed with seed length $O(\log n(\log\log n+\log(1/\varepsilon)))$ [4,5] which was further improved to $O(\log n \log(1/\varepsilon))$ [6], where ε is the error of generators. Moreover, for constant-width *permutation* oblivious 1-branching programs which are regular programs with the two edges incoming to any node labeled 0 and 1, the same seed length was previously achieved [10].

In the constant-width regular 1-branching programs the fraction of inputs that are queried at any node is always lower bounded by a positive constant, which excludes the fundamental capability of general (non-regular) branching programs to recognize the inputs that contain a given substring on a non-constant number of selected positions. In our approach, we manage the analysis also for this essential case by identifying two types of convergence of the number of inputs along a computational path towards zero which implement read-once DNFs and CNFs, respectively. Thus, we achieve the construction of a hitting set generator for general width-3 1-branching programs which need not be regular nor oblivious. In our previous work [14], we constructed the hitting set for so-called *simple* width-3 1-branching programs which exclude one specific pattern of level-to-level transition in their normalized form and cover the width-3 regular case.

In this extended abstract (for a full presentation see [15]), we formulate a so-called *richness* condition (Section 2) which is independent of a rather technical definition of branching programs. In fact, a rich set is a hitting set for read-once conjunctions of a DNF and a CNF. Thus, a related line of study concerns pseudorandom generators for read-once formulas, such as read-once DNFs [7]. We show that the richness property characterizes the hitting sets for width-3 1-branching programs. In particular, a weaker version of the richness condition proves to be necessary for such hitting sets, while the sufficiency of richness represents the main result of this paper. More precisely, we show that any rich set extended with all strings within Hamming distance of 3 is a hitting set for width-3 1-branching programs. The proof which is based on a detailed analysis of structural properties of the width-3 1-branching programs that reject all the inputs from the candidate hitting set is sketched in Sections 3–5.

The presented characterization of hitting sets by the richness property is of independent interest since it opens the possibility of generalizing this condition to more complicated read-once formulas in the constant-width case. In our (chronologically later) related paper [16], we proved that any almost $O(\log n)$-wise independent set, which can be constructed in polynomial time [1], is an example of the rich set (i.e. the hitting set for read-once conjunctions of DNF and CNF).

Combining this example with the sufficiency of the richness condition we achieve an explicit polynomial time construction of an ε-hitting set for 1-branching programs of width 3 with acceptance probability $\varepsilon > 11/12$ (Section 6).

We start with a brief review of basic formal definitions regarding branching programs [17]. A *branching program* P on the set of input Boolean variables $X_n = \{x_1, \ldots, x_n\}$ is a directed acyclic multi-graph $G = (V, E)$ that has one *source* $s \in V$ of zero in-degree and, except for *sinks* of zero out-degree, all the *inner* (non-sink) nodes have out-degree 2. In addition, the inner nodes get labels from X_n and the sinks get labels from $\{0, 1\}$. For each inner node, one of the outgoing edges gets the label 0 and the other one gets the label 1. The branching program P computes Boolean function $P : \{0,1\}^n \longrightarrow \{0,1\}$ as follows. The computational path of P for an input $\mathbf{a} = (a_1, \ldots, a_n) \in \{0,1\}^n$ starts at source s. At any inner node labeled by $x_i \in X_n$, input variable x_i is tested and this path continues with the outgoing edge labeled by a_i to the next node, which is repeated until the path reaches the sink whose label gives the output value $P(\mathbf{a})$. Denote by $P^{-1}(a) = \{\mathbf{a} \in \{0,1\}^n \mid P(\mathbf{a}) = a\}$ the set of inputs for which P outputs $a \in \{0, 1\}$. For inputs of arbitrary lengths, infinite families $\{P_n\}$ of branching programs, one P_n for each input length $n \geq 1$, are used. A branching program P is called *read-once* (or shortly *1-branching* program) if every input variable from X_n is tested at most once along each computational path. Here we consider *leveled* branching programs in which each node belongs to a level, and edges lead from level $k \geq 0$ to the next level $k+1$ only. We assume that the source of P creates level 0, whereas the last level is composed of all sinks. The number of levels decreased by 1 equals the *depth* of P which is the length of its longest path, and the maximum number of nodes on one level is called the *width* of P.

In the sequel, we confine ourselves to the 1-branching programs of width 3, for which we define 3×3 *transition matrix* T_k on level $k \geq 1$ such that $t_{ij}^{(k)} \in \{0, \frac{1}{2}, 1\}$ is the half of the number of edges leading from node $v_j^{(k-1)}$ $(1 \leq j \leq 3)$ on level $k-1$ to node $v_i^{(k)}$ $(1 \leq i \leq 3)$ on level k. For example, $t_{ij}^{(k)} = 1$ implies there is a *double edge* from $v_j^{(k-1)}$ to $v_i^{(k)}$. Denote by a column vector $\mathbf{p}^{(k)} = (p_1^{(k)}, p_2^{(k)}, p_3^{(k)})^\mathsf{T}$ the *distribution* of inputs among 3 nodes on level k of P, that is, $p_i^{(k)}$ equals the ratio of the number of inputs from $M(v_i^{(k)}) \subseteq \{0,1\}^n$ that are tested at $v_i^{(k)}$ to all 2^n possible inputs. It follows $M(v_1^{(k)}) \cup M(v_2^{(k)}) \cup M(v_3^{(k)}) = \{0,1\}^n$ and $p_1^{(k)} + p_2^{(k)} + p_3^{(k)} = 1$ for every level $k \geq 0$. Given the distribution $\mathbf{p}^{(k-1)}$ on level $k-1$, the distribution on the subsequent level k can be computed using transition matrix T_k as $\mathbf{p}^{(k)} = T_k \cdot \mathbf{p}^{(k-1)}$. We say that a 1-branching program P of width 3 is *normalized* if P has the minimum depth among the programs computing the same function and P satisfies $1 > p_1^{(k)} \geq p_2^{(k)} \geq p_3^{(k)} > 0$ for every $k \geq 2$. Any width-3 1-branching program can be normalized by permuting its nodes at each level [14]. Obviously, any normalized P satisfies $p_1^{(k)} > \frac{1}{3}$, $p_2^{(k)} < \frac{1}{2}$, and $p_3^{(k)} < \frac{1}{3}$ for every level $2 \leq k \leq d$ where $d \leq n$ is the depth of P.

2 The Richness Condition

Let $\mathcal{P}$ be the class of read-once branching programs of width 3 and $\varepsilon > 0$ be a real constant. A set of input strings $H \subseteq \{0,1\}^*$ is called an *ε-hitting set* for class $\mathcal{P}$ if for sufficiently large n, for every branching program $P \in \mathcal{P}$ with n inputs

$$\left|P^{-1}(1)\right| / 2^n \geq \varepsilon \quad \text{implies} \quad (\exists\, \mathbf{a} \in H \cap \{0,1\}^n)\, P(\mathbf{a}) = 1\,. \tag{1}$$

We say that a set $A \subseteq \{0,1\}^*$ is *weakly ε-rich* if for sufficiently large n, for any index set $I \subseteq \{1,\ldots,n\}$, and for any partition $\{Q_1,\ldots,Q_q,R_1,\ldots,R_r\}$ of I, if

$$\left(1 - \prod_{j=1}^{q}\left(1 - 1/2^{|Q_j|}\right)\right) \times \prod_{j=1}^{r}\left(1 - 1/2^{|R_j|}\right) \geq \varepsilon\,, \tag{2}$$

then for any $\mathbf{c} \in \{0,1\}^n$ there exists $\mathbf{a} \in A \cap \{0,1\}^n$ that meets

$$(\exists\, j \in \{1,\ldots,q\})(\forall\, i \in Q_j)\, a_i = c_i \;\&\; (\forall\, j \in \{1,\ldots,r\})\,(\exists\, i \in R_j)\, a_i \neq c_i\,. \tag{3}$$

Note that the product on the left-hand side of inequality (2) expresses the probability that a random string $\mathbf{a} \in \{0,1\}^n$ (not necessarily in A) satisfies condition (3). Moreover, formula (3) can be interpreted as a read-once conjunction of a DNF and a CNF (each variable occurs at most once)

$$\bigvee_{j=1}^{q} \bigwedge_{i \in Q_j} \ell(x_i) \wedge \bigwedge_{j=1}^{r} \bigvee_{i \in R_j} \neg\ell(x_i)\,, \quad \text{where} \quad \ell(x_i) = \begin{cases} x_i & \text{for } c_i = 1 \\ \neg x_i & \text{for } c_i = 0 \end{cases} \tag{4}$$

which accepts a random input with probability at least ε according to (2). Hence, a weakly rich set A is a hitting set for read-once conjunctions of DNF and CNF. The following theorem shows that the weak richness is necessary for any set to be a hitting set for width-3 1-branching programs.

Theorem 1 ([15]). *Every ε-hitting set for the class of read-once branching programs of width 3 is weakly ε-rich.*

Furthermore, a set $A \subseteq \{0,1\}^*$ is *ε-rich* if for sufficiently large n, for any index set $I \subseteq \{1,\ldots,n\}$, for any partition $\{R_1,\ldots,R_r\}$ of I ($r \geq 0$) satisfying

$$\prod_{j=1}^{r}\left(1 - 1/2^{|R_j|}\right) \geq \varepsilon\,, \tag{5}$$

and for any $Q \subseteq \{1,\ldots,n\} \setminus I$ such that $|Q| \leq \log n$, for any $\mathbf{c} \in \{0,1\}^n$ there exists $\mathbf{a} \in A \cap \{0,1\}^n$ that meets

$$(\forall\, i \in Q)\, a_i = c_i \text{ and } (\forall\, j \in \{1,\ldots,r\})\,(\exists\, i \in R_j)\, a_i \neq c_i\,. \tag{6}$$

One can observe that any ε-rich set is weakly ε-rich since condition (2) implies that there is $j \in \{1,\ldots,q\}$ such that $|Q_j| \leq \log n$. We have proved [16] that any almost $O(\log n)$-wise independent set is an example of the rich set (see Section 6). The following theorem shows that the richness condition is, in certain sense, sufficient for a set to be a hitting set for $\mathcal{P}$. In particular, for an input $\mathbf{a} \in \{0,1\}^n$ and an integer constant $c \geq 0$, denote by $\Omega_c(\mathbf{a}) = \{\mathbf{a}' \in \{0,1\}^n \mid h(\mathbf{a},\mathbf{a}') \leq c\}$ the set of so-called *h-neighbors* of $\mathbf{a}$, where $h(\mathbf{a},\mathbf{a}')$ is the Hamming distance between $\mathbf{a}$ and $\mathbf{a}'$. We also define $\Omega_c(A) = \bigcup_{\mathbf{a} \in A} \Omega_c(\mathbf{a})$ for a given set $A \subseteq \{0,1\}^*$.

Theorem 2. *Let $\varepsilon > \frac{11}{12}$. If A is ε'^{11}-rich for some $\varepsilon' < \varepsilon$ then $H = \Omega_3(A)$ is an ε-hitting set for the class of read-once branching programs of width 3.*

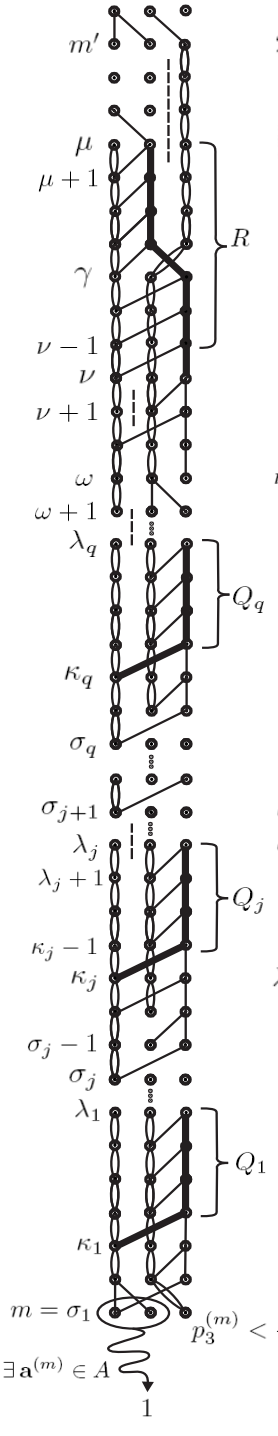
Proof. (sketch) Suppose a normalized read-once branching program P of width 3 with sufficiently many input variables n meets $|P^{-1}(1)|/2^n \geq \varepsilon > \frac{11}{12}$. We will prove that there exists $\mathbf{a} \in H$ such that $P(\mathbf{a}) = 1$. On the contrary, we assume that $P(\mathbf{a}) = 0$ for every $\mathbf{a} \in H$. The main idea of the proof lies in using this assumption first for constraining the structure of branching program P so that the richness of A can eventually be employed to disprove this assumption.

The Plan of the Proof. We start the underlying analysis of the structure of P from its last level d containing the sinks and we go backwards block after block to lower levels. In particular, we inspect the structure of a block whose *last* **level m** satisfies the following four so-called *m-conditions* which can, without loss of generality [15], be met for $m = d$ at the beginning:

1. $t_{11}^{(m)} = t_{21}^{(m)} = \frac{1}{2}$, 2. $t_{32}^{(m)} > 0$, 3. $p_3^{(m)} < \frac{1}{12}$,
4. there is $\mathbf{a}^{(m)} \in A$ such that if we put $\mathbf{a}^{(m)}$ at node $v_1^{(m)}$ or $v_2^{(m)}$, then its onward computational path arrives to the sink labeled with 1.

The block starts at level m' which is defined in Section 4. A typical block from m' through m is schematically depicted in Figure 1. Using the knowledge of this block structure, we define the partition class R associated with this block which includes all the indices of the variables that are queried on the computational path which is indicated in boldface on the top in Figure 1 (Section 3). The edge labels on this path define relevant bits of $\mathbf{c} \in \{0,1\}^n$ so that any input passing through this path that differs from $\mathbf{c}$ on the bit locations from R reaches a double-edge path in the first column, which implements one CNF clause from (4). Similarly, sets $Q_1, \ldots, Q_q$ (candidates for Q) associated with this block are defined so that any input that agrees with $\mathbf{c}$ on the bit locations from Q_j reaches the first column, which implements DNF monomials from (4).

The partition classes associated with the blocks that have been analyzed so far are employed in the richness condition (6) first for $Q = \emptyset$ provided that the partition satisfies (5). The richness is used to prove that the m'-conditions are also met for the first level m' of the block (Section 4). In particular, the richness condition (6) for the partition class R associated with the underlying block ensures that an input $\mathbf{a}^{(m)} \in A$ that is put at node $v_1^{(m')}$ or $v_2^{(m')}$ arrives to the first column (see Figure 1) which implies m'-condition 4 by induction (the recursive step in Section 5). Then the block analysis including the definition of associated partition class and sets Q_j is applied recursively for m replaced with m' etc. If, on the other hand, the underlying partition does not satisfy (5), then one can prove that there is a set Q among Q_j associated with the blocks that have been analyzed so far such that $|Q| \leq \log n$, and the recursive analysis ends (the last paragraph of Section 5). In this case, the richness condition (6) for this Q implies that there is $\mathbf{a} \in H$ whose computational path traverses $v_1^{(m)}$ or $v_2^{(m)}$



$2 \le \mathbf{m'} \uparrow \le \mu:\ t_{32}^{(m')} > 0$
$p_3^{(m')} = p_3^{(\mu)} < \frac{1}{12}$

$2 \le \boldsymbol{\mu} \downarrow < m:\ t_{11}^{(\ell)} = 1$ for $\ell = \mu+1, \dots, m-1$
$p_1^{(\mu)} + p_2^{(\mu)} = p_1^{(m')} + p_2^{(m')}, \quad p_3^{(\mu)} < \frac{1}{12}$

$\mu \le \boldsymbol{\gamma} \uparrow \le \nu:\ t_{12}^{(\gamma)} > 0$

$\mu \le \boldsymbol{\nu} \uparrow \le m:\ t_{12}^{(\ell)} + t_{13}^{(\ell)} > 0$ for $\ell = \mu+1, \dots, \nu$
$p_1^{(\nu)} = p_1^{(\mu)} + p_2^{(\mu)}\left(1 - \frac{1}{2^{|R|}}\right), \quad p_3^{(\nu+1)} = \frac{p_2^{(\mu)}}{2^{|R|+1}}$

$\nu - 1 \le \boldsymbol{\omega} \uparrow \le m:\ \exists$ double-edge path $v_\mu, v_{\mu+1}, \dots, v_\omega$
where $v_\ell \in \{v_2^{(\ell)}, v_3^{(\ell)}\}$
$p_1^{(\omega+1)} \le p_1^{(\nu)} + p_3^{(\nu+1)} = p_1^{(\mu)} + p_2^{(\mu)}\left(1 - \frac{1}{2^{|R|+1}}\right)$
$\longrightarrow \boxed{p_1^{(\omega+1)} \le \left(p_1^{(m')} + p_2^{(m')}\right)\left(1 - \frac{1}{2^{|R|+3}}\right)}$
$p_2^{(\lambda_q)} + p_3^{(\lambda_q)} = p_2^{(\omega+1)} + p_3^{(\omega+1)}$

$\omega + 1 < \boldsymbol{\sigma_{j+1}} \uparrow \le \lambda_i:\ t_{13}^{(\sigma_{j+1})} > 0$
$\omega \le \boldsymbol{\lambda_j} \downarrow < \sigma_j - 1:\ t_{11}^{(\ell)} = t_{22}^{(\ell)} = 1\ \&\ t_{33}^{(\ell)} = \frac{1}{2}$
for $\ell = \lambda_j + 1, \dots, \sigma_j - 1$
$p_2^{(\lambda_j)} + p_3^{(\lambda_j)} = p_2^{(\sigma_{j+1})} + p_3^{(\sigma_{j+1})}$

$\lambda_j + 1 < \boldsymbol{\kappa_j} \downarrow \le \sigma_j:\ t_{13}^{(\kappa_j)} > 0$
$p_3^{(\kappa_j - 1)} = \frac{p_3^{(\lambda_j)}}{2^{|Q_j|-1}} \le \frac{p_2^{(\lambda_j)} + p_3^{(\lambda_j)}}{2^{|Q_j|}}$
$p_2^{(\sigma_j)} + p_3^{(\sigma_j)} \ge \left(p_2^{(\sigma_{j+1})} + p_3^{(\sigma_{j+1})}\right)\left(1 - \frac{1}{2^{|Q_j|}}\right)$

$\boxed{p_3^{(m)} \ge \left(p_2^{(\omega+1)} + p_3^{(\omega+1)}\right)\prod_{j=1}^{q}\left(1 - \frac{1}{2^{|Q_j|}}\right)}$

Fig. 1. The structure of a typical block

of the block defining Q (cf. Figure 1) and m-condition 4 then guarantees this path eventually arrives to the sink labeled with 1 providing $P(\mathbf{a}) = 1$ for $\mathbf{a} \in H$.

The inspection of the block structure has the form of a rather tedious case analysis including various parameters denoting specific levels in the block whose definitions are indicated in boldface. Figure 1 summarizes these definitions having the form of "$a \leq \mathbf{b} \uparrow \leq c : \ C(\mathbf{b})$" which means b is the *greatest* level such that $a \leq b \leq c$ and condition $C(b)$ is satisfied (similarly, $\downarrow$ denotes the *least* such level). Due to the lack of space, the proofs of lemmas are omitted and we will present the analysis only for a 'general' case excluding plenty of degenerated cases which occur when some of the level parameters coincide. In order to focus on this general case illustrating the main idea of the proof we make simplifying assumptions concerning the relations among these levels which will always be introduced in the *bold square brackets* below. The full presentation for all combinations of parameters is available in a preliminary technical report [15].

A Technical Lemma. The following lemma represents a technical tool which will be used for the analysis of the block from **level $\boldsymbol{\mu}$** through m where $2 \leq \mu < m$ denotes the least level of P such that $t_{11}^{(\ell)} = 1$ for every $\ell = \mu + 1, \ldots, m - 1$. For this purpose, define a *switching* path starting from $v \in \{v_2^{(k)}, v_3^{(k)}\}$ at level $\mu \leq k < m$ to be a computational path of length at most 3 edges leading from v to $v_1^{(\ell)}$ for some $k < \ell \leq \min(k + 3, m)$ or to $v_2^{(m)}$ for $m \leq k + 3$.

Lemma 1

(i) $3 < \mu < m - 1$.

(ii) *There are no two simultaneous switching paths starting from* $v_2^{(k)}$ *and from* $v_3^{(k)}$*, respectively, at any level* $\mu \leq k < m$.

(iii) *If* $t_{12}^{(k+1)} > 0$ *for some* $\mu \leq k < m$*, then* $t_{11}^{(\ell)} = t_{33}^{(\ell)} = 1$, $t_{12}^{(\ell)} = t_{22}^{(\ell)} = \frac{1}{2}$ *for every* $\ell = \mu + 1, \ldots, k$*, and* $t_{12}^{(k+1)} = \frac{1}{2}$.

(iv) *If* $t_{13}^{(k+1)} > 0$ *for some* $\mu < k < m$*, then one of the four cases occurs:*

1. $t_{11}^{(k)} = t_{23}^{(k)} = 1$ *and* $t_{12}^{(k)} = t_{32}^{(k)} = \frac{1}{2}$,
2. $t_{11}^{(k)} = t_{23}^{(k)} = 1$ *and* $t_{22}^{(k)} = t_{32}^{(k)} = \frac{1}{2}$,
3. $t_{11}^{(k)} = t_{22}^{(k)} = 1$ *and* $t_{13}^{(k)} = t_{33}^{(k)} = \frac{1}{2}$,
4. $t_{11}^{(k)} = t_{22}^{(k)} = 1$ *and* $t_{23}^{(k)} = t_{33}^{(k)} = \frac{1}{2}$.

In addition, if $t_{23}^{(k)} = 1$ *(case 1 or 2), then* $t_{11}^{(\ell)} = t_{33}^{(\ell)} = 1$ *and* $t_{12}^{(\ell)} = t_{22}^{(\ell)} = \frac{1}{2}$ *for every* $\ell = \mu + 1, \ldots, k - 1$.

3 Definition of Partition Class

The Block Structure from μ to ν (Definition of R). In the following corollary, we summarize the block structure from level μ through **level $\boldsymbol{\nu}$** by using Lemma 1, where $\mu \leq \nu \leq m$ is the greatest level such that $t_{12}^{(\ell)} + t_{13}^{(\ell)} > 0$ for every $\ell = \mu + 1, \ldots, \nu$, and **level $\boldsymbol{\gamma}$** is the greatest level such that $\mu \leq \gamma \leq \nu$ and $t_{12}^{(\gamma)} > 0$ (for $\gamma > \mu$). For simplicity we will further assume $[\boldsymbol{\mu < \gamma < \nu < \mathbf{m}}]$.

Corollary 1

1. $t_{11}^{(\ell)} = t_{33}^{(\ell)} = 1$ *and* $t_{12}^{(\ell)} = t_{22}^{(\ell)} = \frac{1}{2}$ *for* $\ell = \mu+1, \ldots, \gamma-1$ *(Lemma 1.iii),*
2. $t_{11}^{(\gamma)} = t_{23}^{(\gamma)} = 1$ *and* $t_{12}^{(\gamma)} = t_{32}^{(\gamma)} = \frac{1}{2}$ *(case 1 of Lemma 1.iv),*
3. $t_{11}^{(\ell)} = t_{22}^{(\ell)} = 1$ *and* $t_{13}^{(\ell)} = t_{33}^{(\ell)} = \frac{1}{2}$ *for* $\ell = \gamma+1, \ldots, \nu-1$ *(case 3 of Lemma 1.iv),*
4. $t_{13}^{(\nu)} = \frac{1}{2}$ *(similarly to Lemma 1.iii),*
5. $t_{12}^{(\ell)} = 0$ *for* $\ell = \nu, \ldots, m$ *(Lemma 1.iii).*

Now we can define the partition class R associated with the underlying block to be a set of indices of the variables that are tested on the single-edge computational path $v_2^{(\mu)}, v_2^{(\mu+1)}, \ldots, v_2^{(\gamma-1)}, v_3^{(\gamma)}, v_3^{(\gamma+1)}, \ldots, v_3^{(\nu-1)}$, which is illustrated in Figure 1. For the future use of condition (6) we also define relevant bits of string $\mathbf{c} \in \{0,1\}^n$. Thus, let c_i^R be the corresponding labels of the edges creating this computational path (indicated by a bold line in Figure 1) including the edge outgoing from the last node $v_3^{(\nu-1)}$ that leads to $v_2^{(\nu)}$ or to $v_3^{(\nu)}$.

The Block Structure from ω ***to*** m ***(Definition of*** $Q_1, \ldots, Q_q$***).*** We define **level** $\boldsymbol{\omega}$ to be the greatest level such that $\mu < \nu - 1 \leq \omega \leq m$ and the double-edge path from Corollary 1 leading $v_3^{(\mu)}$ to $v_2^{(\nu-1)}$ (see Figure 1) further continues up to level ω containing only nodes $v_\ell \in \{v_2^{(\ell)}, v_3^{(\ell)}\}$ for every $\ell = \mu, \ldots, \omega$. For simplicity we will further assume $[\boldsymbol{\omega < \mathbf{m}}]$. We know $t_{12}^{(m)} = 0$ from Corollary 1.5. We assume $t_{13}^{(m)} > 0$ without loss of generality [15], which implies $t_{32}^{(m)} = 1$ according to Lemma 1.iii. Then Lemma 1.iv can be employed for $k = m-1$ where only case 3 and 4 may occur due to $\omega < m$. In case 3, $t_{13}^{(m-1)} > 0$ and Lemma 1.iv can again be applied recursively to $k = m-2$ etc.

In general, starting with level $\boldsymbol{\sigma_1 = \mathbf{m}}$ that meets $t_{13}^{(\sigma_j)} > 0$ for $j = 1$, we proceed to lower levels and inspect recursively the structure of subblocks indexed as j from **level** $\boldsymbol{\lambda_j}$ through σ_j where λ_j is the least level such that $\mu < \omega \leq \lambda_j < \sigma_j - 1$ and the transitions from case 3 or 4 of Lemma 1.iv, i.e. $t_{11}^{(\ell)} = t_{22}^{(\ell)} = 1$ and $t_{33}^{(\ell)} = \frac{1}{2}$, occur for all levels $\ell = \lambda_j + 1, \ldots, \sigma_j - 1$, as depicted in Figure 1. We will observe that case 4 from Lemma 1.iv occurs at level $\lambda_j + 1$, that is $t_{23}^{(\lambda_j+1)} = \frac{1}{2}$. On the contrary, suppose that $t_{13}^{(\lambda_j+1)} = \frac{1}{2}$ (case 3). For $\lambda_j > \omega$, this means case 1 or 2 occurs at level $\lambda_j < \mu$ by the definition of λ_j, which would be in contradiction to $\omega \leq \lambda_j$ according to Lemma 1.iv. For $\lambda_j = \omega$, on the other hand, $t_{13}^{(\omega+1)} = \frac{1}{2}$ contradicts the definition of ω by Lemma 1.iv. This completes the argument for $t_{23}^{(\lambda_j+1)} = \frac{1}{2}$.

Furthermore, let **level** $\boldsymbol{\kappa_j}$ be the least level such that $\lambda_j + 1 < \kappa_j \leq \sigma_j$ and $t_{13}^{(\kappa_j)} > 0$, which exists since at least $t_{13}^{(\sigma_j)} > 0$. Now we can define the corresponding Q_j (a candidate for Q in the richness condition (6)) to be a set of indices of the variables that are tested on the computational path $v_3^{(\lambda_j)}, v_3^{(\lambda_j+1)}, \ldots, v_3^{(\kappa_j-1)}$, and let $c_i^{Q_j}$ be the corresponding labels of the edges creating this path including the edge from $v_3^{(\kappa_j-1)}$ to $v_1^{(\kappa_j)}$ (indicated by a bold line in Figure 1). This extends

the definition of $\mathbf{c} \in \{0,1\}^n$ associated with R and Q_k for $1 \leq k < j$, which are usually pairwise disjoint due to P is read-once. Nevertheless, the definition of $\mathbf{c}$ may not be unique for indices from their nonempty intersections in some very special cases (including those corresponding to neighbor blocks) but the richness condition will only be used for provably disjoint sets (Section 5). Finally, define next **level $\boldsymbol{\sigma_{j+1}}$** to be the greatest level such that $\omega + 1 < \sigma_{j+1} \leq \lambda_j$ and $t_{13}^{(\sigma_{j+1})} > 0$, if such σ_{j+1} exists, and continue in the recursive definition of $\lambda_{j+1}, \kappa_{j+1}, Q_{j+1}$ with j replaced by $j+1$ etc. If such σ_{j+1} does not exist, then set $q = j$ and the definition of sets $Q_1, \ldots, Q_q$ associated with the underlying block is complete.

The following lemma gives an upper bound on $p_1^{(m)} + p_2^{(m)}$ in terms of $p_1^{(\omega+1)}$.

Lemma 2

$$p_1^{(m)} + p_2^{(m)} \leq 1 - \left(1 - p_1^{(\omega+1)}\right) \prod_{j=1}^{q} \left(1 - \frac{1}{2^{|Q_j|}}\right) . \tag{7}$$

4 The Block Structure below μ Provided That $p_3^{(\mu)} < \frac{1}{12}$

The Block Structure from m' to μ (m'-Conditions 1–3). Throughout this Section 4, we will assume

$$p_3^{(\mu)} < \frac{1}{12} . \tag{8}$$

Based on this assumption, we will further analyze the block structure below level μ in the following lemmas (see Figure 1) in order to satisfy the m'-conditions 1–4 also for the first block level m' so that the underlying analysis can be applied recursively when inequality (8) holds (Section 5). In particular, define the first **level $\boldsymbol{m'}$** of the underlying block to be the greatest level such that $2 \leq m' \leq \mu$ and $t_{32}^{(m')} > 0$ (m'-condition 2), which exists since at least $t_{32}^{(2)} > 0$.

Lemma 3. *$t_{31}^{(k)} = t_{32}^{(k)} = 0$ and $t_{33}^{(k)} = 1$ for $k = m'+1, \ldots, \mu$.*

Lemma 3 together with assumption (8) verifies m'-condition 3 for the first block level m', that is $p_3^{(m')} = p_3^{(\mu)} < 1/12$, which gives $m' \geq 4$ since $p_3^{(3)} \geq 1/2^3$.

Lemma 4. *$t_{11}^{(m')} = t_{21}^{(m')} = \frac{1}{2}$ (m'-condition 1).*

In the following lemma, we will extend an upper bound on $p_1^{(m)} + p_2^{(m)}$ achieved in Lemma 2 (in terms of $p_1^{(\omega+1)}$) in order to derive a recursive formula for an upper bound on $p_1^{(m)} + p_2^{(m)}$ in terms of $p_1^{(m')} + p_2^{(m')}$ which will be used in Section 5 for verifying condition (5).

Lemma 5

$$p_1^{(m)} + p_2^{(m)} \leq 1 - \left(1 - \left(p_1^{(m')} + p_2^{(m')}\right)\left(1 - \frac{1}{2^{|R|+3}}\right)\right) \prod_{j=1}^{q} \left(1 - \frac{1}{2^{|Q_j|}}\right) . \tag{9}$$

5 The Recursion

In the previous Sections 2–4, we have analyzed the structure of the block of P from level m' through m (see Figure 1). We will now employ this block analysis recursively so that $m = m_r$ is replaced by $m' = m_{r+1}$. For this purpose, we introduce additional index $b = 1, \ldots, r$ to the underlying objects in order to differentiate among respective blocks. For example, the sets $R, Q_1, \ldots, Q_q$, defined in Section 3, corresponding to the bth block are denoted as $R_b, Q_{b1}, \ldots, Q_{bq_b}$, respectively. Since we, for simplicity, assume $\nu_b > m_{b-1}$ for $b = 1, \ldots, r$, sets $R_1, \ldots, R_r$ create a partition due to P is read-once.

Inductive Assumptions. We will proceed by induction on r, starting with $r = 0$ and $m_0 = d$. In the induction step for $r + 1$, we assume that the four m_r-conditions are met for the last block level m_r, and let assumption (8) be satisfied for the previous blocks, that is,

$$p_3^{(\mu_b)} < 1/12 \tag{10}$$

for every $b = 1, \ldots, r$. In addition, assume

$$1 - \Pi_r < \delta = \min(\varepsilon - \varepsilon', (12\varepsilon - 11)/13) < 1/13 \tag{11}$$

where $\varepsilon > 11/12$ and $\varepsilon' < \varepsilon$ are the parameters of Theorem 2, and denote $\Pi_k = \prod_{b=1}^{k} \pi_b$ with $\pi_b = \prod_{j=1}^{q_b}(1 - 1/2^{|Q_{bj}|})$, $\varrho_k = \prod_{b=1}^{k} \alpha_b$ with $\alpha_b = (1 - 1/2^{|R_b|+3})$, for $k = 1, \ldots, r$, and $\varrho_0 = \Pi_0 = 1$. Hence, we can employ recursive inequality (9) from Section 4, which is rewritten as $p_{b-1} \leq 1 - (1 - p_b\alpha_b)\pi_b = 1 - \pi_b + p_b\alpha_b\pi_b$ for $b = 1, \ldots, r$ where notation $p_b = p_1^{(m_b)} + p_2^{(m_b)}$ is introduced. Starting with $p_0 = p_1^{(d)} + p_2^{(d)} \geq \varepsilon$, this recurrence can be solved as

$$\begin{aligned} \varepsilon &\leq \textstyle\sum_{k=1}^{r}(1 - \pi_k)\prod_{b=1}^{k-1}\alpha_b\pi_b + p_r\prod_{b=1}^{r}\alpha_b\pi_b \\ &< \textstyle\sum_{k=1}^{r}(1 - \pi_k)\Pi_{k-1} + p_r\varrho_r\Pi_r = 1 - \Pi_r + p_r\varrho_r\Pi_r\,. \end{aligned} \tag{12}$$

In addition, it follows from (12) and (11) that

$$\varrho_r > p_r\varrho_r\Pi_r > \varepsilon - \delta \geq \varepsilon'\,. \tag{13}$$

Recursive Step. Throughout this paragraph, we will consider the case when $1 - \Pi_{r+1} < \delta$ (cf. (11)), while the complementary case concludes the induction and will be resolved in the next paragraph. We know $p_r \leq 1 - (p_2^{(\omega_{r+1}+1)} + p_3^{(\omega_{r+1}+1)})\pi_{r+1}$ according to Lemma 2, and $p_2^{(\omega_{r+1}+1)} + p_3^{(\omega_{r+1}+1)} \geq p_3^{(\mu_{r+1})}$ by the definition of ω_{r+1}, which altogether gives $\varepsilon < 1 - \Pi_r + (1 - p_3^{(\mu_{r+1})}\pi_{r+1})\varrho_r\Pi_r$ according to (12). Hence, $\varepsilon - \delta < (1 - p_3^{(\mu_{r+1})}\pi_{r+1})\varrho_r\Pi_r < 1 - p_3^{(\mu_{r+1})}\pi_{r+1}$ follows from (11), which implies $p_3^{(\mu_{r+1})} < (1 - \varepsilon + \delta)/(1 - \delta) < 1/12$ since $\pi_{r+1} \geq \Pi_{r+1} > 1 - \delta$. Thus, assumption (8) of Section 4 is also met for the $(r + 1)$st block, which justifies recurrence inequality (9) for this block providing the solution

$\varepsilon < 1 - \Pi_{r+1} + p_{r+1}\varrho_{r+1}\Pi_{r+1}$ implying $\varrho_{r+1} > \varepsilon'$ by analogy to (12) and (13). Thus, inductive assumptions (10) and (11) are valid for r replaced with $r+1$.

In Section 4, m_{r+1}-conditions 1–3 have been verified, and thus, it suffices to validate m_{r+1}-condition 4. We exploit the fact that A is ε'^{11}-rich after we show condition (5) for partition $\{R_1, \ldots, R_{r+1}\}$ of $I = \bigcup_{b=1}^{r+1} R_b$. In particular, $\prod_{b=1}^{r+1}(1 - 1/2^{|R_b|}) > \varepsilon'^{11}$ follows from $\varrho_{r+1} > \varepsilon'$ since $(1 - 1/2^{|R_b|+3})^{11} < 1 - 1/2^{|R_b|}$ for $|R_b| \geq 1$. This provides required $\mathbf{a}^{(m_{r+1})} \in A$ such that for every $b = 1, \ldots, r+1$ there exists $i \in R_b$ that meets $a_i^{(m_{r+1})} \neq c_i^{R_b}$ according to (6) where $Q = \emptyset$. Hence, the computational path for this $\mathbf{a}^{(m_{r+1})}$ ends up in sink $v_1^{(d)}$ or $v_2^{(d)}$ labeled with 1 when we put $\mathbf{a}^{(m_{r+1})}$ at node $v_1^{(m_{r+1})}$ or $v_2^{(m_{r+1})}$ (m_{r+1}-condition 4) by the definition of R_b, $c_i^{R_b}$, and the structure of P (see Figure 1). Thus, we can continue recursively for r replaced with $r+1$ etc.

The End of Recursion. In this paragraph, we will consider the complementary case of $1 - \Pi_{r+1} \geq \delta$, which concludes the recursion. Suppose $|Q_{bj}| > \log n$ for every $b = 1, \ldots, r+1$ and $j = 1, \ldots, q_b$, then we would have $\Pi_{r+1} \geq (1 - 1/2^{\log n})^{n/\log n} > 1 - (1/n) \cdot (n/\log n) = 1 - 1/\log n$, which gives a contradiction for sufficiently large n. Hence, there must be $1 \leq b^* \leq r+1$ and $1 \leq j^* \leq q_{b^*}$ such that $|Q_{b^*j^*}| \leq \log n$, and we denote $Q = Q_{b^*j^*}$. Clearly, $Q \cap R_b = \emptyset$ for $b = 1, \ldots, b^* - 2$ due to P is read-once while it may happen that $Q \cap R_{b^*-1} \neq \emptyset$ for $j^* = 1$, $\kappa_{b^*1} = \sigma_{b^*1} = m_{b^*-1}$, and $t_{23}^{(m_{b^*-1})} = 0$. Thus, let r^* be the maximum of $b^* - 2$ and $b^* - 1$ such that $Q \cap R_{r^*} = \emptyset$. We will again employ the fact that A is ε'^{11}-rich. First condition (5) for partition $\{R_1, \ldots, R_{r^*}\}$ of $I = \bigcup_{b=1}^{r^*} R_b$ is verified as $\prod_{b=1}^{r^*}(1 - 1/2^{|R_b|}) > \varrho_r^{11} > \varepsilon'^{11}$ according to (13). This provides $\mathbf{a}^* \in A$ such that $a_i^* = c_i^Q$ for every $i \in Q$ and at the same time, for every $b = 1, \ldots, r^*$ there exists $i \in R_b$ that meets $a_i^* \neq c_i^{R_b}$ according to (6).

Lemma 6. *Denote $\lambda = \lambda_{b^*j^*}$. There are two 'switching' paths starting from $v_2^{(k)}$ and from $v_3^{(k)}$, respectively, at some level $\lambda - 2 \leq k < \lambda$, which may lead to $v_3^{(\lambda)}$ in addition to $v_1^{(\lambda-1)}$ or $v_1^{(\lambda)}$.*

By a similar argument to Lemma 1ii, Lemma 6 gives an h-neighbor $\mathbf{a}' \in \Omega_2(\mathbf{a}^*) \subseteq H$ of $\mathbf{a}^* \in A$ such that $\mathbf{a}' \in M(v_1^{(\lambda)}) \cup M(v_3^{(\lambda)})$. Thus, either $\mathbf{a}' \in M(v_1^{(\lambda)}) \subseteq M(v_1^{(m_{b^*-1})}) \cup M(v_2^{(m_{b^*-1})})$ or $\mathbf{a}' \in M(v_3^{(\lambda)})$ which implies $\mathbf{a}' \in M(v_1^{(\kappa_{b^*j^*})}) \subseteq M(v_1^{(m_{b^*-1})}) \cup M(v_2^{(m_{b^*-1})})$ since $a'_i = a_i^* = c_i^Q$ for every $i \in Q$ according to (6) (see Figure 1). Note that $M(v_1^{(\kappa_{b^*j^*})}) = M(v_1^{(m_{b^*-1})})$ for $r^* = b^* - 2$. Hence, $P(\mathbf{a}') = 1$ because for every $b = 1, \ldots, r^*$ there exists $i \in R_b$ that meets $a'_i = a_i^* \neq c_i^{R_b}$ due to (6). This completes the proof of Theorem 2. □

6 Conclusion

In order to achieve an explicit polynomial time construction of a hitting set for read-once branching programs of width 3 we combine Theorem 2 with our result that almost $O(\log n)$-wise independent sets are rich:

Theorem 3 ([16]). *Let $\varepsilon > 0$, C be the least odd integer greater than $(\frac{2}{\varepsilon} \ln \frac{1}{\varepsilon})^2$, and $0 < \beta < \frac{1}{n^{C+3}}$. Then any $(\lceil (C+2) \log n \rceil, \beta)$-wise independent set is ε-rich.*

In particular, we can use the result due to Alon et al. [1] who, for $\beta > 0$ and $k = O(\log n)$, constructed (k, β)-*wise independent set* $\mathcal{A} \subseteq \{0,1\}^*$ in time polynomial in $\frac{n}{\beta}$ such that for sufficiently large n and any index set $S \subseteq \{1, \ldots, n\}$ of size $|S| \leq k$, the probability that a given $\mathbf{c} \in \{0,1\}^n$ coincides with a string $\mathbf{a} \in \mathcal{A}_n = \mathcal{A} \cap \{0,1\}^n$ on the bit locations from S is almost uniform, that is, $||\{\mathbf{a} \in \mathcal{A}_n \,|\, (\forall i \in S)\, a_i = c_i\}|/|\mathcal{A}_n| - 1/2^{|S|}| \leq \beta$. It follows that $H = \Omega_3(\mathcal{A})$, which can be constructed in polynomial time, is an ε-hitting set for read-once branching programs of width 3 and $\varepsilon > 11/12$.

In the present paper, we have made an important step in the effort of constructing the hitting set generators for the model of read-once branching programs of bounded width. Although this model seems to be relatively weak, the presented proof is far from being trivial. From the point of view of derandomization of unrestricted models, our result still appears to be unsatisfactory but it is the best we know so far. The issue of whether our technique based on the richness condition can be extended to the case of width 4 or to bounded width represents an open problem for further research. Another challenge for improving our result is to optimize parameter ε, e.g. to achieve the result for $\varepsilon \leq \frac{1}{n}$, which would be important for practical derandomizations. In fact, the presented proof can be extended for $\varepsilon > 5/6$ by increasing the number of cases in the analysis.

References

1. Alon, N., Goldreich, O., Håstad, J., Peralta, R.: Simple constructions of almost k-wise independent random variables. Random Struct. Algor. 3(3), 289–304 (1992)
2. Beame, P., Machmouchi, W.: Making branching programs oblivious requires superpolynomial overhead. ECCC TR10-104 (2010)
3. Bogdanov, A., Dvir, Z., Verbin, E., Yehudayoff, A.: Pseudorandomness for width 2 branching programs. ECCC TR09-70 (2009)
4. Braverman, M., Rao, A., Raz, R., Yehudayoff, A.: Pseudorandom generators for regular branching programs. In: Proc. of FOCS 2010, pp. 41–50 (2010)
5. Brody, J., Verbin, E.: The coin problem, and pseudorandomness for branching programs. In: Proc. of FOCS 2010, pp. 30–39 (2010)
6. De, A.: Improved pseudorandomness for regular branching programs. In: Proc. of CCC 2011, pp. 221–231 (2011)
7. De, A., Etesami, O., Trevisan, L., Tulsiani, M.: Improved Pseudorandom Generators for Depth 2 Circuits. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) APPROX and RANDOM 2010. LNCS, vol. 6302, pp. 504–517. Springer, Heidelberg (2010)
8. Fefferman, B., Shaltiel, R., Umans, C., Viola, E.: On beating the hybrid argument. ECCC TR10-186 (2010)
9. Goldreich, O., Wigderson, A.: Improved Derandomization of BPP Using a Hitting Set Generator. In: Hochbaum, D.S., Jansen, K., Rolim, J.D.P., Sinclair, A. (eds.) RANDOM 1999 and APPROX 1999. LNCS, vol. 1671, pp. 131–137. Springer, Heidelberg (1999)

10. Koucký, M., Nimbhorkar, P., Pudlák, P.: Pseudorandom generators for group products. In: Proc. of STOC 2011, pp. 263–272 (2011)
11. Meka, R., Zuckerman, D.: Pseudorandom generators for polynomial threshold functions. In: Proc. of STOC 2010, pp. 427–436 (2010)
12. Nisan, N.: Pseudorandom generators for space-bounded computation. Combinatorica 12(4), 449–461 (1992)
13. Nisan, N., Wigderson, A.: Hardness vs. randomness. J. Comput. Syst. Sci. 49(2), 149–167 (1994)
14. Šíma, J., Žák, S.: A Polynomial Time Constructible Hitting Set for Restricted 1-Branching Programs of Width 3. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 522–531. Springer, Heidelberg (2007)
15. Šíma, J., Žák, S.: A polynomial time construction of a hitting set for read-once branching programs of width 3. ECCC TR10-088 (2010)
16. Šíma, J., Žák, S.: Almost k-Wise Independent Sets Establish Hitting Sets for Width-3 1-Branching Programs. In: Kulikov, A., Vereshchagin, N. (eds.) CSR 2011. LNCS, vol. 6651, pp. 120–133. Springer, Heidelberg (2011)
17. Wegener, I.: Branching Programs and Binary Decision Diagrams—Theory and Applications. SIAM Monographs on Discrete Mathematics and Its Applications (2000)

Complete Problem for Perfect Zero-Knowledge Quantum Proof

Jun Yan*

School of Computer Science and Technology
University of Science and Technology of China
Hefei, Anhui 230027, China
and
State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
Beijing 100190, China
junyan@ios.ac.cn

Abstract. The main purpose of this paper is to prove that (promise) problem Quantum State Identicalness (abbreviated QSI) is *essentially* complete for perfect zero-knowledge quantum interactive proof (**QPZK**). Loosely speaking, problem QSI is to decide whether two efficiently preparable quantum states (captured by quantum circuit of polynomial size) are identical or far apart (in trace distance). It is worthy noting that our result does not have classical counterpart yet; natural complete problem for perfect zero-knowledge interactive proof (**PZK**) is still unknown. Our proof generalizes Watrous' completeness proof for statistical zero-knowledge quantum interactive proof (**QSZK**), with an extra idea inspired by Malka to deal with completeness error. With complete problem at our disposal, we can immediately prove (and reprove) several interesting facts about **QPZK**.

Keywords: Quantum zero-knowledge proof, perfect zero-knowledge, complete problem, quantum complexity, quantum cryptography.

1 Introduction

Zero-knowledge proof has been a hot topic and played an important role in complexity and cryptography research since it was introduced by Goldwasser, Micali, and Rackoff in [11]. Zero-knowledge proof is an intriguing notion, from which verifier "learns" nothing but the truth of the assertion. Recall that in canonical proof system represented by complexity class **NP**, prover just sends witness as the proof for the verifier to check. Intuitively, a canonical proof system cannot be zero-knowledge, for it also reveals the witness to the verifier other than the truth of the assertion. To construct zero-knowledge proof system, we have to generalize the notion of canonical proof. Such generalization turns out to be of

* This work is supported by the National Natural Science Foundation of China (Grant No. 60833001).

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 419–430, 2012.

two folds: First, we allow the proof to be *probabilistic*, with a slight of *completeness and soundness errors*. Second, we allow the proof to be *interactive*, in the sense that prover and verifier can exchange messages. The resulting proof system is known as *interactive proof system*. An alternative way which preserves non-interactiveness of the canonical proof is to let prover and verifier share a random string a prior, resulting in a proof system known as *non-interactive proof system*. There is a *protocol* associated with each interactive and non-interactive proof system, describing (honest) prover's and (honest) verifier's strategies. The formulation of the zero-knowledge property of a proof system follows *simulation paradigm*; loosely speaking, we says verifier "learns nothing" from a proof if the proof itself (a probability distribution) can be approximately generated without prover. According to the "quality" of approximation, we have *perfect*, *statistical*, and *computational* zero-knowledge proofs (denoted by **PZK**, **SZK**, and **ZK** in the interactive model, and **NIPZK**, **NISZK**, and **NIZK** in the non-interactive model, respectively). The formal definition and more details about zero-knowledge proof can be found in standard textbooks such as [8, Chapter 9], [7, Chapter 4].

Quantum proof system is a generalization of classical proof system in the quantum world. In the past decade, a variety of computational models of quantum proof system (see [23,1,12,17,15,4]) were proposed and studied. Since quantumness is a phenomenan, for good or bad, that exists in nature, we cannot help considering the possibility of zero-knowledge quantum proof[1], which may play an important role in quantum cryptography (like its classical counterpart in classical cryptography). As a natural generalization of classical zero-knowledge proof, we can define **QPZK**, **QSZK**, and **QZK** in the interactive model, and **NIQPZK**, **NIQSZK**, and **NIQZK** in the non-interactive model, respectively (see [24,27,15,4]).

Two Generic Approaches

To study properties of zero-knowledge proof, there are two generic approaches. The first one is via (black-box) transformation. That is, given a zero-knowledge proof system, we construct a new one for which prover's and verifier's strategies are constructed using original prover's strategy, original verifier's strategy, plus original simulator, as black-boxes. For example, one can transform an honest-verifier statistical zero-knowledge interactive proof system with completeness error into another one with perfect completeness [6].

In this paper, we are more interested in the second approach to study zero-knowledge proof, namely, via complete problem. This approach is in the same spirit as we study complexity class **NP** via various **NP**-complete problems. Sahai and Vadhan [19] initialized this approach. In particular, they found problems Statistical Difference (SD) and its complement $\overline{\mathsf{SD}}$ are complete for statistical zero-knowledge interactive proof (**SZK**). Follow-up works include [10], [21], [9],

[1] Some researchers may use term "quantum zero-knowledge proof", but we choose to follow Watrous [26].

among others. Using complete problems, many interesting facts about statistical zero-knowledge (interactive and non-interactive) proof are proved *unconditionally* (in contrast to those proved based on complexity assumption such as existence of one-way function). Refer to [22] for a survey on the study of statistical zero-knowledge proof via complete problem.

In quantum case, we can also study zero-knowledge quantum proof via both transformation and complete problem. Interested readers are referred to [12], [17], [14], et al., for the first approach. With respect to complete problem, Watrous [24] was the first to extend the idea of [19] to study statistical zero-knowledge quantum interactive proof (**QSZK**). Specifically, in [24] two promise problems, Quantum State Distinguishability (QSD) and its complement Quantum State Closeness (QSC), were shown to be **QSZK**-complete, where problem QSD can be viewed as the quantum analog of **SZK**-complete problem SD. Later, in the same spirit, Kobayashi [15] (implicitly) found a problem named Quantum State Closeness to Identity (QSCI) that is complete for statistical zero-knowledge quantum non-interactive proof (**NIQSZK**). More recently, using quantum extractor, complete problems for **QSZK** and **NIQSZK** about (von Neumann) entropy difference were found; see [3,4]. Thus far, almost all complete problems for statistical zero-knowledge (classical) proof find their quantum counterparts.

Motivation and Related Work

Note that in either classical or quantum cases, only complete problems for statistical zero-knowledge proof are found. Naturally we shall ask, what about complete problems for perfect and computational zero-knowledge proof, in classical and quantum cases, respectively? In this paper, we shall focus on perfect zero-knowledge proof.

Let us first review some prior related works. In his thesis [22, section 4.7], Vadhan fully discussed the extension of **SZK** completeness proof to **PZK**. In particular, he found that the straightforward adaption of his proof only gives hard problems for a restriction of **PZK**. Since then, there have been no progress towards complete problems for perfect zero-knowledge proof until recently, when Malka [16] constructed a (comparably natural) complete problem for perfect zero-knowledge non-interactive proof (**NIPZK**) and a hard problem for public-coin **PZK**. The genesis of Malka's construction is a way to deal with completeness error.

In quantum case, up until now, nearly nothing is known about the complete problem for **QPZK**. Instead, Kobayashi [14] proved several impressing properties about **QPZK** via transformations, while remarking that the finding of natural complete problem for **QPZK** are definitely helpful. As for non-interactive model, Kobayashi [15] constructed a complete problem for $\mathbf{NIQPZK}_1$ (**NIQPZK** with perfect completeness).

In this paper, we try to answer the following question: can we apply Malka's [16] idea in classical case to construct complete problem for perfect zero-knowledge quantum proof?

Our Contribution

The main result of this paper is to give a (comparably) natural (promise) problem that is complete for **QPZK**. To our knowledge, this is the first time that some natural (not involving computation of universal model of computation) complete problem is found for general perfect zero-knowledge interactive proof (in both classical and quantum case). We can also carry the same study in non-interactive model, obtaining a **NIQPZK**-complete problem.

To get a taste of our **QPZK**-complete problem, it would be beneficial to first recall the **QSZK**-complete problem QSC. Loosely speaking, instances of problem QSC consist of a pair of efficiently preparable quantum states (captured by quantum circuit of polynomial size; see section 3 for detail), where for yes instance these two states are close (in trace distance), while for no instance they are far apart. Our **QPZK**-complete problem is *essentially* (not exactly) a special case of problem QSC as follows: the no instance is the same as problem QSC, whereas the yes instance now is restricted to a pair of efficiently preparable quantum states that are *identical*; we call this special problem Quantum State Identicalness (QSI). Roughly, our actual **QPZK**-complete problem adds a **BQP** instance to each instance of problem QSI; the formal definition is referred to Definition 2.

With complete problems at our disposal, we can immediately prove (and reprove) several interesting facts about perfect zero-knowledge quantum (interactive and non-interactive) proof as follows.

1. Every problem possessing perfect zero-knowledge quantum interactive proof has a two-message honest-verifier perfect zero-knowledge quantum interactive proof, with exponentially small completeness and soundness error; it also has a three-message public-coin honest-verifier perfect zero-knowledge quantum interactive proof, in which verifier's message consists of a single coin flip.
2. **HVQPZK** = **QPZK**. That is, from complexity view, the restriction to honest verifier *does not* change the class of problems possessing perfect zero-knowledge quantum interactive proof.
3. $\mathbf{QPZK} \subseteq \mathbf{BQP}^{\mathbf{QPZK}_1}$, $\mathbf{NIQPZK} \subseteq \mathbf{BQP}^{\mathbf{NIQPZK}_1}$, where the subscript "1" stands for with perfect completeness. This implies that allowing completeness error essentially *does not* increase the complexity of perfect zero-knowledge quantum proof.
4. $\mathbf{NIQPZK}_{\mathrm{h}} = \mathbf{QPZK} = \mathbf{QPZK}_{\mathrm{h}}$, where subscript "h" indicates the help model [4] (a model lying between standard interactive and non-interactive models).
5. $\mathbf{QPZK}_1$ is closed under monotone boolean formula. This result can be viewed as quantum analog of results in [20] and [5], where boolean closure property for some special cases of **PZK** is established.

We remark that among the facts listed above, only the second part of item 1 and item 2 are previously known, which were proved by Kobayashi [14] through a series of transformations. In comparison, our proof via complete problem is almost straightforward.

Main Idea

The main idea of our construction of **QPZK**-complete problem is from Watrous [24] and Malka [16]: we almost follow [24] to do simulator analysis, with only one difference that is similar to [16] to deal with completeness error. Roughly speaking, the difference is that now we no longer move the completeness error into the simulation. This difference will result in the instance of our complete problem having an extra quantum circuit (compared with **QSZK**-complete problem QSC) to encode the acceptance probability of simulator. More detail is referred to section 4. We remark that due to the quantum nature, our construction of **QPZK**-complete problem is different from [16]; indeed, it is simpler and more straightforward.

Our **NIQPZK**-complete problem is obtained by the same idea, except that now the simulator analysis follows Kobayashi [15].

Comparing with Results in Classical Case

Problem QSI can be viewed as the quantum analog of problem $\overline{\mathsf{SD}}^{1/2,0}$ introduced in [22, section 4.7], whose instances consist of a pair of efficiently samplable probability distributions, where for yes instance these two distributions are close (in statistical difference), while for no instance they are far apart. As a special case of **SZK**-complete problem $\overline{\mathsf{SD}}$, it is tempting to prove that problem $\overline{\mathsf{SD}}^{1/2,0}$ is **PZK**-complete. But whether this is true is still open: we only know that this problem is hard for public-coin **PZK** with respect to honest verifier and with perfect completeness. Malka [16] modified problem $\overline{\mathsf{SD}}^{1/2,0}$ to get a hard problem for public-coin **PZK** with respect to honest verifier, removing perfect completeness restriction. In comparison, our quantum result is much stronger: it does not suffer any restrictions, giving a complete problem for general **QPZK**.

Organization

In this extended abstract, we shall highlight the specification of our **QPZK**-complete problem and the idea of its construction. The technical detail of the proof, as well as the completeness theorem in non-interactive model, and applications of complete problems, are all referred to the full version of this paper [28].

The remainder of this paper is organized as follows. In section 2 we review some background materials. Section 3 is devoted to the formal definition of our complete problems. Section 4 contains the sketch of the proof of completeness theorem for **QPZK**. We conclude with section 5.

2 Preliminaries

We assume readers are familiar with basic quantum computation and information (see [18,13]), as well as basic notion of zero-knowledge (classical) interactive proof system (see [2,7,8]).

2.1 Quantum Circuit Model

In this paper, we shall restrict our attention to *unitary* quantum circuit model, where the choice of universal gate set could be arbitrary[2]. In particular, one can choose *Shor basis*: Toffoli gate, Hadamard gate, and Phase-shift gate. Measurement of a qubit is with respect to computational basis $\{|0\rangle, |1\rangle\}$, described by $\{\Pi_0, \Pi_1\}$.

We formalize efficient quantum algorithm Q in terms of *polynomial-time uniformly generated* family of quantum circuits $\{Q_x\}$, where by "polynomial-time uniformly generated" we mean there is a (classical) Turing machine which on input x, outputs a description of quantum circuits Q_x in time polynomial of $|x|$.

2.2 Efficiently Preparable Quantum State

An *efficiently preparable quantum state* is encoded by a quantum circuit Q of polynomial size in the following way: apply Q on quantum registers denoted by (O, G) that are initialized in state $|0\rangle$, where registers O and G correspond to the *output* and *non-output* (garbage) qubits, respectively. That is, quantum state encoded by quantum circuit Q, which we denote by ρ^Q, is $\mathrm{Tr}_{\mathcal{G}}\big(Q|0\rangle\langle 0|Q^*\big)$, where *partial trace* $\mathrm{Tr}_{\mathcal{G}}(\cdot)$ is tracing out qubits corresponding to register G.

Efficiently preparable quantum state can be viewed as quantum analog of efficiently samplable probability distribution [22, Definition 3.1.1].

2.3 Perfect Zero-Knowledge Quantum Interactive Proof

Quantum interactive proof system [12] generalizes classical interactive proof system by allowing prover, verifier, as well as communication channel, to use quantumness. To formally define perfect zero-knowledge property of quantum interactive proof system, we need first to introduce the notion of verifier's view.

Suppose (P,V) is an m-message quantum interactive proof system. Following [24], we define *verifier's view* immediately after the i-th message is sent, denoted by $\mathrm{view}_{\mathrm{P,V}}(x, i)$, as the joint quantum state of all qubits other than those at prover's hand at that moment. For our convenience, we also define $\mathrm{view}_{\mathrm{P,V}}(x, 0)$ and $\mathrm{view}_{\mathrm{P,V}}(x, m+1)$ as the *initial* (before the running) and *final* (after the running) views of verifier, respectively.

Following [24], we say quantum interactive proof system (P,V) has *perfect zero-knowledge* property with respect to *honest verifier* if there exists a collection of efficiently preparable quantum states $\{\sigma_{x,i}\}$ such that for each input $x \in A_{\mathrm{yes}}$, and for each $i \in \{0, 1, \ldots, m+1\}$,

$$\mathrm{view}_{\mathrm{P,V}}(x, i) = \sigma_{x,i}. \tag{1}$$

[2] We remark that our complete theorems are insensible to the choice of universal unitary quantum gate set. However, to prove **HVQPZK** = **QPZK**, we need *reversible computation* and *phase-flip* be implemented without error (this is required in *quantum rewinding lemma* [24] that will be applied).

In other words, there is a *simulator* which on input $x \in A_{\text{yes}}$, runs in polynomial time and outputs $\text{view}_{\text{P,V}}(x, i)$ for each i.

We shall denote by **HVQPZK** the class of promise problems possessing honest-verifier perfect zero-knowledge quantum interactive proof. Though perfect zero-knowledge property only with respect to honest-verifier seems a little bit weak in practice, class **HVQPZK** is nevertheless suitable for complexity study. In this paper, we actually prove completeness theorem for **HVQPZK**; it turns out that with our **HVQPZK**-complete problem, we immediately have **HVQPZK** = **QPZK** by calling quantum rewinding lemma [27]. The equivalence of **HVQPZK** and **QPZK** in turn justifies that our focus on **HVQPZK** does not lose any generality. Thus, here we even choose not to give formal definition of **QPZK**, which requires more setup that is not relevant to the focus of this paper; the formal definition of **QPZK** can be found in [27,14].

As a remark about the definition of perfect zero-knowledge quantum interactive proof, note that the generally accepted definition for perfect zero-knowledge (classical) proof allows simulator to fail with some probability. In spite of this, it turns out that such relaxation does not change the corresponding complexity classes induced by perfect zero-knowledge proof, either in classical or quantum cases (see [16] and [14], respectively). These facts once again illustrate the robustness of complexity classes **PZK** and **QPZK**.

2.4 Perfect Zero-Knowledge Quantum Non-interactive Proof

Recall that in classical case, non-interactive proof consists of only one message which is sent from prover to verifier; moreover, prover and verifier share a prior a uniformly distributed random string known as *common reference string* [7]. In quantum case, Kobayashi [15] suggested replacing the random string with *EPR pairs* such that prover and verifier keep one qubit of each EPR pair privately before the execution of the protocol.

3 Complete Problems

In this section, we shall introduce several promise problems concerning about efficiently preparable quantum state. Before giving formal definition, we need first introduce the notion of trace distance between two quantum states. Specifically, the *trace distance* between two quantum states ρ and ξ, which we denote by $\delta(\rho, \xi)$, is equal to $\|\rho - \xi\|_1 / 2$, where $\|\cdot\|_1$ is the *trace norm*, or 1-norm (see [25]). The trace distance can be viewed as the quantum analog of statistical difference between two probability distributions.

The first problem we are to introduce is problem Quantum State Identicalness (abbreviated QSI).

Definition 1. *The specification of problem* QSI *is as follows.*

Input: description of a pair of quantum circuits (Q_0, Q_1), which encode two quantum states, respectively.

Promise: Circuits Q_0 and Q_1 act on, and output, the same number of qubits. Moreover, either of the following two conditions hold:
(1) $\delta(\rho^{Q_0}, \rho^{Q_1}) = 0$,
(2) $\delta(\rho^{Q_0}, \rho^{Q_1}) \geq 2/3$.
Output: Accept in case (1) and reject in case (2).

We point out that problem QSI can be viewed as a special case problem QSC, in which the yes instance is relaxed to be $\delta(\rho^{Q_0}, \rho^{Q_1}) \leq 1/3$. We are interested in problem QSI because later in this paper we shall show its $\mathbf{QPZK}_1$-completeness (**QPZK** with perfect completeness); moreover, our **QPZK**-complete problem is just a slight variant of problem QSI, as described below.

Definition 2. *The specification of problem* QSI′ *is as follows.*

Input: description of a triple of quantum circuits (Q_0, Q_1, Q_2), which encode three quantum states, respectively.
Promise: Circuits Q_0 and Q_1 act on, and output, the same number of qubits; circuit Q_2 outputs one qubit. Moreover, either of the following two conditions hold:
(1) $\delta(\rho^{Q_0}, \rho^{Q_1}) = 0$ and $Tr(\Pi_1 \rho^{Q_2}) \geq 2/3$;
(2) $\delta(\rho^{Q_0}, \rho^{Q_1}) \geq 1/2$ or $Tr(\Pi_1 \rho^{Q_2}) \leq 1/3$.
Output: Accept in case (1) and reject in case (2).

Compared with problem QSI, the instance of problem QSI′ has an extra quantum circuit Q_2, which induces a **BQP** instance; the motivation of its construction is referred to section 4.

We remark that the choice of constants in the definitions above is arbitrary, due to a straightforward polarization lemma.

Next, we are going to introduce two additional problems concerning about efficiently preparable quantum state. Actually, these two problems can be viewed as special cases of the two problems defined above respectively: if we fix quantum circuit Q_0 to encode *maximally mixed state* (represented by density operator $\mathbb{1}/2^k$, where integer k is the number of qubits designated as output) in the definitions of problem QSI and QSI′, then we obtain problems that we shall denote by QSII (Quantum State Identicalness to Identity) and QSII′, respectively. Kobayashi [15] proved that problem QSII is $\mathbf{NIQPZK}_1$-complete; we can extend this result to show that problem QSII′ is **NIQPZK**-complete.

4 The Completeness Theorem

In this section, we shall sketch the completeness proof for **HVQPZK**, with the focus on the idea of the construction of our complete problem QSI′. The proof itself is adapted from Watrous' completeness proof for **HVQSZK** [24], with a new idea inspired by Malka [16] to deal with completeness error. We shall also give the statement of complete theorem for **NIQPZK** without proof.

Theorem 1. *Problem* QSI' *is* **HVQPZK***-complete.*

Proof. We only sketch the proof here, highlighting the main idea.

A **HVQPZK** protocol for problem QSI' is as follows. On input (Q_0, Q_1, Q_2), we let verifier first run a procedure resembling **BQP** error reduction: apply many copies of Q_2 on qubits in state $|0\rangle$, and then measure the output qubits of all these copies: reject immediately if less than a half of outcomes are one. Then conditioned on verifier does not reject, we let prover and verifier execute either of two *identicalness tests*, which are adapted from closeness tests given in [24], on input (Q_0, Q_1). This will establish that problem QSI' belongs to **HVQPZK**.

Next, we give a reduction from an arbitrary problem $A \in$ **HVQPZK** to problem QSI'.

Suppose (P,V) is an m-message honest-verifier perfect zero-knowledge quantum interactive proof system for problem A. Following [12] and [24], we can formalize the running of (P,V) on input $x \in A_{\text{yes}} \cup A_{\text{no}}$ in terms of quantum circuits. Specifically, the workspace of (P,V) is divided into three parts of quantum registers P, M and V, corresponding to prover's private workspace, communication channel, and verifier's private workspace, respectively. At the beginning, all qubits of the workspace are initialized to be in state $|0\rangle$. Then prover and verifier take in turns to apply their operations (represented by quantum circuits) on quantum register (P, M) and (M, V), respectively. Since in this paper we restrict to unitary quantum circuit model, all these operations are unitary. One qubit, say the first qubit of register V, is designated as the *output* of the whole proof system.

We introduce some notations that are consistent with [24]. Let $n = |x|$. Without loss of generality, assume m is even (thus, verifier sends the first message); let $k = m/2 + 1$. Suppose prover's and verifier's operations are $P_1, \ldots, P_{k-1}$ and $V_1, \ldots, V_k$, respectively. Suppose the simulator for (P,V) outputs a collection of quantum states, $\{\rho_j\}$ and $\{\xi_j\}$, to approximate verifier's views. The case for $m = 4$ is illustrated in Figure 1.

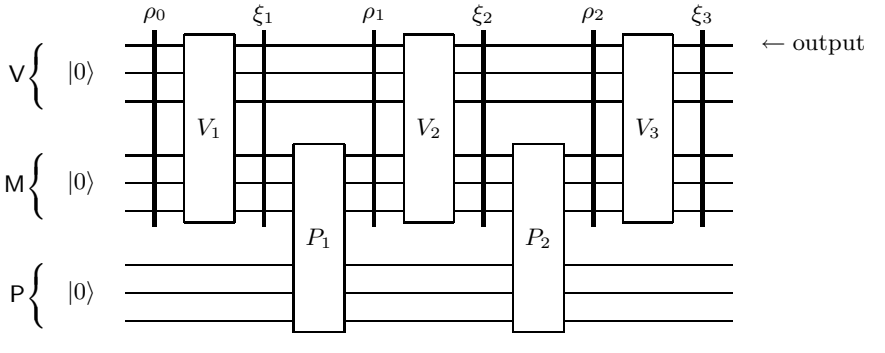


Fig. 1. A 4-message perfect zero-knowledge quantum interactive proof system

Without loss of generality, we can assume that the collection of quantum states $\{\rho_j\}$ and $\{\xi_j\}$ satisfy (whether $x \in A_{\text{yes}}$ or $x \in A_{\text{no}}$) the following properties:

1. $\rho_0 = |0\rangle\langle 0|$;
2. $V_j \rho_{j-1} V_j^* = \xi_j$, for $j = 1, \ldots, k$.

These can be achieved by a simple modification of the simulator as [24].

It turns out that whether $x \in A_{\text{yes}}$ or $x \in A_{\text{no}}$ can be based on the simulator analysis as below:

1. If input $x \in A_{\text{yes}}$, then by completeness and honest-verifier perfect zero-knowledge property of the protocol, we have $\text{Tr}_{\mathcal{M}}(\xi_j) = \text{Tr}_{\mathcal{M}}(\rho_j)$, $j = 1, \ldots, k-1$, and $\text{Tr}\big(\Pi_1 \xi_k\big) \geq 1 - 2^n$.
2. If input $x \in A_{\text{no}}$, then by soundness of the protocol, either for some j, $\delta(\text{Tr}_{\mathcal{M}}(\xi_j), \text{Tr}_{\mathcal{M}}(\rho_j))$ is "noticeable", or $\text{Tr}\big(\Pi_1 \xi_k\big)$ is "negligible". For otherwise, prover can use a simulator-based strategy to cheat verifier to accept with a noticeable amount of probability.

We highlight that compared with the simulator analysis for **HVQSZK** in Watrous' proof, here we have an extra term $\text{Tr}\big(\Pi_1 \xi_k\big)$, which is used to capture the acceptance probability of the final state output by the simulator (probability that the final state will cause verifier to accept). In Watrous' proof, this term is not needed because in case of **HVQSZK**, one can assume, also by a simple modification of simulator, that the resulting simulator always outputs a final state which will cause verifier to accept with certainty. However, this modification moves completeness error into the simulation. Note that this error of simulation is allowable in case of statistical zero-knowledge, which can tolerate exponentially small error. But in case of perfect zero-knowledge, we cannot do this. So in our reduction, we do not do this modification of simulator; instead, we use an extra quantum circuit to capture the acceptance probability of the final state. This will cause the resulting complete problem (QSI') a bit more complex (having an extra quantum circuit to capture $\text{Tr}\big(\Pi_1 \xi_k\big)$) than **HVQSZK**-complete problem QSC. Actually, this is exactly quantum analog of Malka's idea [16] in classical case.

Now we describe the instance of problem QSI' to which input x is reduced:

- Q_0: quantum circuit which encodes quantum state $\text{Tr}_{\mathcal{M}}(\rho_1) \otimes \cdots \otimes \text{Tr}_{\mathcal{M}}(\rho_{k-1})$.
- Q_1: quantum circuit which encodes quantum state $\text{Tr}_{\mathcal{M}}(\xi_1) \otimes \cdots \otimes \text{Tr}_{\mathcal{M}}(\xi_{k-1})$.
- Q_2: quantum circuit which encodes quantum state ξ_k, with the output redesignated as the qubit intended as the approximation of the first qubit of register V.

Clearly, the description of quantum circuits Q_0, Q_1, Q_2 can be computed in polynomial time given the simulator (which runs in polynomial time). □

We observe that for $\mathbf{HVQPZK}_1$, a special case of **HVQPZK** with perfect completeness, quantum circuit Q_2 in our reduction above can be discarded by the same modification of simulator as Watrous [24]. We thus have the following completeness theorem for $\mathbf{HVQPZK}_1$.

Theorem 2. *Problem* QSI *is* $\mathbf{HVQPZK}_1$*-complete.*

We note that complete problems for **HVQPZK** and $\mathbf{HVQPZK}_1$ only differ up to a **BQP** instance. Does $\mathbf{HVQPZK} = \mathbf{HVQPZK}_1$? This is an interesting open problem. It is worthy noting that Kobayashi [14] showed that $\mathbf{HVQSZK} = \mathbf{HVQSZK}_1$ by giving a transformation. However, this transformation cannot be applied directly to perfect zero-knowledge quantum proof, because it will introduce an additional message which may not be perfectly output by simulator (though it can be approximated with exponentially small error).

In non-interactive model, we can also prove a completeness theorem with the same strategy as in interactive model, except that now the proof is adapted from Kobayashi [15].

Theorem 3. *Problem* QSII′ *is* **NIQPZK***-complete.*

5 Conclusion

Combining our results with [24] and [14], we can draw a table as below to summerize all complete problems we known for statistical and perfect zero-knowledge quantum proofs.

Complexity class	**QSZK**	$\mathbf{QPZK}_1$	**QPZK**	**NIQSZK**	$\mathbf{NIQPZK}_1$	**NIQPZK**
Complete problem	QSC	QSI	QSI′	QSCI	QSII	QSII′

We note that all these complete problems can be viewed as derived from problem QSC, comparing them may reveal the relationship among corresponding complexity classes.

References

1. Aharonov, D., Naveh, T.: Quantum NP - a survey (2002)
2. Arora, S., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press (2009)
3. Ben-Aroya, A., Schwartz, O., Ta-Shma, A.: Quantum expanders: Motivation and construction. Theory of Computing 6(1), 47–79 (2010)
4. Chailloux, A., Ciocan, D.F., Kerenidis, I., Vadhan, S.P.: Interactive and Noninteractive Zero Knowledge are Equivalent in the Help Model. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 501–534. Springer, Heidelberg (2008)
5. Damgård, I.B., Cramer, R.J.: On monotone function closure of perfect and statistical zero-knowledge. Technical report, Amsterdam, The Netherlands (1996)
6. Furer, M., Goldreich, O., Mansour, Y., Sipser, M., Zachos, S.: On completeness and soundness in interactive proof systems. In: Micali, S. (ed.) Randomness and Computation, Greenwich, Connecticut. Advances in Computing Research, vol. 5, pp. 429–442. JAI Press (1996)
7. Goldreich, O.: Foundations of Cryptography, Basic Tools, vol. I. Cambridge University Press (2001)

8. Goldreich, O.: Computational Complexity: A Conceptual Approach. Cambridge University Press (2008)
9. Goldreich, O., Sahai, A., Vadhan, S.P.: Can Statistical Zero Knowledge be Made Non-Interactive? or on the Relationship of SZK and NISZK. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, p. 467. Springer, Heidelberg (1999)
10. Goldreich, O., Vadhan, S.P.: Comparing entropies in statistical zero knowledge with applications to the structure of SZK. In: IEEE Conference on Computational Complexity, pp. 54–73 (1999)
11. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. 18(1), 186–208 (1989)
12. Kitaev, A.Y., Watrous, J.: Parallelization, amplification, and exponential time simulation of quantum interactive proof systems. In: STOC, pp. 608–617 (2000)
13. Kitaev, A.Y., Shen, A.H., Vyalyi, M.N.: Classical and Quantum Computation. In: American Mathematical Society. Graduate Studies in Mathematics, vol. 47. American Mathematical Society (2002)
14. Kobayashi, H.: General Properties of Quantum Zero-Knowledge Proofs. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 107–124. Springer, Heidelberg (2008), arXiv.org e-Print 0705.1129
15. Kobayashi, H.: Non-Interactive Quantum Perfect and Statistical Zero-Knowledge. In: Ibaraki, T., Katoh, N., Ono, H. (eds.) ISAAC 2003. LNCS, vol. 2906, pp. 178–188. Springer, Heidelberg (2003)
16. Malka, L.: How to Achieve Perfect Simulation and A Complete Problem for Non-Interactive Perfect Zero-Knowledge. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 89–106. Springer, Heidelberg (2008)
17. Marriott, C., Watrous, J.: Quantum Arthur-Merlin games. Computational Complexity 14(2), 122–152 (2005)
18. Nielsen, M.A., Chuang, I.L.: Quantum computation and Quantum Informatioin. Cambridge University Press (2000)
19. Sahai, A., Vadhan, S.P.: A complete problem for statistical zero knowledge. J. ACM 50(2), 196–249 (2003)
20. Santis, A.D., Crescenzo, G.D., Persiano, G., Yung, M.: On monotone formula closure of SZK. In: FOCS, pp. 454–465 (1994)
21. De Santis, A., Di Crescenzo, G., Persiano, G., Yung, M.: Image Density is Complete for Non-Interactive-SZK (Extended Abstract). In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) ICALP 1998. LNCS, vol. 1443, pp. 784–795. Springer, Heidelberg (1998)
22. Vadhan, S.: Ph.D Thesis: A Study of Statistical Zero-Knowledge Proofs (1999)
23. Watrous, J.: Succinct quantum proofs for properties of finite groups. In: FOCS, pp. 537–546 (2000)
24. Watrous, J.: Limits on the power of quantum statistical zero-knowledge. In: FOCS, pp. 459–468 (2002)
25. Watrous, J.: Theory of Quantum Information. Online Lecture Notes (2008), http://www.cs.uwaterloo.ca/~watrous/798/
26. Watrous, J.: Quantum computational complexity. In: Encyclopedia of Complexity and Systems Science, pp. 7174–7201 (2009)
27. Watrous, J.: Zero-knowledge against quantum attacks. SIAM J. Comput. 39(1), 25–58 (2009)
28. Yan, J.: Complete problem for perfect zero-knowledge quantum proof. Full version, http://lcs.ios.ac.cn/~junyan/Yan11_qpzk-SOFSEM12-final-full.pdf

An Algorithm for Probabilistic Alternating Simulation

Chenyi Zhang[1,2] and Jun Pang[3]

[1] University of Queensland, Brisbane, Australia
[2] University of New South Wales, Sydney, Australia
[3] University of Luxembourg, Luxembourg

Abstract. In probabilistic game structures, probabilistic alternating simulation (PA-simulation) relations preserve formulas defined in probabilistic alternating-time temporal logic with respect to the behaviour of a subset of players. We propose a partition based algorithm for computing the largest PA-simulation. It is to our knowledge the first such algorithm that works in polynomial time. Our solution extends the generalised coarsest partition problem (GCPP) to a game-based setting with mixed strategies. The algorithm has higher complexities than those in the literature for non-probabilistic simulation and probabilistic simulation without mixed actions, but slightly improves the existing result for computing probabilistic simulation with respect to mixed actions.

1 Introduction

Simulation and bisimulation relations are useful tools in the verification of finite and infinite state systems. State space minimisation modulo these relations is a valuable technique to fight the state explosion problem in model checking, since bisimulation preserves properties formulated in logics like CTL and CTL* [8] while simulation preserves the universal (or safe) fragment of these logics [14].

In some situations, however, it is necessary to model quantitative aspects of a system. It is the case, for instance, in wireless networks, where we often need to assume that there is a chance of connection failure with a given rate. This requires modelling network systems with randomised behaviours (e.g., by pooling a connection after uncertain amount of time to minimise conflict). Another important fact of real-world systems is that environment changes, such as unexpected power-off, are often unpredictable. Therefore, we need to encode appropriate system behaviours to handle such situations, and in order to do so, it is sometimes crucial to employ probabilistic strategies to achieve the best possible outcomes [24]. One simple example is the rock-scissor-paper game where there is no deterministic strategy to win since the other player's move is unknown, but there is a probabilistic strategy, sometimes called *mixed strategy*, to win at least a third of all cases in a row, regardless of what the other player does.[1]

[1] A mixed strategy also ensures an eventual win but deterministic strategies do not.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 431–442, 2012.

A probabilistic game structure (PGS) is a model that has probabilistic transitions, and allows the consideration of probabilistic choices of players. The simulation relation in PGSs, called probabilistic alternating simulation (PA-simulation), has been shown to preserve a fragment of probabilistic alternating-time temporal logic (PATL) under *mixed strategies*, which is used in characterising what a group of players can enforce in such systems [25]. In this paper we propose a polynomial-time algorithm for computing the largest PA-simulation, which is, to the best of our knowledge, the first algorithm for computing a simulation relation in probabilistic concurrent games. A PGS combines the modelling of probabilistic transitions from probabilistic automata (PA), and the user interactions from concurrent game structures (GS). In PA, the probabilistic notions of simulation preserve PCTL safety formulas [20]. The *alternating simulation* [2] in GS has been been proved to preserve a fragment of ATL*, under the semantics of *deterministic strategies*. These simulation relations are computable in polynomial time for finite systems [26,2].

Related work. Efficient algorithms have been proposed for computing the largest simulation (e.g., see [15,22,4,13,23]) in finite systems, with a variety of time and space complexities. In particular, Gentilini et al. [13] developed an efficient algorithm with an improved time complexity based on the work of Henzinger et al. [15] without losing the optimal space complexity. Van Glabbeek and Ploeger [23] later found a flaw in [13] and proposed a non-trivial fix. So far the best algorithm for time complexity is [18]. To compute probabilistic simulation, Baier et al. [3] reduce the problem of establishing a weight function for the lifted relation to a maximal flow problem. Cattani and Segala [5] reduce the problem of deciding strong probabilistic bisimulation to LP problems. Zhang and Hermanns [27] develop algorithms with improved time complexity for probabilistic simulations, following [3,5]. A space efficient probabilistic simulation algorithm is proposed by Zhang [26] using the techniques proposed in [13,23].

Studies on stochastic games have actually been carried out since as early as the 1950s [21], and a rich literature has developed in recent years (e.g. see [10,9,11,6]). One existing approach called game metrics [12] defines approximation-based simulation relations, with a kernel simulation characterising the logic quantitative game μ-calculus [9], an extension of modal μ-calculus for concurrent games where each state is assigned a quantitative value in $[0, 1]$ for every formula. However, so far the best solutions in the literature on approximating the simulation as defined in the metrics for concurrent games potentially take exponential time [7]. Although PA-simulation is strictly stronger than the kernel simulation relation of the game metrics in [12], the algorithm presented in the paper has a more tractable complexity result, and we believe that it will benefit the abstraction or refinement based techniques for verifying game-based properties.

2 Preliminaries

Probabilistic game structures are defined in terms of discrete probabilistic distributions. A *discrete probabilistic distribution* Δ over a finite set S is a function

of type $S \to [0,1]$, where $\sum_{s \in S} \Delta(s) = 1$. We write $\mathcal{D}(S)$ for the set of all such distributions on a fixed S. For a set $T \subseteq S$, define $\Delta(T) = \sum_{s \in T} \Delta(s)$. Given a finite index set I, a list of distributions $(\Delta_i)_{i \in I}$ and a list of probabilities $(p_i)_{i \in I}$ where, for all $i \in I$, $p_i \in [0,1]$ and $\sum_{i \in I} p_i = 1$, $\sum_{i \in I} p_i \Delta_i$ is obviously also a distribution. For $s \in S$, $\overline{s}$ is called a *point (or Dirac) distribution* satisfying $\overline{s}(s) = 1$ and $\overline{s}(t) = 0$ for all $t \neq s$. Given $\Delta \in \mathcal{D}(S)$, we define $\lceil \Delta \rceil$ as the set $\{s \in S \mid \Delta(s) > 0\}$, which is the *support* of Δ.

In this paper we assume a set of two players $\{\mathtt{I}, \mathtt{II}\}$ (though our results can be extended to handle a finite set of players as in the standard game structure and ATL semantics [1]), and *Prop* a finite set of propositions.

Definition 1. *A probabilistic game structure $\mathcal{G}$ is a tuple $\langle S, s_0, \mathcal{L}, Act, \delta \rangle$, where*

- *S is a finite set of states, with s_0 the initial state;*
- *$\mathcal{L} : S \to 2^{Prop}$ is the labelling function which assigns to each state $s \in S$ a set of propositions that are true in s;*
- *$Act = Act_{\mathtt{I}} \times Act_{\mathtt{II}}$ is a finite set of joint actions, where $Act_{\mathtt{I}}$ and $Act_{\mathtt{II}}$ are, respectively, the sets of actions for players $\mathtt{I}$ and $\mathtt{II}$;*
- *$\delta : S \times Act \to \mathcal{D}(S)$ is a transition function.*

If in state s player $\mathtt{I}$ performs action a_1 and player $\mathtt{II}$ performs action a_2 then $\delta(s, \langle a_1, a_2 \rangle)$ is the distribution for the next states. During each step the players choose their next moves simultaneously. We define a *mixed action* of player $\mathtt{I}$ ($\mathtt{II}$) as a distribution over $Act_{\mathtt{I}}$ ($Act_{\mathtt{II}}$), and write $\Pi_{\mathtt{I}}$ ($\Pi_{\mathtt{II}}$) for the set of mixed actions of player $\mathtt{I}$ ($\mathtt{II}$).[2] In particular, $\overline{a}$ is a *deterministic* mixed action which always chooses a. We lift the transition function δ to handle mixed actions. Given $\pi_1 \in \Pi_{\mathtt{I}}$ and $\pi_2 \in \Pi_{\mathtt{II}}$, for all $s, t \in S$, we have

$$\overline{\delta}(s, \langle \pi_1, \pi_2 \rangle)(t) = \sum_{a_1 \in Act_{\mathtt{I}}, a_2 \in Act_{\mathtt{II}}} \pi_1(a_1) \cdot \pi_2(a_2) \cdot \delta(s, \langle a_1, a_2 \rangle)(t)$$

Simulation relations in probabilistic systems require a definition of *lifting* [16], which extends the relations to the domain of distributions.[3] Let S, T be two sets and $\mathcal{R} \subseteq S \times T$ be a relation, then $\overline{\mathcal{R}} \subseteq \mathcal{D}(S) \times \mathcal{D}(T)$ is a *lifted relation* defined by $\Delta \,\overline{\mathcal{R}}\, \Theta$ if there exists a weight function $w : S \times T \to [0,1]$ such that (1) $\sum_{t \in T} w(s,t) = \Delta(s)$ for all $s \in S$, (2) $\sum_{s \in S} w(s,t) = \Theta(t)$ for all $t \in T$, (3) $s \mathcal{R} t$ for all $s \in S$ and $t \in T$ with $w(s,t) > 0$.

The intuition behind the lifting is that each state in the support of one distribution may correspond to a number of states in the support of the other distribution, and vice versa. The example in Fig. 1 is taken from [19] to show how to lift one relation. We have two set of states $S = \{s_1, s_2\}$ and $T = \{t_1, t_2, t_3\}$, and $\mathcal{R} = \{(s_1, t_1), (s_1, t_2), (s_2, t_2), (s_2, t_3)\}$. We have $\Delta \,\overline{\mathcal{R}}\, \Theta$, where $\Delta(s_1) = \Delta(s_2) =$

[2] Note $\Pi_{\mathtt{I}}$ is equivalent to $\mathcal{D}(Act_{\mathtt{I}})$, though we choose a different symbol because the origin of a mixed action is a simplified *mixed strategy* of player $\mathtt{I}$ which has type $S^+ \to \mathcal{D}(Act_{\mathtt{I}})$. A mixed action only considers player $\mathtt{I}$'s current step.

[3] In a probabilistic system without explicit user interactions, state s is simulated by state t if for every $s \xrightarrow{a} \Delta_1$ there exists $t \xrightarrow{a} \Delta_2$ such that Δ_1 is simulated by Δ_2.

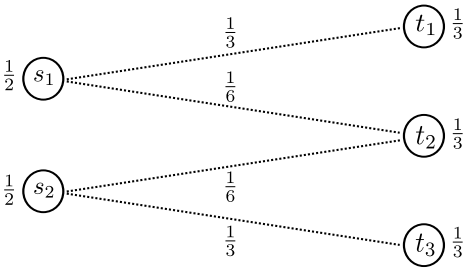


Fig. 1. An example showing how to lift one relation

$\frac{1}{2}$ and $\Theta(t_1) = \Theta(t_2) = \Theta(t_3) = \frac{1}{3}$. To check this, we define a weight function w by: $w(s_1, t_1) = \frac{1}{3}$, $w(s_1, t_2) = \frac{1}{6}$ $w(s_2, t_2) = \frac{1}{6}$, and $w(s_2, t_3) = \frac{1}{3}$. The dotted lines indicate the allocation of weights required to relate Δ to Θ via $\overline{\mathcal{R}}$. By lifting in this way, we are able to extend the notion of alternating simulation [2] to a probabilistic setting.

Definition 2. *Given a PGS* $\langle S, s_0, \mathcal{L}, Act, \delta \rangle$*, a* probabilistic alternating I-simulation *(PA-*$\mathtt{I}$*-simulation) is a relation* $\sqsubseteq \subseteq S \times S$ *such that if* $s \sqsubseteq t$*, then*

- $\mathcal{L}(s) = \mathcal{L}(t)$,
- *for all* $\pi_1 \in \Pi_I$*, there exists* $\pi_1' \in \Pi_I$*, such that for all* $\pi_2' \in \Pi_{II}$*, there exists* $\pi_2 \in \Pi_{II}$*, such that* $\overline{\delta}(s, \langle \pi_1, \pi_2 \rangle) \; \overline{\sqsubseteq} \; \overline{\delta}(t, \langle \pi_1', \pi_2' \rangle)$.

If s PA-I-simulates t and t PA-I-simulates s, we say s and t are *PA-*$\mathtt{I}$*-simulation equivalent*.[4]

PA-I-simulation has been shown to preserve a fragment of PATL which covers the ability of player I to enforce certain temporal requirements [25]. For example, if in state s player I can enforce reaching some states satisfying p within 5 transition steps and with probability at least $\frac{1}{2}$, written $s \models \langle\langle \mathtt{I} \rangle\rangle^{\geq \frac{1}{2}} \Diamond^{\leq 5} p$, then for every state t that simulates s with respect to I, i.e., $s \sqsubseteq t$ by some PA-I-simulation '$\sqsubseteq$', we also have $t \models \langle\langle \mathtt{I} \rangle\rangle^{\geq \frac{1}{2}} \Diamond^{\leq 5} p$.

General Coarsest Partition Problem

The general coarsest partition problem (GCPP) provides a characterisation of (non-probabilistic) simulation in finite state transition systems [13]. Informally, in this approach, states that are (non-probabilistic) simulation equivalent are grouped into the same block, and all such blocks form a partition over the (finite) state space. Based on the partition, blocks are further related by a partial order $\preceq$, so that if $P \preceq Q$, then every state in block P is simulated by every state in block Q. The GCPP is to find, for a given PGS, the smallest such set of blocks. In the literature such a methodology yields space efficient algorithms for computing the largest (non-probabilistic) simulation relation in a finite system [13,23]. Similar methods have been adopted and developed to compute the largest simulation relations in the model of probabilistic automata [26].

[4] Alternating simulations and equivalences are for player I unless stated otherwise.

We briefly review the basic notions that are required to present the GCPP problem. A *partition* over a set S, is a collection $\Sigma \subseteq \mathcal{P}(S)$ satisfying (1) $\bigcup \Sigma = S$ and (2) $P \cap Q = \emptyset$ for all distinct *blocks* $P, Q \in \Sigma$. Given $s \in S$, write $[s]_\Sigma$ for the block in partition Σ that contains s. A partition Σ_1 is *finer* than Σ_2, written $\Sigma_1 \lhd \Sigma_2$, if for all $P \in \Sigma_1$ there exists $Q \in \Sigma_2$ such that $P \subseteq Q$.

Given a set S, a *partition pair* over S is $(\Sigma, \preceq)$ where Σ is a partition over S and $\preceq \subseteq \Sigma \times \Sigma$ is a partial order. Write $Part(S)$ for the set of partition pairs on S. If $\Upsilon \lhd \Sigma$ and $\preceq$ is a relation on Σ, then $\preceq(\Upsilon) = \{(P, Q) \mid P, Q \in \Upsilon, \exists P', Q' \in \Sigma, P \subseteq P', Q \subseteq Q', P' \preceq Q'\}$ is the relation on Υ *induced* by $\preceq$. Let $(\Sigma_1, \preceq_1)$ and $(\Sigma_2, \preceq_2)$ be partition orders, write $(\Sigma_1, \preceq_1) \leq (\Sigma_2, \preceq_2)$ if $\Sigma_1 \lhd \Sigma_2$, and $\preceq_1 \subseteq \preceq_2 (\Sigma_1)$. Define a relation $\sqsubseteq_{(\Sigma,\preceq)} \subseteq S \times S$ as determined by a partition pair $(\Sigma, \preceq)$ by $s \sqsubseteq_{(\Sigma,\preceq)} t$ iff $[s]_\Sigma \preceq [t]_\Sigma$.

Let $\rightarrow \subseteq S \times S$ be a (transition) relation and $\mathcal{L} : S \to 2^{Prop}$ a labelling function, then a relation $\sqsubseteq$ is a simulation on S if for all $s, t \in S$ with $s \sqsubseteq t$, we have (1) $\mathcal{L}(s) = \mathcal{L}(t)$ and (2) $s \rightarrow s'$ implies that there exists t' such that $t \rightarrow t'$ and $s' \sqsubseteq t'$. Let $(\Sigma, \preceq)$ be a partition pair on S, then it is *stable* with respect to $\rightarrow$ if for all $P, Q \in \Sigma$ with $P \preceq Q$ and $s \in P$ such that $s \rightarrow s'$ with $s' \in P' \in \Sigma$, then there exists $Q' \in \Sigma$ such that $P' \preceq Q'$ and for all $t \in Q$, there exists $t' \in Q'$ such that $t \rightarrow t'$. The following result is essential to the GCPP approach, as we derive the largest simulation relation by computing the coarsest stable partition pair over a finite state space.[5]

Proposition 1. *[13,23] Let $(\Sigma, \preceq)$ be a partition pair, then it is stable with respect to $\rightarrow$ iff the induced relation $\sqsubseteq_{(\Sigma,\preceq)}$ is a simulation (with respect to $\rightarrow$).*

Given a transition relation on a state space there exists a unique largest simulation relation. Thus, solutions to GCPP provide the coarsest stable partition pairs, and they have been proved to characterise the largest simulation relations in non-probabilistic systems [13,23].

3 Solving GCPP in Probabilistic Game Structures

In this section we extend the GCPP framework to characterise PA-simulations in PGSs. Given a PGS $\mathcal{G} = \langle S, s_0, \mathcal{L}, Act, \delta \rangle$, a *partition pair* over $\mathcal{G}$ is $(\Sigma, \preceq)$ where Σ is a partition over S. Write $Part(\mathcal{G})$ for the set of all partition pairs over S. We show how to compute the coarsest partition pair and prove that it characterises the largest PA-simulation for a given player.

Since in probabilistic systems transitions go from states to distributions over states, we first present a probabilistic version of *stability*, as per [26]. Let $\rightarrow \subseteq S \times \mathcal{D}(S)$ be a probabilistic (transition) relation. For a distribution $\Delta \in \mathcal{D}(S)$ and Σ a partition, write Δ_Σ as a distribution on Σ defined by $\Delta_\Sigma(P) = \Delta(P)$ for all $P \in \Sigma$. Let $(\Sigma, \preceq)$ be a partition pair, it is *stable* with respect to the

[5] We choose the word *coarsest* for partition pairs to make it consistent with the standard term GCPP, and it is clear in the context that *coarsest* carries the same meaning as *largest* with respect to the order $\leq$ defined on partition pairs.

relation $\rightarrow$, if for all $P, Q \in \Sigma$ with $P \preceq Q$ and $s \in P$ such that $s \rightarrow \Delta$, then for all $t \in Q$ there exists $t \rightarrow \Theta$ such that $\Delta_\Sigma \overline{\preceq} \Theta_\Sigma$.

Another obstacle in characterising PA-simulation is that the concerned player can only partially determine a transition. That is, after player I performs an action on a state, the exact future distribution on next states depends on an action from player II. Therefore, we need to (again) lift the stability condition for PA-I-simulation from distributions to sets of distributions.

Let $\leq \subseteq S \times S$ be a partial order on a set S, define $\leq_{Sm} \subseteq \mathcal{P}(S) \times \mathcal{P}(S)$, by $P \leq_{Sm} Q$ if for all $t \in Q$ there exists $s \in P$ such that $s \leq t$. In the literature this definition is known as a 'Smyth order'. In a PGS, we 'curry' the transition function by defining $\overline{\delta}(s, \pi_1) = \{\overline{\delta}(s, \langle \pi_1, \pi_2 \rangle) \mid \pi_2 \in \Pi_{\mathtt{II}}\}$, which is the set of distributions that are possible if player I takes a mixed action $\pi_1 \in \Pi_{\mathtt{I}}$ on $s \in S$.

Definition 3. *(lifted stability) Let $(\Sigma, \preceq)$ be a partition pair on S in a PGS, it is stable with respect to player I's choice, if for all $\pi \in \Pi_I$, $P, Q \in \Sigma$ with $P \preceq Q$ and $s \in P$, there exists $\pi' \in \Pi_I$ such that $\overline{\delta}(s, \pi)_\Sigma \overline{\preceq}_{Sm} \overline{\delta}(t, \pi')_\Sigma$ for all $t \in Q$.*

Intuitively, the Smyth order captures the way of *behavioral* simulation. That is, if $\overline{\delta}(t, \pi')$ is at least as restrictive as $\overline{\delta}(s, \pi)$, then whatever player I is able to enforce by performing π in s, he can also enforce it by performing π' in t, as player II has *fewer* choices in $\overline{\delta}(t, \pi')$ than in $\overline{\delta}(s, \pi)$. At this point, for the sake of readability, if it is clear from the context, we write W for W_Σ as the distribution W mapped onto partition Σ.

For simulation relations, it is also required that the related states agree on their labelling. Define Σ_0 as the *labelling partition* satisfying for all $s, t \in S$, $\mathcal{L}(s) = \mathcal{L}(t)$ iff $[s]_{\Sigma_0} = [t]_{\Sigma_0}$. Write $Part^0(\mathcal{G}) \subseteq Part(\mathcal{G})$ for the set of partition pairs $(\Sigma, \preceq)$ satisfying $(\Sigma, \preceq) \leq (\Sigma_0, \mathtt{Id})$, where Id is the identity relation.

Lemma 1. *For all $(\Sigma, \preceq) \in Part^0(\mathcal{G})$, if $(\Sigma, \preceq)$ is a stable partition pair with respect to player I's choice then $\sqsubseteq_{(\Sigma, \preceq)}$ is a PA-I-simulation.*

Obviously every PA-I-simulation is contained in the relation induced by $(\Sigma_0, \mathtt{Id})$, and moreover, the above lemma asserts that every stable partition pair smaller than $(\Sigma_0, \mathtt{Id})$ is a PA-I-simulation. In the following, we try to compute the coarsest partition pair by refining $(\Sigma_0, \mathtt{Id})$ until it stabilises. The resulting stable partition pair can be proved to characterise the largest PA-I-simulation on the state space S as required.

We say t simulates s with respect to player-I's choice on a partition pair $(\Sigma, \preceq)$ if for all $\pi \in \Pi_{\mathtt{I}}$, there exists $\pi' \in \Pi_{\mathtt{I}}$ such that $\overline{\delta}(s, \pi) \overline{\preceq}_{Sm} \overline{\delta}(t, \pi')$. For better readability, sometimes we also say t simulates s on $(\Sigma, \preceq)$ if it is clear from the context. Let $(\Sigma_1, \preceq_1) \leq (\Sigma_2, \preceq_2)$, we say $(\Sigma_1, \preceq_1)$ is stable on $(\Sigma_2, \preceq_2)$, if for all $P, Q \in \Sigma_1$ with $P \preceq_1 Q$, $s \in P$ and $t \in Q$, t simulates s on $(\Sigma_2, \preceq_2)$.

Definition 4. *Define an operator $\rho : Part(\mathcal{G}) \rightarrow Part(\mathcal{G})$, such that $\rho((\Sigma, \preceq))$ is the largest partition pair $(\Sigma', \preceq') \leq (\Sigma, \preceq)$ that is stable on $(\Sigma, \preceq)$.*

The operator ρ has the following properties.

Lemma 2. *1) ρ is well defined on $Part(\mathcal{G})$. 2) ρ is monotonic on $(Part^0(\mathcal{G}), \leq)$.*

Lemma 1 ensures that for all $(\Sigma, \preceq) \in Part^0(\mathcal{G})$, $\sqsubseteq_{(\Sigma,\preceq)}$ is a PA-I-simulation if $\rho((\Sigma, \preceq)) = (\Sigma, \preceq)$, i.e., $(\Sigma, \preceq)$ is a fixpoint of ρ. However, we still need to find the largest PA-I-simulation. The following result indicates that if S is finite, the coarsest stable partition pair achieved by repetitively applying ρ on $(\Sigma_0, \mathtt{Id})$ indeed yields the largest PA-I-simulation.[6] Define $\rho^0(X) = X$ and $\rho^{n+1}(X) = \rho(\rho^n(X))$ for partition pairs X.

Theorem 1. *Let* $(\Sigma, \preceq) = \bigcap_{i\in\mathbb{N}} \rho^i((\Sigma_0, \mathtt{Id}))$*, then* $\sqsubseteq_{(\Sigma,\preceq)}$ *is the largest PA-I-simulation on* $\mathcal{G}$*.*

Proof. (sketch) Let $\sqsubseteq^+$ be the largest PA-I-simulation on $\mathcal{G}$. Define a set $\Sigma^+ = \{\{t \in S \mid s \sqsubseteq^+ t \wedge t \sqsubseteq^+ s\} \mid s \in S\}$. Since $\sqsubseteq^+$ is the largest PA-I-simulation, it can be shown that $\sqsubseteq^+$ is reflexive, symmetric and transitive within each block $P \in \Sigma^+$. Moreover, we define a relation $\preceq^+$ by $P \preceq^+ Q$ if there exists $s \in P$ and $t \in Q$ such that $s \sqsubseteq^+ t$, and it can be shown that $\preceq^+$ is a partition order on Σ^+. Then $(\Sigma^+, \preceq^+)$ forms a partition pair on $\mathcal{G}$, and furthermore, it is stable, and we also have $(\Sigma^+, \preceq^+) \leq (\Sigma_0, \mathtt{Id})$.

We apply ρ on both sides. By Lemma 2(2) (monotonicity), and $(\Sigma^+, \preceq^+)$ being stable, we have $(\Sigma^+, \preceq^+) = \rho^i((\Sigma^+, \preceq^+)) \leq \rho^i((\Sigma_0, \mathtt{Id}))$ for all $i \in \mathbb{N}$. As $Part(\mathcal{G})$ is finite, there exists $j \in \mathbb{N}$, such that $\rho^j((\Sigma_0, \mathtt{Id})) = \rho^{j+1}((\Sigma_0, \mathtt{Id}))$. Therefore, $\rho^j((\Sigma_0, \mathtt{Id}))$ is a stable partition pair, and $\sqsubseteq_{\rho^j((\Sigma_0,\mathtt{Id}))}$ is a PA-I-simulation by Lemma 1. Straightforwardly we have $\sqsubseteq^+ \subseteq \sqsubseteq_{\rho^j((\Sigma_0,\mathtt{Id}))}$. Since $\sqsubseteq^+$ is the largest PA-I-simulation by assumption, we have $\sqsubseteq^+ = \sqsubseteq_{\rho^j((\Sigma_0,\mathtt{Id}))}$, and the result directly follows. □

4 A Decision Procedure for PA-I-Simulation

Efficient algorithms for simulation in the non-probabilistic setting sometimes apply predecessor based methods [15,13] for splitting blocks and refining partitions. This method can no longer be applied for simulations in the probabilistic setting, as the transition functions now map a state to a state distribution rather than a single state, and simulation relation needs to be *lifted* to handle distributions. The algorithms in [27,26] follow the approaches in [3] by reducing the problem of deciding a weight function on lifted relations to checking the value of a maximal flow problem. This method, however, does *not* apply to combined transitions, where a more general solution is required. Algorithms for deciding probabilistic bisimulations [5] reduce the problem on checking weight functions with combined choices to solutions in linear programming (LP), which are known to be decidable in polynomial time [17].[7]

Simulation relations are characterised by partition pairs in the solutions to the GCPP. We propose the following characterisation of lifting in order to handle

[6] The following proof resembles the classical paradigm of finding the least fixpoint in an ω-chain of a complete partial order by treating $(\Sigma_0, \mathtt{Id})$ as $\bot$. However, here we also need that fixpoint to represent the largest PA-I-simulation.

[7] The maximal flow problem is a special instance of an LP problem, which can be solved more efficiently.

the partial order relation on partitions. Let S be a finite set and $\preceq$ a partial order on S. Define $\lfloor s \rfloor_{\preceq} = \{t \in S \mid s \preceq t\}$, which is called the *up-closure* of s. The following lemma reduces the problems of finding a weight function for two distributions on a partition pair to comparing weights of each up-closed block, and the latter problem can be easily encoded in LP when checking PA-I-simulation on a given partition pair between two states (as shown in Lemma 7).

Lemma 3. *Let S be a set with a partial order $\preceq \subseteq S \times S$ and $\Delta_1, \Delta_2 \in \mathcal{D}(S)$, then $\Delta_1 \overline{\preceq} \Delta_2$ iff we have $\Delta_1(\lfloor s \rfloor_{\preceq}) \leq \Delta_2(\lfloor s \rfloor_{\preceq})$ for all $s \in S$.*

When deciding whether s is able to simulate t with respect to I's choice on a certain partition pair, we need to examine potentially infinitely many mixed actions in $\Pi_{\mathtt{I}}$. This problem can be moderated by the following observations. First we show that for s to be simulated by t, it is only required to check all deterministic choices of player I on s.

Lemma 4. *Let $(\Sigma, \preceq)$ be a partition pair, then t simulates s on $(\Sigma, \preceq)$ if for all $a \in Act_I$, there exists $\pi \in \Pi_I$ such that $\overline{\delta}(s, \overline{a}) \overline{\preceq}_{Sm} \overline{\delta}(t, \pi)$.*

The next lemma states that for checking a Smyth order $\overline{\delta}(s, \pi) \overline{\preceq}_{Sm} \overline{\delta}(t, \pi')$, it suffices to focus on player II's deterministic choices in $\overline{\delta}(t, \pi')$, since all probabilistic choices can be represented as interpolations from deterministic choices.

Lemma 5. *$\overline{\delta}(s, \pi) \overline{\preceq}_{Sm} \overline{\delta}(t, \pi')$ if for all $a \in Act_{II}$, there exists $\pi'' \in \Pi_{II}$ such that $\overline{\delta}(s, \langle \pi, \pi'' \rangle) \overline{\preceq} \overline{\delta}(t, \langle \pi', \overline{a} \rangle)$.*

Combining the above two lemmas, we have the following.

Lemma 6. *Let $(\Sigma, \preceq)$ be a partition pair, then t simulates s with respect to player-I's choice on $(\Sigma, \preceq)$ if for all $a_1 \in Act_I$, there exists $\pi_1 \in \Pi_I$ such that for all $a_2 \in Act_{II}$, there exists $\pi_2 \in \Pi_{II}$ such that $\overline{\delta}(s, \langle \overline{a_1}, \pi_2 \rangle) \overline{\preceq} \overline{\delta}(t, \langle \pi_1, \overline{a_2} \rangle)$.*

The following lemma states how to check if the action a can be followed by a mixed action from $\Pi_{\mathtt{I}}$.

Lemma 7. *Given a partition pair $(\Sigma, \preceq)$, two states $s, t \in S$ and $a \in Act_I$, there exists $\pi \in \Pi_I$ such that $\overline{\delta}(s, \overline{a}) \overline{\preceq}_{Sm} \overline{\delta}(t, \pi)$, iff the following LP has a solution: Let $Act_I = \{a_1, a_2, \ldots, a_\ell\}$ and $Act_{II} = \{b_1, b_2, \ldots, b_m\}$*

$$\sum_{i=1}^{\ell} \alpha_i = 1 \tag{1}$$

$$\forall i = 1, 2, \ldots, \ell : 0 \leq \alpha_i \leq 1 \tag{2}$$

$$\forall j = 1, 2, \ldots, m : \sum_{k=1}^{m} \beta_{j,k} = 1 \tag{3}$$

$$\forall j, k = 1, 2, \ldots, m : 0 \leq \beta_{j,k} \leq 1 \tag{4}$$

$\forall B \in \Sigma : j = 1, 2, \ldots, m :$

$$\sum_{k=1}^{m} \beta_{j,k} \cdot \delta(s, a, b_k)(\lfloor B \rfloor_{\preceq}) \leq \sum_{i=1}^{\ell} \alpha_i \cdot \delta(t, a_i, b_j)(\lfloor B \rfloor_{\preceq}) \tag{5}$$

Here $\alpha_1, \alpha_2, \ldots, \alpha_\ell$ are used to 'guess' a mixed action from player $\mathtt{I}$, as constrained in Eq. 1 and Eq. 2. To establish the Smyth order $\overline{\preceq}_{Sm}$, by Lemma 6, for every player $\mathtt{II}$ action b_j with $j = 1, 2, \ldots, m$, we 'guess' a mixed action from $Act_{\mathtt{II}}$ represented by $\beta_{j,1}, \beta_{j,2} \ldots, \beta_{j,m}$, as constrained in Eq. 3 and Eq. 4. Then for each block B in Σ, the established distributions need to satisfy the lifted relation $\overline{\preceq}$, which is characterised by the inequalities on the up-closure of B with respect to the order $\preceq$, by Lemma 3.

We define a predicate $\mathsf{CanFollow}$ such that $\mathsf{CanFollow}((\Sigma, \preceq), s, t, a)$ decides whether there exists a mixed action of player $\mathtt{I}$ from t which simulates action $a \in Act_{\mathtt{I}}$ from s on the partition pair $(\Sigma, \preceq)$. $\mathsf{CanFollow}$ establishes an LP problem from its parameters (see Lemma 7). We further define a predicate CanSim which decides whether a state simulates another with respect to player $\mathtt{I}$'s choice on $(\Sigma, \preceq)$ for all actions in $Act_{\mathtt{I}}$, i.e., $\mathsf{CanSim}((\Sigma, \preceq), s, t)$ returns *true* if $\mathsf{CanFollow}((\Sigma, \preceq), s, t, a)$ returns *true* for all $a \in Act_{\mathtt{I}}$.

Algorithm 1. Refining a block to make it stable on a partition pair

INPUT: a partition pair $(\Sigma, \preceq)$, a block $B \in \Sigma$
OUTPUT: a partition pair $(\Sigma_B, \preceq_B)$ on B
function Split $((\Sigma, \preceq), B)$
  $\Sigma_B := \{\{s\} \mid s \in B\}$; $\preceq_B := \{(s, s) \mid s \in B\}$; $\Sigma' := \emptyset$; $\preceq' := \emptyset$
  while $\Sigma_B \neq \Sigma' \vee \preceq_B \neq \preceq'$ **do**
    $\Sigma' := \Sigma_B$; $\preceq' := \preceq_B$
    for each *distinct* $B_1, B_2 \in \Sigma_B$ **do**
      pick any $s_1 \in B_1$ *and* $s_2 \in B_2$
      if $(\mathsf{CanSim}((\Sigma, \preceq), s_1, s_2) \wedge \mathsf{CanSim}((\Sigma, \preceq), s_2, s_1))$ **then**
        $\Sigma_B := \Sigma_B \setminus \{B_1, B_2\} \cup \{B_1 \cup B_2\}$
        $\preceq_B := \preceq_B \cup \{(X, B_1 \cup B_2) \mid X \in \Sigma : (X, B_1) \in \preceq_B \vee (X, B_2) \in \preceq_B\}$
            $\cup \{(B_1 \cup B_2, X) \mid X \in \Sigma : (B_1, X) \in \preceq_B \vee (B_2, X) \in \preceq_B\}$
            $\setminus \{(B_i, X), (X, B_i) \mid X \in \Sigma : (B_i, X), (X, B_i) \in \preceq_B \wedge i \in \{1, 2\}\}$
      else if $(\mathsf{CanSim}((\Sigma, \preceq), s_1, s_2))$ **then**
        $\preceq_B := \preceq_B \cup \{(B_2, B_1)\}$
      else if $(\mathsf{CanSim}((\Sigma, \preceq), s_2, s_1))$ **then**
        $\preceq_B := \preceq_B \cup \{(B_1, B_2)\}$
    endfor
  endwhile
  return $(\Sigma_B, \preceq_B)$

Algorithm 1 defines a function Split which refines a block $B \in \Sigma$ into a partition pair corresponding the maximal simulation that is stable on $(\Sigma, \preceq)$. It starts with the finest partition and the identity relation (as the final relation is reflexive). For each pair of blocks in the partition, we check if they can simulate each other by picking up a state from each block. If they are simulation equivalent on $(\Sigma, \preceq)$ then we merge the two blocks as well as all incoming and

outgoing relation in the current partial order. If only one simulates the other we add an appropriate pair into the current ordering. This process continues until the partition pair stablises.

Algorithm 2. Computing the Generalised Coarsest Partition Pair

INPUT: a probabilistic game structure $\mathcal{G} = \langle S, s_0, \mathcal{L}, Act, \delta \rangle$
OUTPUT: a partition pair $(\Sigma, \preceq)$ on S
function GCPP $(\mathcal{G})$
  $\Sigma := \{\{t \mid \mathcal{L}(t) = \mathcal{L}(s)\} \mid s \in S\}$; $\preceq := \{(B, B) \mid B \in \Sigma\}$
  $\Sigma' := \emptyset$; $\preceq' := \emptyset$
  while $\Sigma \neq \Sigma' \vee \preceq \neq \preceq'$ **do**
    $\Sigma' := \Sigma$; $\preceq' := \preceq$
    for each $B \in \Sigma$ **do**
      $(\Sigma_B, \preceq_B) := \mathsf{Split}((\Sigma', \preceq'), B)$
      $\Sigma := \Sigma \setminus \{B\} \cup \Sigma_B$
      $\preceq := \preceq \cup \preceq_B$
          $\cup \{(B', X) \mid X \in \Sigma : B' \in \Sigma_B : (B, X) \in \preceq\}$
          $\cup \{(X, B') \mid X \in \Sigma : B' \in \Sigma_B : (X, B) \in \preceq\}$
          $\setminus \{(B, X), (X, B) \mid X \in \Sigma : (X, B), (B, X) \in \preceq\}$
    endfor
  endwhile
  return $(\Sigma, \preceq)$

Algorithm 2 is based on the functionality of Split in Algorithm 1. Starting from the partition $(\Sigma_0, \mathtt{Id})$, which is identified as $(\{\{t \mid \mathcal{L}(t) = \mathcal{L}(s)\} \mid s \in S\}, \{(B, B) \mid B \in \Sigma_0\})$, the algorithm computes a sequence of partition pairs $(\Sigma_1, \preceq_1), (\Sigma_2, \preceq_2) \ldots$ until it stabilises, which is detected by checking the condition $\Sigma \neq \Sigma' \vee \preceq \neq \preceq'$. At each iteration we have $(\Sigma_{i+1}, \preceq_{i+1}) \leq (\Sigma_i, \preceq_i)$, and moreover, $(\Sigma_{i+1}, \preceq_{i+1})$ is the maximal partition pair that is stable on $(\Sigma_i, \preceq_i)$. The correctness of the algorithm is justified by Theorem 1, which states that it converges to the coarsest partition pair that is contained in $(\Sigma_0, \mathtt{Id})$ and returns a representation of the largest PA-I-simulation.

Space complexity. For a PGS $\langle S, s_0, \mathcal{L}, Act, \delta \rangle$, it requires $\mathcal{O}(|S|)$ to store the state space and $\mathcal{O}(|S|^2 \cdot |Act|)$ for the transition relation, since for each $s \in S$ and $\langle a_1, a_2 \rangle \in Act$ it requires an array of size $\mathcal{O}(|S|)$ to store a distribution. Recording a partition pair takes $\mathcal{O}(|S| \log |S| + |S|^2)$ as the first part is needed to record for each state which equivalence class in the partition it belongs, and the second part is needed for the partial order relation $\preceq$ which takes at most $\mathcal{O}(|S|^2)$. The computation from $(\Sigma_i, \preceq_i)$ to $(\Sigma_{i+1}, \preceq_{i+1})$ can be done in-place which only requires additional constant space to track if the partition pair has been modified during each iteration. Another extra space-consuming part is for solving LP constrains, which we assume has space usage $\mathcal{O}(\gamma(N))$ where $N = 1 + |Act_{\mathrm{I}}| + |Act_{\mathrm{II}}| + |Act_{\mathrm{II}}|^2 + |S| \cdot |Act_{\mathrm{II}}|$ is the number of linear constraints at most, and $\gamma(N)$ some polynomial. The space complexity roughly sums up to $\mathcal{O}(|S|^2 \cdot |Act| + |S| \log |S| + \gamma(|Act|^2 + |S| \cdot |Act|))$. (The first part $\mathcal{O}(|S|^2 \cdot |Act| + |S| \log |S|)$ for the PGS itself can be considered optimal, while the second part depends on the efficiency of the LP algorithm being used.)

Time complexity. The number of variables in the LP problem in Lemma 7 is $|Act_{\mathrm{I}}|+|Act_{\mathrm{II}}|^2$, and the number of constraints is bounded by $1+|Act_{\mathrm{I}}|+|Act_{\mathrm{II}}|+|Act_{\mathrm{II}}|^2+|S|\cdot|Act_{\mathrm{II}}|$. The predicate CanSim costs $|Act_{\mathrm{I}}|$ times LP solving. Each Split invokes at most $|B|^2$ testing of CanSim where B is a block in Σ. Each iteration of GCPP splits all current blocks, and the total number of comparisons within each iteration of GCPP is be bounded by $|S|^2$. (However it seems heuristics on the existing partition can achieve a speed close to linear in practice by caching previous CanSim checks [27].) The number of iterations is bounded by $|S|$. This gives us time complexity which is in the worst case to solve $\mathcal{O}(|Act_{\mathrm{I}}|\cdot|S|^3)$ many such LP problems, each of which has $\mathcal{O}(|S|\cdot|Act|+|Act|^2)$ constraints.

Remark. By removing the interaction between players (i.e., the alternating part), our algorithm downgrades to a partition-based algorithm computing the largest *strong* probabilistic simulation relation in probabilistic automata, where *combined transitions* are needed. The algorithm of [27] for computing strong probabilistic simulation has time complexity of solving $\mathcal{O}(|S|^2\cdot m)$ LP problems, where m is the size of the transition relation comparable to $\mathcal{O}(|S|^2\cdot|Act|)$. They have $\mathcal{O}(|S|^2)$ constraints for each LP instance. The improvement achieved in our algorithm is due to the use of partitions in each iteration instead of working on the whole relation, which is made possible by applying Lemma 3.

The space-efficient algorithm [26] for probabilistic simulation (*without* combined transitions) has the same space complexity but better time complexity than ours, which is due to the reduction to the maximal flow problem.

5 Conclusion

We have presented a partition-based algorithm to compute the largest probabilistic alternating simulation relation in finite probabilistic game structures. To the best of our knowledge, our work presents the first polynomial-time algorithm for computing a relation in probabilistic systems considering (concurrently) mixed choices from players. As aforementioned, PA-simulation is known as stronger than the simulation relation characterising quantitative μ-calculus [12], though it is still a conservative approximation which has a reasonable complexity to be useful in verification of game-based properties.

Acknowledgement. Wan Fokkink, Rob van Glabbeek and Lijun Zhang give us a lot of helpful comments. Especially we thank Timothy Bourke, whose comments have greatly improved the presentation of this paper.

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of ACM 49(5), 672–713 (2002)
2. Alur, R., Henzinger, T.A., Kupferman, O., Vardi, M.Y.: Alternating Refinement Relations. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 163–178. Springer, Heidelberg (1998)
3. Baier, C., Engelen, B., Majster-Cederbaum, M.E.: Deciding bisimilarity and similarity for probabilistic processes. Journal of Computer and System Sciences 60(1), 187–231 (2000)
4. Bustan, D., Grumberg, O.: Simulation based minimization. ACM Transactions on Computational Logic 4(2), 181–206 (2003)

5. Cattani, S., Segala, R.: Decision Algorithms for Probabilistic Bisimulation. In: Brim, L., Jančar, P., Křetínský, M., Kučera, A. (eds.) CONCUR 2002. LNCS, vol. 2421, pp. 371–386. Springer, Heidelberg (2002)
6. Chatterjee, K., de Alfaro, L., Henzinger, T.A.: The complexity of quantitative concurrent parity games. In: Proc. SODA, pp. 678–687. ACM (2006)
7. Chatterjee, K., de Alfaro, L., Majumdar, R., Raman, V.: Algorithms for game metrics (full version). Logical Methods in Computer Science 6(3:13), 1–27 (2010)
8. Clarke, E.M., Emerson, E.A.: Synthesis of Synchronization Skeletons for Branching-Time Temporal Logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982)
9. de Alfaro, L.: Quantitative Verification and Control Via the Mu-Calculus. In: Amadio, R.M., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 103–127. Springer, Heidelberg (2003)
10. de Alfaro, L., Henzinger, T.A., Kupferman, O.: Concurrent reachability games. In: Proc. FOCS, pp. 564–575. IEEE CS (1998)
11. de Alfaro, L., Majumdar, R.: Quantitative solution of omega-regular games. Journal of Computer and System Sciences 68(2), 374–397 (2004)
12. de Alfaro, L., Majumdar, R., Raman, V., Stoelinga, M.: Game refinement relations and metrics. Logic Methods in Computer Science 4(3:7), 1–28 (2008)
13. Gentilini, R., Piazza, C., Policriti, A.: From bisimulation to simulation: Coarsest partition problems. Journal of Automatic Reasoning 31(1), 73–103 (2003)
14. Grumberg, O., Long, D.: Model checking and modular verification. ACM Transactions on Programming Languages and Systems 16(3), 843–871 (1994)
15. Henzinger, M.R., Henzinger, T.A., Kopke, P.W.: Computing simulations on finite and infinite graphs. In: Proc. FOCS, pp. 453–462. IEEE CS (1995)
16. Jonsson, B., Larsen, K.G.: Specification and refinement of probabilistic processes. In: Proc. LICS, pp. 266–277. IEEE CS (1991)
17. Karmakar, N.: A new polynomial-time algorithm for linear programming. Combinatorica 4(4), 373–395 (1984)
18. Ranzato, F., Tapparo, F.: A new efficient simulation equivalence algorithm. In: Proc. LICS, pp. 171–180. IEEE CS (2007)
19. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. PhD thesis, Massachusetts Institute of Technology (1995)
20. Segala, R., Lynch, N.A.: Probabilistic simulations for probabilistic processes. Nordic Journal of Computing 2(2), 250–273 (1995)
21. Shapley, L.S.: Stochastic games. Proc. National Academy of Science 39, 1095–1100 (1953)
22. Tan, L., Cleaveland, R.: Simulation Revisited. In: Margaria, T., Yi, W. (eds.) TACAS 2001. LNCS, vol. 2031, pp. 480–495. Springer, Heidelberg (2001)
23. van Glabbeek, R.J., Ploeger, B.: Correcting a Space-Efficient Simulation Algorithm. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 517–529. Springer, Heidelberg (2008)
24. von Neumann, J., Morgenstern, O.: Theory of Games and Economic Behavior. Princeton University Press (1947)
25. Zhang, C., Pang, J.: On Probabilistic Alternating Simulations. In: Calude, C.S., Sassone, V. (eds.) TCS 2010. IFIP AICT, vol. 323, pp. 71–85. Springer, Heidelberg (2010)
26. Zhang, L.: A Space-Efficient Probabilistic Simulation Algorithm. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 248–263. Springer, Heidelberg (2008)
27. Zhang, L., Hermanns, H., Eisenbrand, F., Jansen, D.N.: Flow faster: Efficient decision algorithms for probabilistic simulations. Logical Methods in Computer Science 4(4:6), 1–43 (2008)

Towards a Smart, Self-scaling Cooperative Web Cache

Tomáš Černý[1], Petr Praus[2], Slávka Jaroměřská[2],
Luboš Matl[1], and Michael J. Donahoo[2]

[1] Department of Computer Science and Engineering,
Czech Technical University, Charles Square 13, 121 35 Prague 2, CZ
{tomas.cerny,matllubo}@fel.cvut.cz
[2] Department of Computer Science,
Baylor University, P.O. Box 97356, 76798-7356 Waco, TX, USA
{petr_praus,slavka_jaromerska,jeff_donahoo}@baylor.edu

Abstract. The traditional client/server architecture for web service delivery fails to naturally scale. This results in growing costs to the service provider for powerful hardware or extensive use of Content Distribution Networks. A P2P overlay network provides inherent scalability with multiple benefits to both clients and servers. In this paper, we provide analysis, design and prototype implementation of Cooperative Web Cache, which allows us to scale web service delivery and cope with demand spikes by employing clients in content replication. To demonstrate performance capabilities, we provide a prototype emulation for both client and server.

1 Introduction

Expectations on web service delivery rapidly grow every year. What we experienced in the past in a standalone, rich user interface application can be seen today online and on demand. As this trend continues, we expect online version of many applications such as office tools, remote desktops, user-friendly presentations, etc. This direction requires a network that can scale naturally and can provide soon feedback on client gestures and navigation.

The contemporary model used for web services follows the client-server architecture. This model works well until we reach hardware or bandwidth bottlenecks. In order to provide better scalability we must invest in powerful server hardware, load balancing, CDN services [1], and/or application rewrite to allow Cloud [2]. CDN distributes resources to multiple places around the world and allows the end-user to use the closest host for communication. Cloud computing allows to scale the service quickly. As the demand grows, the service might be replicated to multiple computers and handle larger load. Unfortunately the cloud involves extra charges for such ability.

In this paper, we argue that there exists a more natural way to deal with growing interest in a service. From the CDN and the cloud approaches, we see three aspects that should be addressed. First, the service should be as close as

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 443–455, 2012.

possible to the client. Second, if the interest grows, there should exist multiple replicas of such service. Finally, the existing service should not experience any changes and should not be recompiled. One possible approach to address the above mentioned aspects is to involve decentralized model to communicate with services. Peer-to-peer (P2P) overlay network has qualities that may satisfy our needs. Such network can be built by involved clients communicating with each other. This way the client requesting a service may cache the service results (similar as CDN) and those results might be provided to others. Such an overlay network may grow large, in that case we want two clients to have the best conditions for their communication, such as the best bandwidth or the lowest latency. Two distinct communication types exists, such as the client-to-server and client-to-client. The network itself should be self-scalable and resilient to client departure. Similar to clouds the client provides hardware resources as contribution to others. Such mechanism does not require any changes neither to the service nor to the client. Our prototype Cooperative Web Cache (CWC) provides the above mentioned functionality and properties. It allows us to evaluate real-world case study and to identify advantages and disadvantages to both client and server sides, and to identify challenges and further research.

This paper is organized as follow: As next analysis and design details are provided (Section 2). Case study and evaluation is in Section 3. Related work is discussed in Section 4. Paper is concluded in Section 5.

2 Analysis and Design of the CWC

Client-server model puts the entire burden on the service provider, who needs to invest into new hardware or CDN. CWC relocates most of the burden to clients who reuse the service results and share it with others. As popularity of a service grows more clients are participating in serving and in result the service scales more naturally with the size of clients. Furthermore, the distribution of participating clients may form clusters, so that a client in a given cluster should communicate within the cluster. The aim of CWC is to decrease server load and to speedup service delivery. The original service should stay unchanged and the whole process should be transparent to both sides. It can be seen as HTTP web browser extension or a proxy.

The CWC should be based on peer collaboration and expect various peer capabilities. CWC overlay network must sustain clients arrivals and departures and must be self-organizable. When clouds of clients with certain service results exist the requesting client should be directed to the nearest client. The nearest client in the overlay represents a location with the highest communication bandwidth and the lowest latency. In order to select the serving client an anycast request should be sent to the network. Although anycast is not supported by all networks the application level anycast can be used.

Regards web applications the service is to provide a page that consists of various resources (html, css, js, etc.). These resources can be divided on static and dynamic. Static resources often distributed on CDN are browser cacheble

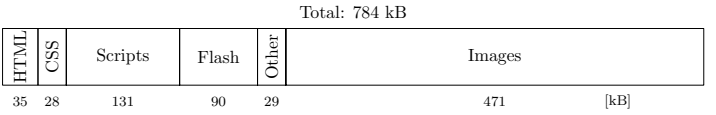


Fig. 1. Average web page content

and can be reused over the time (until changed by the host). Dynamic resources change often based on context, GET or POST request parameters [3] or user rights. Dynamic resources cannot be cached by browser and the replication would require a deeper knowledge of the system, although technologies such as AJAX [4] or AHAH (Asynchronous HTML and HTTP) can be seen as a successful approach with incremental changes to a temporarily cached dynamic resource. From the above, only static service results can be replicated. In order to impact the overall page load time by our approach the page must have static resources. Evaluation of Alexa Top 500 web sites [5] shows that 90% of web page resources are static in average case. This is in line with Google statistics as shown in Fig. 1.

The life-cycle of expected behavior shown in Fig. 2 is described from the client perspective. A client browser sends a request to a server (a1,b1,c1). The request is inspected to determine whether it provides a cacheable result (CWC). If not, the request is sent to the server (c2), otherwise the anycast request for the service result is sent to CWC cloud (a2,b2). If the result was not found (a3) the request is forwarded (a4) and fetched from the server (a5), otherwise is fetched from the cloud (b3). In both cases the client registers for the CWC group (a6) of clients having the result. In case the group does not exist then a new one is created (a6). The CWC group allows other clients to request the service result.

The life-cycle can be summarize as follows:

1. Client sends a request to a server
2. The request is inspected for the result
   (a) [Not cacheable] Request forwarded to the Server ⇒ step 3.
   (b) [Cacheable] Anycast for service result to CWC cloud
      i. [Result found] fetch from CWC Peer
      ii. [Result no found] fetch from Server
      iii. Join (create) CWC Group with service result
3. Done

The *basic* life-cycle above can be further extended with multiple strategies. A web page that consists multiple sources (service results) is most likely either cached in the network with all its sources or none of those will be registered in the network. We an approach based on this assumption *cache miss avoidance*. This way the CWC proxy can predict whether to send a request to the CWC cloud or directly to the server. As next, an index of expected service results can be built per a web page (as in [6]) and registered in the CWC cloud, this way the

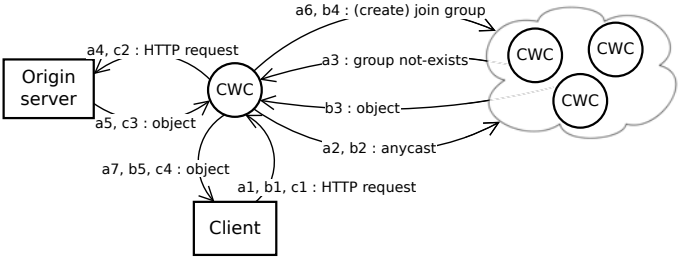


Fig. 2. Cooperative Web Cache request life-cycle

CWC proxy knows what services to call before the first service is requested and result parsed. This approach is referred as speculative *preloading*. Furthermore, a similar principle as in Bittorent [7] can be applied. The large service result can be divided into multiple chunks and requested from different locations in parallel, referred as *chunking*.

In order to verify validity of service result received from the CWC network multiple approaches can be applied. From the most optimistic approach a simple time-stamp will exist, once the result is stale based on the timeout the whole service result group is invalidated and must be rebuilt. The most pessimist approach is to always verify the result hash towards the server (as in HTTP), in case of failure the result group is invalidated. A reasonable approach is to employ probability where the client verifies the service result with probability p. In case of failure the group is invalidated and service requested from the server. The setting of p then reflects the level of optimism placed on the cache.

The CWC design [8] must be flexible towards modification and extension. For this reason it should be build on multiple layers to support maintenance and allow plugins for extension. The top layer should communicate with web browser as a proxy or web browser plugin. The bottom layer should contain a cache which can be either local or distributed, furthermore there can exist an addition extension or new services available on CWC. The middle layer should provide lookup services and mediate communication between the layers.

CWC prototype implementation consists from multiple parts. The distributed cache is build on P2P Framework developed at Rice University called Pastry [9]. This frameworks provides self-scalable P2P overlay network with operations like join, create or query. Its network is robust towards often joining and departing nodes and is tree-based towards its groups. A Tree allows to invalidated the whole group by contacting the root. Pastry itself does not provide neither anycast nor multicast capability necessary for optimal client-to-client communication. Pastry extension, Scribe [10] [11], provides both services in the application layer and guarantees that a message is delivered with logarithmic complexity regards the size of participants. The prototype is implemented in Java technology.

3 Case Study

In our case study we look at both client and server perspectives. In order to provide real-world study we must know capabilities of a server and a client that should be emulated. It is necessary to know the upload and download bandwidth capacity and latency of both.

From our measurement [5] for client-to-server, the median time for establishing a TCP connection (round-trip time) with the Alexa Top 500 websites through a backbone connection located in Prague, Czech Republic was 114 milliseconds. The latency for client-to-client is chosen from the above measurement to be 20 milliseconds. The presumption is that most traffic will be between geographically clustered nodes that are selected by anycast with distance metrics. End-user speeds vary significantly based on locations of such measurements or type of connectivity. The value for the U.S. household is based on Speed Matters report [12] and presented in Table 1. Average bandwidth across all providers in Prague and bandwidth average of state-wide provider UPC are taken from speed measuring website rychlost.cz on January 24th, 2011. Average bandwidth values for servers are much higher, commercial offers in data centers provide 100/100 Mbit/s for upload and download.

Our testbed environment [13] is based on a central unit directing distributed emulators via a test script. Each emulator node can be distributed on multiple machines where multiple nodes can be hosted by the same machine as well. Each emulated node has a specific network setting via latency and bandwidth [14].

3.1 Client Evaluation in Homogeneous Network

The evaluation in a homogeneous network applies identical client communication bandwidth and latency between all clients. The scenario is that a group of clients has the intention to download a web page. The page has 100 resources with consistence provided in Fig. 1. The first client requests and downloads the page, once done, a second client requests and downloads the page and so on. The client-to-client communication has latency 20 ms and bandwidth 2.2/9.3 Mbps for upload/download, which reflects the network properties in Prague. Server bandwidth capability is 100/100 Mbps for upload/download and the client-to-server latency is 114 ms.

Table 1. Typical download/upload bandwidth condditions

Type	Download [Mbps]	Upload	#Tests
Server	100	100	–
USA	3	0.6	–
Sweden	22.2	4.5	–
Japan	18.0	7.0	–
Prague, CZ	9.3	2.2	6467
UPC ISP, CZ	10.5	1.3	13195

Table 2. Massive download from server

Number of clients	1	20	30	40
Download time [ms]	6311	6860	6939	7297

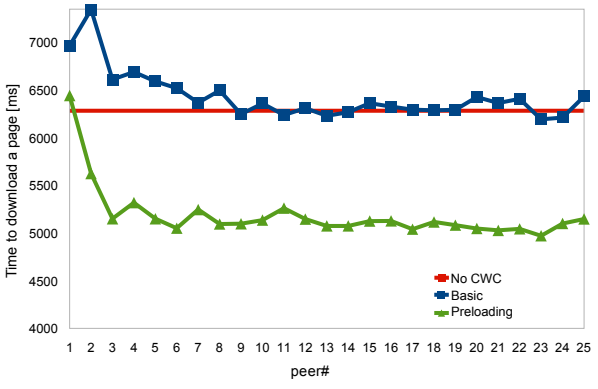


Fig. 3. CWC strategy evaluation

We measure the time it takes to a client to download the web page. An average download time of a single client downloading from the server with *no CWC* is **6.3** seconds (10 measurements). The time gets worse when multiple client load from the server as shown in Table 2.

The *basic*, *cache miss avoidance* and *preloading* CWC strategies described in Section 2 are compared with standard communication (*no CWC*). The results from the measurement are available in Fig. 3. Axis x,y of graphs represent the number of a client/peer and a total page download time respectively. For each measurement the first client requesting the page was fully served by the server, it is because the page resources were not stored in the CWC network yet. An average time for downloading a page from a server for the first client of the *basic* strategy is **6.9** seconds. The time difference of 0.6 seconds is caused by searches for files not available in the CWC network. When the *cache miss avoidance* strategy is employed, the average download time for the first client drops to **6.4** seconds, representing 7.3% improvement. The *preloading* strategy achieves similar results for the first peer. When a second client requests the page, the resources are fetched from both server and the first client. The page load time for *basic* strategy has worse load time, which is caused by lower first client upload compared to the server. Increasing the number of clients in the same CWC group allows a client to request service results from multiple locations, which positively impacts the overall load time. The saturation of the load time is in the range of 6-8 clients. In such case the load time drops to **6.3** seconds for both *basic* and *cache miss avoidance* strategies, respective **5.2** seconds for *preloading* strategy. An inspection of client participation shows that clients are

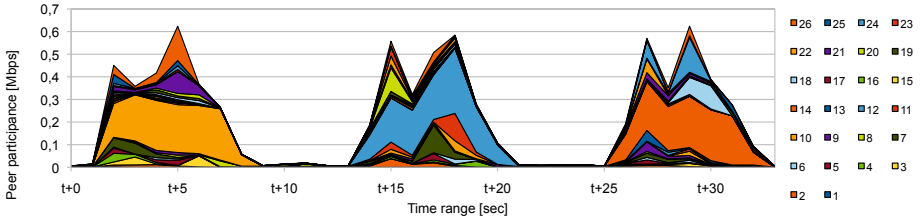


Fig. 4. Example peer participation in upload

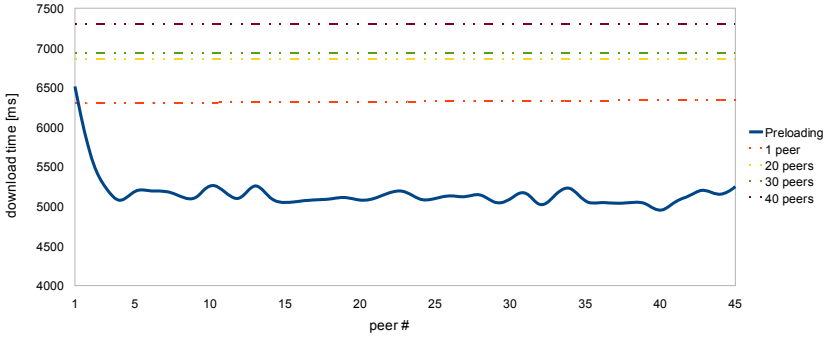


Fig. 5. Massive server download - compared to CWC with preloading strategy

not using the whole download capacity for *basic* strategy. However, when using *preloading* the download capacity was brought close to the limit distributed to all group members, although, no fairness mechanism was applied. In Fig. 4 is shown an example client participation in upload to a given client. The figure shows three consequent web page downloads in a CWC network consisting of 26 CWC clients with shared cacheable web page resources.

We also evaluate multiple clients downloading the same web page from a server at the same time. Fig. 5 shows that the *preloading* strategy is significantly faster than standard server download (no CWC).

3.2 Client Evaluation in Heterogeneous Network

This evaluation considers clients with different network capabilities. The scenario is the same as in the homogeneous evaluation. Clients has network conditions as in Prague, UPC, and Sweden listed in Table 1. The client-to-client latencies for all locations are specified in Table 3. Initial web page load times for evaluated locations (without CWC) are shown also in Table 3. Our evaluation with CWC network provides results in Fig. 6. The first requesting client is from Sweden and its load time is **4.3** seconds. Later load times for Swedish clients converge to **2.9** seconds. Prague load times converge to **5.6** seconds. Compare to homogeneous

Table 3. Location settings for peers

Location	Download	Upload	Latency [ms]				Web page load time
	[Mbps]	[Mbps]	Sweden	Prague UPC	Prague	Server	[ms]
Sweden	22200	4500	20	40	30	114	4303
Prague UPC	10500	1300	40	60	50	134	9497
Prague	9300	2200	30	50	40	124	6311

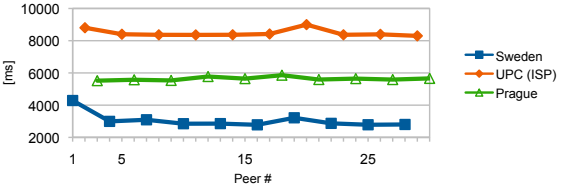


Fig. 6. Progressive heterogenous environment

network the result reflects increased delay that aggregates with number of resources. Prague UPC load times converge to **8.4** seconds.

3.3 Server Load Evaluation

From the server point of view the load can be seen by the number of hits or by downloaded data over a duration of the test. We compare requests going straight to the server with the use of CWC and without. Fig. 7 shows data downloaded from the server during the test. The green curve suggests that once the data is stored in the cooperative web cache the server load declines by 90%. This figure coincides with the ratio of dynamic content on a web page discussed above. Fig. 8 showing the number of requests to the server supports this idea. It should be noted the amount of downloaded data during the first five seconds roughly corresponds to the size of the page and its resources. After the first client downloaded some resources he provides them to other requesting clients.

3.4 Server Delay Dependency

Motivation for this test is discovering the boundary of server distance with which it is advantageous (performance-wise) to use CWC. For each bandwidth setup from Table 1 we compared download times of a client downloading directly from the server and a CWC client downloading from a network of 10 CWC clients. Fig. 9 shows that for Prague-like bandwidth it is useful to download page cached in a CWC network from server-to-client distances greater than 170 milliseconds. The same applies to UPC ISP and USA bandwidths with RTT 250 ms and

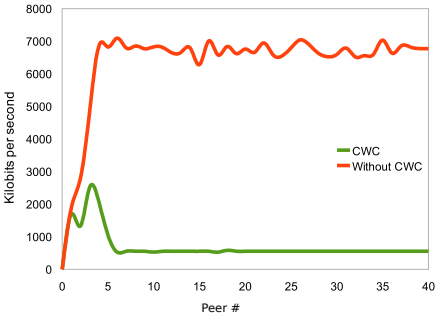


Fig. 7. Server load - data flow

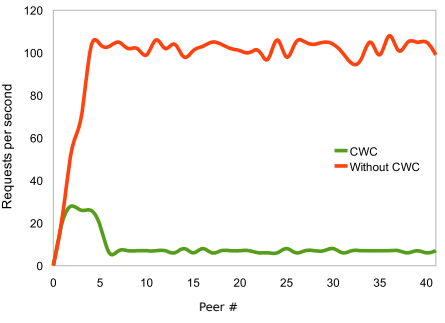


Fig. 8. Server load - requests

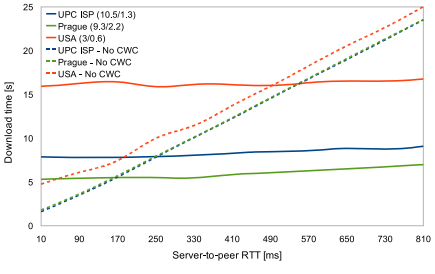


Fig. 9. Server delay - dependency between client-to-server RTT and speed

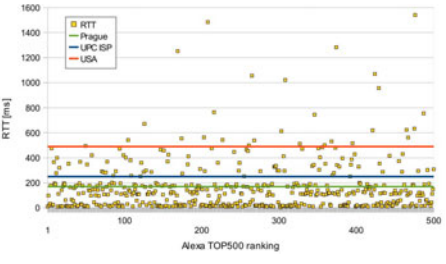


Fig. 10. Websites with improvable RTT values by different CWC setups

490 ms respectively. Fig. 10 shows RTT values from Alexa TOP500 mentioned above. The three horizontal lines represent efficiency boundaries for different setups. Access to websites above the lines could be improved by using a CWC network. The percentage of improvable websites by Prague, UPC ISP and USA bandwidth setups is 34%, 22% and 6.4% respectively.

3.5 Resilience to Abrupt Departure

Simulated departure of 5% and 10% clients from a network of 50 clients with all static sources in their caches was made. All peers left at the very same moment (no TCP FIN packet has been sent). An instant after the departure a new client joined the CWC network and downloaded cached page. The measured page download time is show in Table 4 (ten measurements). The referential download times were measured in a healthy CWC network with the same number of clients - for 5% and 10% departure 48 and 45 clients respectively.

Table 4. Resilience to abrupt departure of clients

	Network	Time [ms]	Deviation [ms]
5% DEPARTURE	Healthy	5124	96
	After departure	5435	349
10% DEPARTURE	Healthy	5109	97
	After departure	5523	414

4 Related Work

Survey of web caching [15] (section 4.1.2) contains early (1999) thoughts how a distributed caching of web content might look like and focuses on exchanging content between institution and national level caches. Distributed web caches are topics of multiple researches, for example Yingwu [16] shows their potentials and benefits. Squirrel [17] tries to engage clients in web content distribution and also builds on Pastry. It is similar to our proposal with multiple aspects, however, it aims only for corporate LAN networks and lacks anycast and multicast capabilities. Squirrel node, hosting a popular file, is bound to be eventually overloaded because all traffic for that given object is routed through a "home node". This also applies to its "directory mode" where the home node keeps reference to a certain number of it's latest clients and redirects new clients' requests to them just holding off the inevitable overload with redirect responses. Scribe [10], [11] implements anycast and multicast on top of Pastry [9] and allows us to implement efficient invalidation/update mechanism (multicast) and better scaling of popular files using the anycast. Dalesa [18] aims to provide web content caching for LAN networks with a working prototype. Kache [19] is focuses on lookup performance, its authors present a method reducing the number of necessary hops between nodes to one. They do this by using $O(\sqrt{n})$ space on each node (where n is the number of nodes) and a probabilistic approach to content retrieval. A node has a certain (very low) probability it will not be able to fulfill a request for the content it previously advertised.

P2P networks have unique security considerations, because peers are more powerful than clients. They issue their own node IDs, act as routers, relay messages and issues with trust arise. Uniform random distribution of node IDs is fundamental operational presumption of Pastry and all similar networks. If an attacker can *choose node ID*, she can surround a victim node, isolating it from the rest of the network, partition the network into smaller pieces or become a root key holder. This issue could be solved by centralizing node ID issuing into the hands of trusted certificate authority. Legitimate peers would refuse to communicate with peer not able to produce a signed combination of node ID and timestamp. A malicious node might also tamper with relayed messages, posing as the closest node. Furthermore one client can act with many identities [20]. Castro et al. describe possible solutions [21], but this area remains largely unexplored and addressing these concerns will be crucial for a practical use.

Multiple applications and research topics in this area are distributed web caches/proxies that specialize solely on content caching and its redistribution known as Content distribution networks (CDN's). Survey on CDN's and its optimization proposals are provided in detail by [22] or [23]. Among the most known solutions are Akamai [1] or Squid [24]. Squid is a hierarchical web proxy cache consisting of multiple independent servers. The content is shared by simple multicast query mechanism. These CDN solutions are statically distributed based on service provider decision. The motivation behind our approach is natural cache population formed by clients and its relocation based on popularity. Our approach does not involve costs related with cache server maintenance (Squid) or costs for given service (Akamai). On the other hand this approach does not face neither security issues nor content invalidation.

Cloud Computing [2] supports application scaling, it provides the illusion of infinite computing resources available on demand, eliminates an up-front commitment by cloud users and has the ability to pay for use of computing resources on a short-term basis as needed. The disadvantage comes with the implementation and extended costs. Our approach pushes the responsibility of the server-cloud to a user-cloud with no charge or changes in the implementation and could complement the cloud-computing in order to decrease charges for service.

5 Conclusion

In this paper we suggest that Internet web services could take advantage of peer-to-peer overlay network that is formed by clients with similar interest in the given time frame. We have presented our prototype Cooperative Web Cache and its simulations which are relatively close to the real-world conditions and prove our concept. CWC provides benefits to end-users who may experience improved download times for relatively common cases, benefits to service providers involve significant savings and better capability to survive peak loads. Although the beforementioned results are promising, we are aware that CWC concept is missing some key properties, such as proper security and content invalidation, which are subjects to further research. The invalidation scheme may impact performance as well as security. Security for decentralized approaches is more complicated and may require addition centralized mechanism issuing certificates or Facebook-like friendships. In order to provide chunking strategy for CWC the underlying network must provide manycast service.

Future work includes extensive emulation environment. We were able to emulate a network of at most about a fifty nodes with our hardware. Using more computers (for example on PlanetLab) would better reflect viability of our proposal in practical terms. Introducing jitter or packet-loss into our environment with heterogeneous network would provide more real-world performance results. Efficiency of CWC relies on a large user population and therefore it should be implemented in the most user-friendly way as a web browser plug-in.

References

1. Nygren, E., Sitaraman, R.K., Sun, J.: The akamai network: a platform for high-performance internet applications. SIGOPS Oper. Syst. Rev. 44, 2–19 (2010)
2. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Zaharia, M.: Above the clouds: A berkeley view of cloud computing. Technical report. Berkeley (2010)
3. Stevens, W.R.: TCP/IP illustrated: the protocols, vol. 1. Addison-Wesley Longman Publishing Co., Inc., Boston (1993)
4. Ullman, C., Dykes, L.: Beginning Ajax. Wrox (2007)
5. Cerny, T., Jaromerska, S., Praus, P., Matl, L., Donahoo, J.: Cooperative web cache. In: 18th International Conference on Systems, Signals and Image Processing, pp. 85–88. IEEE (2011)
6. Swen, B.: Outline of initial design of the structured hypertext transfer protocol. J. Comput. Sci. Technol. 18, 287–298 (2003)
7. Cohen, B.: Incentives Build Robustness in BitTorrent (2003)
8. Matl, L.: System for source distribution to support web application load time (cz). Master's thesis. Czech Technical University (2011), https://dip.felk.cvut.cz/browse/pdfcache/matllubo_2011bach.pdf
9. Rowstron, A., Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: Liu, H. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)
10. Druschel, P., Kermarrec, A.-M., Rowstron, A.: Scalable Application-Level Anycast for Highly Dynamic Groups. In: Stiller, B., Carle, G., Karsten, M., Reichl, P. (eds.) NGC 2003 and ICQT 2003. LNCS, vol. 2816, pp. 47–57. Springer, Heidelberg (2003)
11. Castro, M., Druschel, P., Kermarrec, A., Rowstron, A.: SCRIBE: A large-scale and decentralized application-level multicast infrastructure. IEEE Journal on Selected Areas in Communications 20, 1489–1499 (2002)
12. Speed matters, internet speeds report (2010), http://www.speedmatters.org/2010report
13. Jaromerska, S.: Environment for peer-to-peer application simulation with application on cooperative web cache. Master's thesis. Czech Technical University (2011), https://dip.felk.cvut.cz/browse/pdfcache/jaromsla_2011bach.pdf
14. Praus, P.: Framework for network management to support simulation of varying network conditions. Master's thesis. Czech Technical University (2011), https://dip.felk.cvut.cz/browse/pdfcache/prauspet_2011bach.pdf
15. Wang, J.: A survey of web caching schemes for the internet. ACM SIGCOMM Computer Communication Review 29, 36–46 (1999)
16. Zhu, Y.: Exploiting client caches: An approach to building large web caches. In: Proceedings of the 2003 International Conference on Parallel Processing, ICPP 2003 (2002)
17. Iyer, S., Rowstron, A., Druschel, P.: Squirrel: A decentralized peer-to-peer web cache. In: Proceedings of the Twenty-First Annual Symposium on Principles of Distributed Computing, pp. 213–222. ACM (2002)
18. Dalesa: The Peer-to-Peer Web Cache, http://www.dalesa.lk/
19. Linga, P., Gupta, I., Birman, K.: Kache: Peer-to-Peer Web Caching Using Kelips (2004)

20. Douceur, J.R.: The Sybil Attack. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 251–260. Springer, Heidelberg (2002)
21. Castro, M., Druschel, P., Ganesh, A., Rowstron, A., Wallach, D.: Secure routing for structured peer-to-peer overlay networks. ACM SIGOPS Operating Systems Review 36, 299–314 (2002)
22. Ball, N., Pietzuch, P.: Distributed content delivery using load-aware network coordinates. In: Proceedings of the 2008 ACM CoNEXT Conference, CoNEXT 2008, pp. 77:1–77:6. ACM, New York (2008)
23. Bakiras, S., Loukopoulos, T., Papadias, D., Ahmad, I.: Adaptive schemes for distributed web caching. J. Parallel Distrib. Comput. 65, 1483–1496 (2005)
24. Spare, I.: Deploying the squid proxy server on linux. Linux J. (2001)

Named Entity Disambiguation Based on Explicit Semantics

Martin Jačala and Jozef Tvarožek

Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology, Bratislava, Slovakia
martin.jacala@gmail.com,
jtvarozek@fiit.stuba.sk

Abstract. In our work we present an approach to the Named Entity Disambiguation based on semantic similarity measure. We employ existing explicit semantics present in datasets such as Wikipedia to construct a disambiguation dictionary and vector–based word model. The analysed documents are transformed into semantic vectors using explicit semantic analysis. The relatedness is computed as cosine similarity between the vectors. The experimental evaluation shows that the proposed approach outperforms traditional approaches such as latent semantic analysis.

1 Introduction

The constantly increasing amount of written text available online makes the automatic processing of these resources a challenging task. The published textual resources (e.g. personal blogs, newswire articles, web pages) provide us with highly unstructured yet interesting source of knowledge. To make the automatic processing possible, we need to solve many problems along the way. The field of Natural Language Processing studies and tries to solve such challenges.

The named entities, such as persons names, locations, names of organisations and similar are widely used in any form of written text. Many of the widely used words to describe entities are subject to homonymy. For example, if we find a word "Jaguar" in the text, we often need to disambiguate the true meaning of the particular statement among the set of all possible meanings (sports car, animal, etc.). The understanding of the true meaning of the word is crucial in many situations.

It has been shown that NEs carry much of the user interest making it an source of information in user modelling and personalisation approaches [15]. Recent studies show that at least 70% of submitted search queries contain reference to named entity [7,16] and true understanding of document content can be helpful in search engine optimisation.

In our work we present an approach to the Named Entity Disambiguation problem based on the explicit semantics extracted from Wikipedia data. The presented paper is organised as follows: in the following chapter we discuss the

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 456–466, 2012.

related work, the third chapter presents the proposed method, in the fourth chapter we present the evaluation and the attained results and finally we conclude and summarise the contribution. Approaches to disambiguate search by recommending keywords according to users social network [10], and ad-hoc navigation using automatic term retrieval, ranking and categorization [21] were proposed.

2 Related Work

The identification of named entities in text is very important and well studied task of the NLP since its introduction at MUC6 conference [6]. The goal of the identification is to label all surface forms used to describe an entity in the written text with an appropriate label, such as person, organisation, etc. While labelling the entities, we may encounter several types of disambiguation problems. While we solve the structural ambiguity we need to break the surface forms found in the text correctly into entities (e.g. how many entities does *The Metropolitan Museum of Art in New York* refer to?). The second problem is the identification of the proper type of an entity (e.g. *Jordan advanced* is a person or location?).

2.1 Semantic Ambiguity

If we process more text resources there is high probability that the same entities are present in many of them. Due to the semantic ambiguity and homonymy, however, the surface forms used in these resources may differ significantly. For example, the entity *International Business Machines Corporation* is often referred using the acronym *IBM* or its nick name *Big Blue*. Grouping all occurrences of the same entity regardless to the surface form used in the text is an challenging task providing vital information about the true semantic content of the resources contained in the data set. On the other hand, there are entities with the same surface forms but with different meanings (e.g. *John Smith*).

This problem has been originally solved with model based on heuristic measured incorporated into the Nominator system [22] originally developed as a named entity recogniser. The method uses the University of Pennsylvania's CAMP coreference detection system further extended with the Vector Space Model to solve homonymous cases. During the evaluation they compare the impact of used scoring function and achieved an F1 score (harmonic mean of precision and recall) of 87% [1].

Similar approach based on the context in which entities appear is presented in [19,9]. The approach employs within-document coreference analysis followed by context analysis when necessary. During this stage, each entity is transformed into an canonical structure used later during the cross–document analysis. The canonical structures are then merged or split into two or more separate entities according to many heuristics and linguistic rules. Comparison of the context is done using the Context Thesaurus tool, which returns a list of related terms to any given input text. The comparison of term sets for analysed entities provide additional information used in the coreference analysis.

2.2 Large–Scale Disambiguation

The clustering approaches presented in the previous section detect and track an entity even across multiple documents but are usually unable to uniquely identify the meaning using an external dictionary. Several approaches using large scale data such as Wikipedia or Open Directory has been recently proposed to overcome this limitation.

The concept of semantic relatedness between two text fragments is the fundamental idea of the majority of latest approaches to the named entity disambiguation. One of the first approaches to compute semantic relatedness using Wikipedia data is WikiRelate [20]. The evaluation shows that Wikipedia significantly outperforms traditionally used Wordnet on larger datasets.

The approach described in [3] presents a baseline method based on dictionary constructed from Wikipedia image. The dictionary contains a list of surface forms extracted using heuristic analysis of letter capitalisation in Wikipedia articles together with *context* for each surface form. The context is captured with a sliding window of 50 words around each occurrence of hypertext link. The name of linked article is then used to label and uniquely identify the disambiguated meaning. The same technique is used to analyse the input document, the fragments are then transformed into a set of tf-idf weight vectors and compared using the cosine similarity measure. The baseline method is then further extended with support vector machile (SVM) disambiguation kernel trained on the data extracted from Wikipedia image. Experimental evaluation on selected Wikipedia articles show the precision of 82.3% and 84.4% for the cosine measure and the taxonomy kernel, respectively.

The approach presented in [4] present hybrid approach consisting of separate entity recognition and disambiguation stages. The disambiguation process employs a vector space model to compare both vector representations of analysed document and information extracted from Wikipedia. The document vector contains aggregated categories and contexts of the entities discovered in the article. This vector is then compared with each possible entity vector, maximising the similarity between vectors. The evaluation done on set of manually annotated newswire articles gave the precision of 91.4%.

There are many other uses of knowledge extracted from Wikipedia, such as mining a domain–specific thesauri, training named entity recognisers or machine translations between multiple languages [14,18]. The Wikipedia–based corpora has been proven more suitable than the ones created from Open Directory Project or even gold–standard corpora for applications in the NLP [5,17]. Automatic term recognition algorithm used for keyword extraction from Wikipedia pages taking advantage of HTML tags was proposed [12]. Keyword relations can be extracted using different large scale data, such as social bookmarks from delicious and CiteULike [2].

2.3 Formal Definition

We may cast the entity disambiguation as a ranking problem: we search for the most related meaning (concept) c of an surface form of an entity s among all

of the possible meanings $c \in C$, where C is set of all possible meanings in the knowledge base extracted during training. We define an ranking function

$$sim = \arg\max_{c} \ rank(s, c)$$

and compute the similarity (sim) between the surface form found in the text and the possible meanings.

3 Entity Disambiguation Using Wikipedia

In this section we present an overview of the proposed method. We divided the disambiguation process into four main stages: (1) Entity identification and boundary detection, (2) Transformation of the document, (3) Lookup of the candidate meanings and finally (4) Ranking and disambiguation. The input to this process is plain text without any formatting or markup, each later stage process the output from it's predecessor.

As the source of background knowledge for the disambiguation we pre–process Wikipedia image and construct a disambiguation dictionary and vectorial word model (semantic space) for later use in the mentioned process. In the following parts of this chapter we discuss the individual steps in detail.

3.1 Dataset Structure

The main idea of our contribution is to leverage the explicit semantics already present in Wikipedia. This source of rich markup created by the human editors, such as hyperlinks, redirect or disambiguation pages is very useful for the purpose of named entity disambiguation.

In our work, we create a *disambiguation dictionary* where each surface form of an entity we find in Wikipedia image has a set of related *candidate meanings*. The extraction follows a simple idea that each hypertext written in the MediaWiki syntax in the form [*Surface Form*|*Article Name*] provides a mapping between the surface form commonly used in the text to describe the entity to it's proper name. For instance, considering the following text fragment we extract the mapping of Big Blue as a surface form for the entity IBM: *The software giant, also known as* [[*Big Blue*|*IBM*]]*, released the much anticipated product.*

Similarly, we create such mapping for any redirect page we find in the dataset. Usually, the redirect pages are used to correct misspelling or to create mapping of an informal to the official name. We then further extract all possible meanings of a proper name through the disambiguation pages found in Wikipedia. The disambiguation page is a page created when a proper name has many different, unrelated meanings. These pages usually form a long list of hypertext links all lexicologically related to the proper name. We extract these entities as the values in the disambiguation dictionary, so the dictionary contains a set of the possible, candidate meanings for each surface form. Each meaning is identified by the name of appropriate Wikipedia article as shown on Fig. 1.

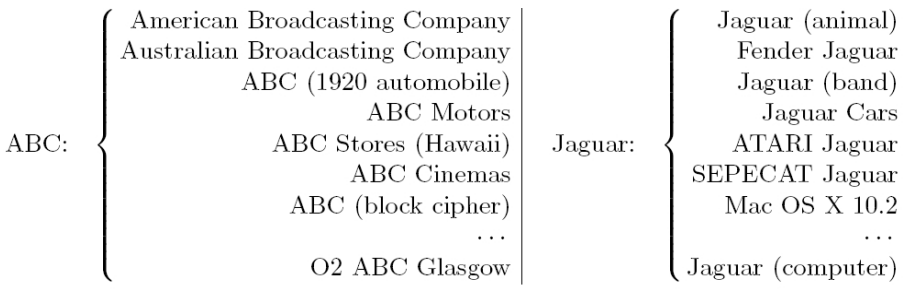


Fig. 1. Example of the constructed disambiguation dictionary for surface forms ABC and Jaguar

3.2 Semantic Space

The Wikipedia is not only a source of rich semantic and a network of interlinked documents, but a source of vast amount of text as well. We use the information extracted from the way the words are used in different contexts to discover the most probable meanings.

According to the Charles and Miller [13] co-occurence hypothesis – similar, or the same words are used in semantically similar contexts. Following this hypothesis, we should be able to discover the related contexts of a document or text fragment by analysing the word usage across the dataset.

This word usage information is used later in the text processing and disambiguation stage of the proposed method. To discover and capture the word – concept relationships, we construct a *semantic space* – a vector word model created from large amount of training text data.

Traditionally, latent semantic analysis (LSA) is used to create such vectorial models. The LSA method does not need any kind of structurality in the input data [11]. Using matrix operations such as singular value decomposition reduces the number of dimensions in the space leaving out smaller and merging adjacent concepts. Finally, the method discovers hidden (latent) concepts in the provided training text. The inverted document frequency (tf-idf) is computed for each word in relation to the discovered latent concepts.

If we assume that each of Wikipedia's articles describes one *natural* concept, we may leave out the detection of latent concepts. Each of the articles in the dataset is well defined, categorised and written using consistent writing style. If we consider the natural concepts as the concepts in the constructed semantic space, we may leave out the discovery process of latent semantic analysis.

Method described in [5] uses semantic relatedness to generate topics or concepts from any given fragment of text. The method is referred to as explicit semantic analysis – each dimension of the semantic space corresponds to one natural concept defined in the training data set.

We build the vectorial semantic space using the explicit semantic analysis from Wikipedia data. The process is relatively straightforward, we count the occurrences of words in the individual articles and the whole dataset, then we compute term and inverse document frequencies. The completed semantic space is an term – concept matrix, where columns corresponds to the natural concepts and individual rows are words found in the dataset. The matrix values are tf–idf frequencies of the words. Thus, each row vector gives us an idea of how related is the given word to each of the generated concepts.

3.3 Entity Identification

Our proposed method does not attempt to solve the problem of identification of named entities in the text. Instead, we use existing named entity recogniser system (Stanford NER). The recogniser offers state–of–art precision and recall on the ConLL 2003 dataset. The proposed method is not tied with this specific system, this step can be left out if the analysed text has already defined entity boundaries or if the text will be annotated manually.

Currently, we only use the information about entity boundaries ignoring any additional information provided (e.g. entity type) making it even easier and more flexible to adopt for different environments.

3.4 Document Transformation

The analysed document is transformed into a semantic vector using the term – concept matrix we described earlier. The semantic vector is in the following form.

$$Q = (M_1, M_2, M_3, \cdots, M_n)$$

where n is the number of dimensions in the semantic space. Each of the vector values $M_1, \cdots, M_n$ is the relatedness of the document to corresponding concept, in our case this is equal to Wikipedia article used in the training stage.

The vector is created as follows. Firstly, we apply simple stemmer to reduce the number of unique words in the document. Then, we look up appropriate row vector from the term – concept matrix for the words found in the document. The final semantic vector is a running total of the selected rows. The semantic space is a sparse matrix, so we need to add only the non–zero tf-idf values. The final vector is used later when we compute the semantic relatedness.

3.5 Candidate Meanings

As we mentioned earlier, the disambiguation dictionary contains a set of {surface form–possible meanings} pairs extracted from Wikipedia. In this stage of the disambiguation process we use the dictionary to narrow down the set of possible meanings.

For each surface form retrieved from the NER system we query the dictionary for the possible meanings according to extracted explicit semantics. There are three possible outcomes. First, there is exactly one possible meaning for the current surface form. This means we found unambiguous entity or an entity with minor other meanings. If the latter is the case, we are currently unable to detect and disambiguate the out–of–Wikipedia entities. If an unambiguous entity is found, we label the occurrence with the name of appropriate article in Wikipedia and the processing ends here.

Second, the most common case is when we match a set of possible meanings for analysed surface form. In this case, we retrieve the text of the articles corresponding to the candidate meanings and pass them to the next stage.

Third, we may have found no candidates for given surface form. Either we found an out-of-Wikipedia entity or the entity has been misspelled, or expressed in very rare way (it has been not expressed in that way in Wikipedia). In this case, we run approximate string matching algorithm to overcome some of the differences in exact spelling. All matching surface forms are then processed as in the second case.

The disambiguation dictionary is crucial to the disambiguation process not just because of narrowing down the number of possible candidates to reduce workload. During this process we also eliminate false meanings that may be very related to the topic of the document fragment, but not being an exact meaning of the surface form. We demonstrate the impact of the disambiguation dictionary on the following text fragment.

[I]f successful, the changes could get incorporated into future Mars missions. Spirit and Opportunity were also fitted with a new navigation system that allows them to think several steps ahead when faced with an obstacle [...].

It is clear that surface form *Spirit* and *Opportunity* are names of the Mars remote operated vehicles. The Tab. 1 shows the highest ranked meanings for these surface forms. In the first column are the highest ranked meanings among all of Wikipedia articles. The retrieved concepts are related to the analysed text, however, it does not relate to the given surface forms. If we limit the set of meanings only to those retrieved from the dictionary (columns 2 and 3) we find that the true meanings for the given surface form (in bold) rank higher than other meanings from the dictionary.

Table 1. Top ranking meanings for a given text fragment

Unrestricted	*Spirit* surface form	*Opportunity* surface form
Navigation	**Spirit rover**	**Opportunity rover**
Offensive tackle	37452 Spirit	39382 Opportunity
Geology	B-2 Spirit	Launch window
Inland navigation	The Spirit	Opportunity cost
Robot	Holy Spirit	Business opportunity
Planetary geology	Spirit (band)	Equal opportunity

3.6 Ranking

In the previous sections we discussed key components of the disambiguation process. In this final stage we process each surface form with at least two candidate meanings retrieved in the third step. Firstly, we transform the articles that describe each possible meaning into the semantic vectors. Then, we compute the semantic relatedness between the document vector and each vector created from appropriate Wikipedia article.

According to the formal definition mentioned earlier, we define the ranking function as cosine between the document vector Q and candidate meaning vectors $M_1, \ldots, M_n$.

$$rank(q, m) = \cos(\theta) = \frac{Q \cdot M}{\|Q\|\|M\|} = \frac{\sum_{i=1}^{n} Q_i \times M_i}{\sqrt{\sum_{i=1}^{n} (Q_i)^2} \times \sqrt{\sum_{i=1}^{n} (M_i)^2}}$$

Finally, we take the most similar meaning and label the occurrence in the input document. This approach, however, fails to rank some of the correct meanings as the most related because of several reasons, e.g. too short input document, too short or incomplete Wikipedia article or too few references to the entity found in the document. For instance, in an article about the band Texas surface forms are correctly mapped to the *Texas (band)* article, however, if another ambiguous entity with one of the meaning related to the music is present in the text, the method may incorrectly prefer this alternative over the correct, intended meaning.

To partially solve this problem, we compare modified vectors and combine both results of ranking functions. The modified vectors are constructed as vectors of words found in the sliding window around the surface forms found in the documents. Using this technique, we partially eliminate strong incline e.g. to music related topics in case there is no evidence of being related to music in close vicinity of the surface form in question. The impact of this enhancement is further discussed in the section Evaluation.

4 Evaluation

We implemented the proposed method using Java programming language and the s–space framework [8]. Three evaluation dataset were used, one during the development and the other two during the evaluation run only. Each dataset contain 20 manually disambiguated newswire articles with marked entity boundaries. We discarded references to entities not found in Wikipedia. We found that approximately 10% of all surface forms has been discarded due to out-of-wikipedia entities. The named entities in our datasets have 18 meanings on average, while 78% of entities have more than one meaning extracted from Wikipedia.

In the evaluation we compared the results of human annotator with the results of our automated method. Further, we studied the impact of two-vector approach where we create two semantic vectors for each document fragment, this modification is referred to as *Combined* (Tab. 2).

Table 2. Precision – Proposed method

Dataset	ESA - Article		ESA - Combined	
	Acc [%]	σ	Acc [%]	σ
devel	89.31	9.93	**91.93**	10.33
eval-1	87.84	7.61	**90.25**	6.67
eval-2	85.06	8.93	**86.56**	8.56

We have found that the *Combined* approach does improve the precision in some edge cases, especially the cases when the surface form in question does refer to a topic significantly different than the rest of the article.

Further, we modified our implementation to follow the latent semantic analysis approach and limited the number of dimensions to n=250. We tried to further increase and decrease the number of dimensions with no significant effect on the precision of the method. The results of the LSA approach are shown in Table 3.

Table 3. Precision – LSA model

Dataset	LSA - Article		LSA - Combined	
	Acc [%]	σ	Acc [%]	σ
devel	80.60	16.60	81.41	16.47
eval-1	80.34	15.44	81.05	16.35
eval-2	80.63	15.06	82.33	11.65

The baseline method presented in [3] does not use the semantic space model at all. We compared the results of both ESA and LSA models to this approach. We did not used the *Combined* variant here, as we followed the original description during our implementation.

The sliding window method often fails to capture the important pieces of knowledge because of the way typical article is written – the article is often divided into sub-topics (sections) to improve readability. Mostly, if a Wikipedia article consists of many paragraphs of text, the hypertext links does not appear through the entire article but only couple of times in the first paragraphs of an article. The results of baseline evaluation are shown in Table 4.

Table 4. Precision – Baseline method

Dataset	Baseline	
	Acc [%]	σ
devel	76.28	17.30
eval-1	85.36	10.44
eval-2	82.36	14.94

5 Conclusion and Future Work

In our paper we present an approach to automated named entity disambiguation based on Wikipedia data. We use explicit semantics already defined in Wikipedia to retrieve all possible disambiguations for the entities. Additionally, we leverage the semantics to create a high-dimensional word model to compute the similarity between the documents based on human created concepts defined in the Encyclopaedia. Our method uses an existing named entity recogniser as preprocessor, therefore no human annotation of unknown text is necessary.

The evaluation of our method shows better results than the traditionally used Latent Semantic Analysis as well as the baseline system. As the future work we plan to extend the method with a contextual classifier. Such classifier will take into account already disambiguated entities in the document as opposed to individual disambiguation of the entities. Such contextual awareness can be of great help to resolve cases when an entity has been successfully disambiguated earlier in the text (expressed with different surface form) but fails to rank appropriately later on.

Additionally, we experiment with various scenarios of the context generation to solve the outstanding issue when the occurrences of the same surface form in the document have two distinct meanings. Currently, all such occurrences are merged into one, most probable meaning.

Acknowledgement. This work was partially supported by the grants VG1/0971/11/2011-2014, KEGA 028-025STU-4/2010, APVV-0208-10 and it is the partial result of the Research & Development Operational Programme for the project Research of methods for acquisition, analysis and personalized conveying of information and knowledge, ITMS 26240220039, co-funded by the ERDF.

References

1. Bagga, A., Baldwin, B.: Entity-based cross-document coreferencing using the Vector Space Model. In: Proceedings of the 36th Annual Meeting on Association for Computational Linguistics, pp. 79–85. Association for Computational Linguistics, Morristown (1998)
2. Barla, M., Bieliková, M.: On Deriving Tagsonomies: Keyword Relations Coming from Crowd. In: Nguyen, N.T., Kowalczyk, R., Chen, S.-M. (eds.) ICCCI 2009. LNCS, vol. 5796, pp. 309–320. Springer, Heidelberg (2009)
3. Bunescu, R., Pasca, M.: Using encyclopedic knowledge for named entity disambiguation. In: Proceedings of EACL, pp. 9–16 (2006)
4. Cucerzan, S.: Large-scale named entity disambiguation based on Wikipedia data. In: Proceedings of EMNLP-CoNLL, vol. (6), pp. 708–716 (2007)
5. Gabrilovich, E., Markovitch, S.: Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, pp. 1606–1611 (2007)
6. Grishman, R., Sundheim, B.: Message understanding conference-6: A brief history. In: Proceedings of the 16th Conference on Computational Linguistics (COLING 1996), Copenhagen, Denmark, vol. 1, pp. 466–471 (1996)

7. Guo, J., Xu, G., Cheng, X., Li, H.: Named entity recognition in query. In: Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR 2009, p. 267. ACM Press, New York (2009)
8. Jurgens, D., Stevens, K.: The S-Space Package: An Open Source Package for Word Space Models. In: Proceedings of the ACL 2010 System Demonstrations, pp. 30–35. Association for Computational Linguistics (2010)
9. Kazi, Z., Ravin, Y.: Whos who? Identifying concepts and entities across multiple documents. In: Proceedings of the 33rd Annual Hawaii International Conference, p. 7. IEEE Computer Society (2000)
10. Kramár, T., Barla, M., Bieliková, M.: Disambiguating Search by Leveraging a Social Context Based on the Stream of User's Activity. In: De Bra, P., Kobsa, A., Chin, D. (eds.) UMAP 2010. LNCS, vol. 6075, pp. 387–392. Springer, Heidelberg (2010)
11. Landauer, T.K., Foltz, P.W.: An Introduction to Latent Semantic Analysis. Discourse Processes 1(25), 259–284 (1998)
12. Lučanský, M., Šimko, M., Bieliková, M.: Enhancing automatic term recognition algorithms with HTML tags processing. In: Rachev, B., Smrikarov, A. (eds.) Proceedings of the 12th Int. Conf. on Computer Systems and Technologies (CompSysTech 2011), pp. 173–178. ACM, New York (2011)
13. Miller, G.A., Charles, W.G.: Contextual correlates of semantic similarity. Language and Cognitive Processes 6(1), 1–28 (1991)
14. Milne, D., Medelyan, O., Witten, I.: Mining domain-specific thesauri from wikipedia: A case study. In: IEEE/WIC/ACM International Conference on Web Intelligence, pp. 442–448. IEEE Computer Society, Hong Kong (2006)
15. Min, J., Jones, G.J.F.: Building User Interest Profiles from Wikipedia Clusters Categories and Subject Descriptors. In: Proceedings of the Workshop on Enriching Information Retrieval (2011)
16. Nguyen, D., Overwijk, A., Hauff, C., Trieschnigg, D.R.B., Hiemstra, D., de Jong, F.: WikiTranslate: Query Translation for Cross-Lingual Information Retrieval Using Only Wikipedia. In: Peters, C., Deselaers, T., Ferro, N., Gonzalo, J., Jones, G.J.F., Kurimo, M., Mandl, T., Peñas, A., Petras, V. (eds.) CLEF 2008. LNCS, vol. 5706, pp. 58–65. Springer, Heidelberg (2009)
17. Nothman, J., Murphy, T., Curran, J.: Analysing Wikipedia and gold-standard corpora for NER training. In: Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics, pp. 612–620. Association for Computational Linguistics (April 2009)
18. Ponzetto, S.P., Strube, M.: Knowledge derived from Wikipedia for computing semantic relatedness. Journal of Artificial Intelligence Research 30(1), 181–212 (2007)
19. Ravin, Y., Kazi, Z.: Is Hillary Rodham Clinton the president?: disambiguating names across documents. In: Proceedings of the Workshop on Coreference and its Applications, pp. 9–16. Association for Computational Linguistics, Maryland (1999)
20. Strube, M., Ponzetto, S.: WikiRelate! Computing semantic relatedness using Wikipedia. In: Proceedings of the National Conference on Artificial Intelligence, vol. 21, p. 1419. AAAI Press, MIT Press, Menlo Park, Cambridge (1999) (2006)
21. Ševce, O., Tvarožek, J., Bieliková, M.: Term Ranking and Categorization for Ad-Hoc Navigation. In: Dicheva, D., Dochev, D. (eds.) AIMSA 2010. LNCS, vol. 6304, pp. 71–80. Springer, Heidelberg (2010)
22. Wacholder, N., Ravin, Y., Choi, M.: Disambiguation of proper names in text. In: Proceedings of the Fifth Conference on Applied Natural Language Processing, pp. 202–208. Association for Computational Linguistics, Stroudsburg (1997)

Design Pattern Support Based on the Source Code Annotations and Feature Models

Peter Kajsa and Pavol Návrat

Faculty of Informatics and Information Technologies, Slovak University of Technology,
Ilkovičova 3, 842 16 Bratislava, Slovakia
{kajsa,navrat}@fiit.stuba.sk

Abstract. Nowadays there exist many approaches to support design pattern instantiation, evolution, etc., but most of the approaches focus mainly at the design level (i.e. model). By the transition to the source code level the pattern instances become almost invisible in the huge amount of source code lines. The goal of this paper is to present our method supporting design pattern instantiation, evolution and identification in the source code. The method is based on source code annotations and feature models of individual patterns. The method does not require a manual annotation of the source code by a human, instead the method works on the idea of architectural and design information propagation and expansion from higher levels of abstraction (i.e. models) into the source code. In this paper we also present a method defining how to connect the necessary knowledge to the model and the source code.

Keywords: design patterns, instantiation, evolution, identification, source code, context assistant, annotations, feature models.

1 Introduction

In general, patterns are based on abstraction and generalization of effective, reliable and robust solutions to recurring problems. The idea of applying verified pattern solutions to common recurring problems has very quickly attracted considerable attention also in the software design (e.g. [1]), since the quality of software systems depends greatly on the design solutions chosen by developers.

Nowadays there are many efforts to improve the quality of software system development or maintenance based on the identification, acquisition and application of some kind of design or architectural knowledge (e.g. [2]). Most of the approaches are focused on the support of design patterns at the design level (i.e. model). Many of them are based on pattern modeling languages, pattern ontologies (e.g. [4]), UML profiles (e.g. [5]), pattern templates (e.g. [6]) or model transformations (e.g. [7]), etc.

As a result, developers have available wide tool-based support of many pattern aspects at the design level. However, the support becomes less robust by the transition to the source code and pattern instances become almost invisible in the huge amount of the source code lines. The evolution of the existing instances of patterns in the source code is very difficult without any tool-based support, because a developer does

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 467–478, 2012.

not have a good view of all the participants of pattern instances in the source code. Moreover, due to the inability to identify the individual participants of pattern instances in the source code, they may be modified in an incorrect way during the system evolution and maintenance, and this may result in the breakdown of the pattern and the loss of the benefits gained by its application in the software system.

Consequently, it is necessary to mark explicitly and make visible the higher-level (i.e. design) intentions in the source code. However, a pure source code does not provide explicit expression of its semantics, because this knowledge is mainly available to developers and domain experts involved in the design process. Despite these difficulties, the missing semantics can be added into the source code via annotations (e.g. [8]) expressing the semantics and intention of the annotated code fragments.

2 Related Work

Meffert [11] introduces an approach assisting developers in selection of the correct design pattern for a given context. The approach introduces the annotations to the source code in order to express an intention of the given source code fragment. Meffert also proposes the description of the intention for some design patterns. The suitable pattern is recommended to a developer on the basis of comparison of the annotated source code intention with the intention defined for the design pattern's parts.

Sabo et al. [10] present a method of preserving the correct form of applied design patterns during the process of software system evolution. The method aims to explicit indication of the pattern participants in the source code by annotations. The authors also propose a mechanism determining whether the applied pattern instances are still valid or have been broken due meantime code modifications.

Kirasić et al. [12] present an ontology-based architecture for pattern recognition. The authors integrate the knowledge representation ground and static code analysis for pattern recognition.

Another method of the patterns recovery based on code annotations and regular expressions has been introduced by Rasool et al. [9]. The authors extend the list of annotations defined in [11] in order to detect the similarity of different annotations used in multiple patterns. Authors' intention is to use the annotations for the static analysis of the source code and subsequent recognition of structural design patterns.

Fülleborn et al. [3] present an approach of the documentation of the particular source code or UML models that have design deficiencies, in order to document the problems in their context that the chosen design pattern solves. Documenting is done by adding non-functional requirements in form of annotations. Next, the authors formally document also the solved problems so that they can be compared to the situation before the chosen design pattern was applied. By the way of comparison, the transformation between the situation before and after the application of the design pattern is made explicitly in order to derive the reusable cross-domain representation of the situation.

3 Open Problems and Our Ideas

Source code annotations as metadata information about the intention of the source code fragments have been introduced in several approaches. All of the approaches, however, require a manual annotation of source code fragments by a developer. So a developer needs to insert the annotations manually in the source code as a guideline before the source code analysis (e.g. [9], [10]) or to annotate the intentions of the source code fragments, manually as well. Only after that an approach is capable of the pattern recommendation (e.g. [11]).

However, the manual annotation of the source code by developers is very lengthy and tiresome. Therefore our method does not require the manual annotation of the source code by a human, instead the method works on the idea of the architectural and design information propagation and expansion from higher levels of abstraction (i.e. models) into the source code.

Moreover, the other approaches do not try to support the instantiation or evolutions of the applied patterns via annotations in the source code, even though some try to recover patterns from the source code (e.g. [9]), to indicate the errors in the pattern structure during the compilation (e.g. [10]) or to perform a recommendation of a suitable pattern for application (e.g. [11]).

Our idea is to support also the instantiation and evolution of the pattern instances in the form of the context assistant in the source code working via annotations and feature models.

4 Method of Design Pattern Support in the Source Code

The semantics of patterns introduced into the source code by annotations emphasizes the visibility of pattern instances and therefore makes identification of pattern participants in the source code quite easy. In consequence, the support of the pattern detection, instantiation and evolution in the source code can be achieved in a very suitable form of a source code context assistant. Thanks to annotations, the support mechanism will be able to identify the pattern participants already implemented, and subsequently it will be able to offer an option to generate any missing pattern participant or to perform possible pattern evolution in the given context, etc. This idea brings significant improvement of the pattern support at this level of abstraction (i.e. the source code).

4.1 Proposal of Annotation for Design Patterns

Source code annotations work as metadata information for different artifacts and fragments of the source code. This information can be processed by various tools (compilers, etc.). We propose the following definition of annotation for design patterns (see Figure 1).

```
public @interface DesignPattern{
    PatterNames patterName();
    String instanceAlias();
    RoleNames roleName();
    String variant() default "DEFAULT";

    enum PatterNames{ Observer; //...
    }
    enum RoleNames{Subject, attach, detach, notifyObservers,
        update, observers, Observer, ConcreteObserver; //...
    }
```

Fig. 1. Proposal of the source code annotation for design patterns

The attribute `patternName` of the annotation expresses the name of the pattern e.g. `Observer`, `Mediator`, `Command`, etc. Because one pattern (for example Observer) may have more different instances applied, the pattern instance "alias" is necessary for the recognition among these instances. The `roleName` expresses the name of the pattern participant e.g. `Subject`, `ConcreteSubject`, `attach`, etc. Some participants of the pattern instances may have more possible variants and therefore the `variant` attribute is also necessary.

In the following Figure 2 an example of the source code snippet with some annotated participants of Observer pattern instance by the proposed definition of annotation is illustrated.

```
@DesignPattern(patterName = PatterNames.Observer, instanceAlias = "obs1",
        roleName = RoleNames.Subject)
public abstract class AbstractUserModel {
    // ...
    @DesignPattern(patterName = PatterNames.Observer, instanceAlias = "obs1",
            roleName = RoleNames.observers)
    ArrayList<View> views;
    // ...
    @DesignPattern(patterName = PatterNames.Observer, instanceAlias = "obs1",
            roleName = RoleNames.attach)
    public void addView(View v){
        views.add(v);
    }
    //...
    @DesignPattern(patterName = PatterNames.Observer, instanceAlias = "obs1",
            roleName = RoleNames.notifyObservers, variant="modelOfNotification = callback")
    public void updateViews(){
        for(View v : views) v.refresh(this);
    }
    //...
}
```

Fig. 2. Example of the source code snippet with some annotated participants of Observer pattern instance by proposed definition of annotation

4.2 Support of Design Pattern Instantiation and Evolution

The support of the pattern instantiation and evolution is realized in form of the source code context assistant with the consequent source code generation driven by typed annotation and its location. The method is described in the following steps.

1. In the first step, the developer begins with the writing of a pattern annotation in the desired location in the code. When the developer writes `@DesignPattern (patternName...`, the context assistant offers the set of names of supported patterns. The developer, for example, chooses `PatternNames.Observer`.
2. Next the developer continues with the writing of the annotation and writes `instanceAlias`. So the annotation looks as follows: `@DesignPattern (patternName = PatternNames.Observer, instanceAlias = ...` Now the context assistant searches all the existing instances of the pattern with the given name i.e. `PatternNames.Observer` and it offers the developer the set of aliases of all the existing instances of Observer pattern in the project. Because of the suitable annotation structure this search is very straightforward.

Consequently, the developer chooses an `instanceAlias` from the offered set or writes a new, unique alias. When the developer writes a new, unique instance alias, the support mechanism deduces that the developer desires a creation of a new pattern instance. Otherwise, when the developer chooses one of the offered existing instance aliases, the support mechanism deduces that the developer desires evolution of the pattern instance identified by the chosen alias and the pattern name. According to the developer's choice pattern instantiation (4.2.1.) or evolution (4.2.2) follows.

4.2.1 Design Pattern Instantiation

Because in the second step of the previous section the developer wrote a new, unique instance alias, the instantiation of the pattern with the typed name is performed (in our case instantiation of Observer). The method continues with the following steps.

1. The support mechanism loads feature model of the pattern. It selects all mandatory features at the first level (i.e. classes) and generates them into the source code.
2. If one of the mandatory features has more possible variants, the developer is asked for selection of its variant via dialogue during the instance generation.

Illustration of the feature model of Observer pattern is shown in following Figure 3.

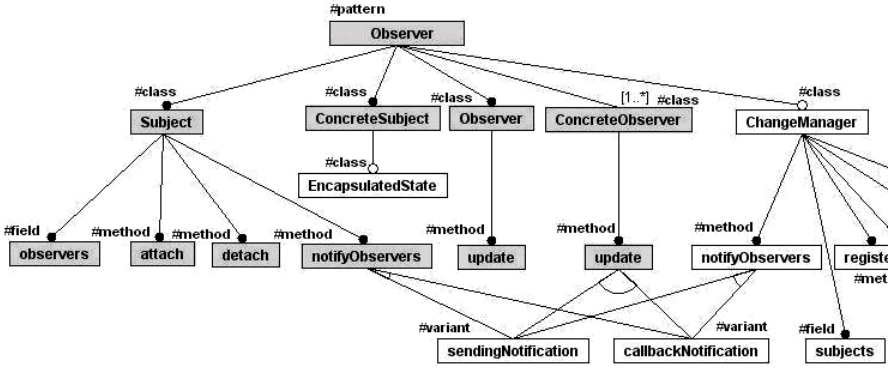


Fig. 3. Illustration of feature model of Observer. Mandatory features are filled with gray color.

The first mandatory class is generated at the position of the entered annotation in the current file, therefore in case of the pattern instantiation the developer should write the annotation in a new empty file. Other mandatory classes are generated into new automatically created empty files in the current package of the project. Of course, an element is always generated with all its mandatory sub-elements.

4.2.2 Pattern Evolution

When the developer selects alias from offered set of the existing instance aliases of the pattern with the name typed in the second step (section 4.2), the support mechanism deduces that the developer wants to perform the evolution of the pattern instance with the selected instance alias. The support continues with following steps.

1. The support mechanism creates a feature model configuration of the pattern instance identified by the selected alias. Thanks to the annotations, the recognition of the pattern instance participants present in the source code is quite easy.
2. The support mechanism loads the feature model of the pattern.
3. The created feature model configuration of the pattern instance is compared with the loaded feature model of the pattern. In consequence, the options of possible evolution of the pattern instance are detected (Figure 4).
4. The support mechanism offers the detected set of possible instance evolution options in form of the code assistant (Figure 5). So the developer may choose the desired pattern instance evolution.

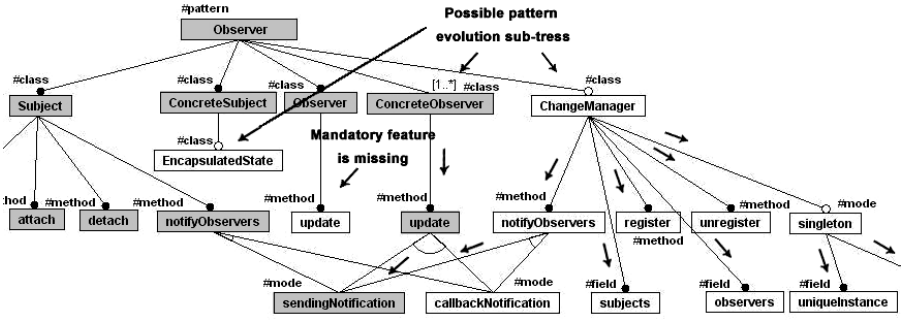


Fig. 4. A comparison of the feature model configuration of an existing Observer instance with the feature model of Observer pattern (existing participants - features are filled with gray color). The possible options of pattern instance evolution are illustrated by the arrows.

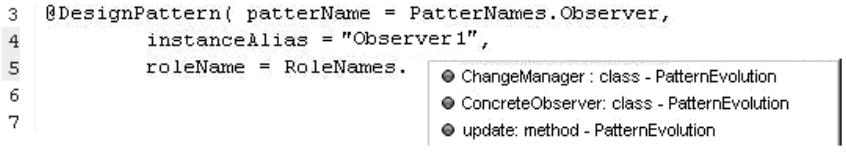


Fig. 5. Example of detected set of possible instance evolution options offered in form of the code assistant

It is important to remark that only the roots of possible instance evolution sub-trees are offered to the developer, because generation of child elements (e.g. methods) has no sense as long as the parent element (e.g. class) does not exist in the source code.

The selected element with all its mandatory sub-elements is generated at the position of the entered annotation in the current file. So the method supposes at least basic knowledge of patterns. If an element has more possible variants within the scope of the given instance, the developer is asked to select one of the variants via the dialogue during the element generation.

Within the scope of the pattern evolution also the detection of missing mandatory features is supported (e.g. Figure 4, the update method of Observer instance is missing). This way the basic check of the pattern instance validity is achieved.

4.3 Realization

Each element of the pattern feature model[1] has its own code template attached. Each code template of an element includes subsequent templates of all related mandatory sub-elements of the element in accord with the feature model of the pattern. Therefore an element is always generated with all its mandatory sub-elements. For example, Subject template includes `observers`, `attach`, `detach` and `notifyObservers` templates. Example of Subject template is illustrated in the following Figure 6.

```
<%@ jet class="Observer-SubjectTemplate" package="dp.anot.jet" startTag="<%" endTag="%>"
<% Map context = (Map)argument;
String instanceAlias = (String)context.get("instanceAlias");
String curr_package = (String)context.get("package");
//...
%>
package <%=curr_package%>;
import java.util.*;

@DesignPattern( patterName = PatterNames.Observer, instanceAlias =  <%=instanceAlias%>,
        roleName = RoleNames.Subject )
public class Subject<%=instanceAlias%> {

  /*included and generated subparticipants */
  <%= includeParticipant("observers") %>
  <%= includeParticipant("attach") %>
  //...
  <%= includeParticipant("notifyObservers") %>
}
```

Fig. 6. Example of Subject template. The template includes subsequent templates of sub-elements of Subject in accord to the feature model of Observer.

If an element has more possible variants, the template of such element contains the source code for both variants distinguished by annotations (for example, see Fig. 7). The following notation has been introduced for the variant attribute of annotations:

```
[~]Attribute_name = value[;]
```

If the attribute value selected by the developer in GUI dialogue corresponds with the introduced notation, the variant of an element is generated from the template.

[1] Except the elements marked as #pattern or #variant.

Dependency on more than one value or attribute can be attached via ";", while the symbol "~" expresses negation. So when the element - feature has more than one possible variant, the developer's selection is compared with annotations in the template and in consequence, the desired variant of element – feature is generated.

```
<%@ jet class="Observer-notifyObserversTemplate" package="dp.anot.jet" startTag="<%" endT
<% Map context = (Map)argument;
String instanceAlias = (String)context.get("instanceAlias");
String observersClassName = "";
if(instanceEvolution)
 observersClassName = getObserverClassName(PatterNames.Observer,
         instanceAlias,RoleNames.Observer);
if(observersClassName.equals(""))
 observersClassName = "Observer"+instanceAlias;
//...
%>
@DesignPattern( patterName = PatterNames.Observer, instanceAlias =  <%=instanceAlias%>,
        roleName = RoleNames.notifyObservers, variant = "modelOfNotification = callback")
    public void notifyObservers<%=instanceAlias%>() {
        for (<%=observersClassName%> o : <$=observersCollectionRefName>)
      o.<%=updateMethodName%>(this);
    }

@DesignPattern( patterName = PatterNames.Observer, instanceAlias =  <%=instanceAlias%>,
        roleName = RoleNames.notifyObservers, variant = "modelOfNotification = sending")
    public void notifyObservers<%=instanceAlias%>() {
        for (<%=observersClassName%> o : <$=observersCollectionRefName>)
      o.<%=updateMethodName%>(this.get<%=subjectStateClassName>();
    }
```

Fig. 7. Example of `notifyObservers` template which contains two different variants distinguished by annotations (notice difference of `variant` attributes of annotations)

As it can be noticed, in the Figure 6, the names of new generated classes, methods and fields are created as `roleName+InstanceAlias`. So the name collisions are minimized, because the concatenation of `roleName` and `instanceAlias` is unique in the scope of pattern instantiation and evolution as well. The developer may rename the elements later, of course. However, when a body of a method is generated in the scope of an instance evolution, the introduced name convention is not sufficient enough. The bodies of generated methods should be tied to an existing implementation of the instance and therefore the particular names of existing elements should be found out (for example, see `observerClassName` retrieving in the Figure 7). Because of the annotations of existing pattern participants this task is straightforward.

Moreover, the whole method is based on the defined name conventions. The names of feature models are identical to the `PatternNames` used in the source code annotations and the feature names are identical to the `RoleNames` used in the source code annotations as well. The templates are named as follows: `PatternName-RoleNameTemplate`. As a consequence, the support mechanism is able to automatically deduce from the annotations typed by the developer in the source code which feature model and which templates it should load and generate. This way the flexibility of the method is improved and achieved, since the addition of a new feature model and new templates is sufficient enough to extend the support for a new pattern. An extension of `PatternNames` and `RoleNames` about the new pattern name and roles is also necessary, of course.

4.3.1 Implementation

Implementation of the method is based on the Eclipse platform. The templates are implemented in JET framework. The JET framework is part of Eclipse Modeling Project in M2T (Model to Text) area and it provides very good infrastructure for the source code generation based on code templates.

The feature models of patterns are implemented as UML class diagrams analogically, as has been introduced in [13] (see the section Feature Modeling Profile for UML), but for the method purposes we rather use the class diagram instead of the component diagram.

However, because Java does not support the annotation of one code unit (i.e. method, class, etc.) by more than one annotation with the identical name, the current implementation is limited in case that one fragment - unit of the code represents more roles in more patterns (for example, in case of pattern composition). This problem can be resolved by enclosing the next `DesignPattern` annotations in comments.

The presented implementation is intended for Java platform, but it can be simply adjusted also for other platforms, even if they do not support source code annotations. In such case the annotations may be enclosed in comments as well.

4.4 Elimination of Manual Annotation of the Source Code

The instances of patterns applied into the source code via the presented method are automatically generated into the source code with the annotations of all instance participants. So the evolution of these instances is supported by the method directly, without any need of manual annotation.

For other instances of patterns we propose a mechanism of architectural and design information propagation and expansion from higher levels of abstraction (i.e. models) into the source code.

4.4.1 Realization

For the purpose of architectural and design decision suggestion in the model we use the UML profile which represents the built-in standard for semantic extension of UML. UML profile enables to define semantic extension of UML in form of semantic marks (i.e. stereotypes) and their meta-attributes (i.e. tagged values), enumerations and constraints. The elements from UML profile can be applied directly to the context (i.e. specific model elements such as Classes, Attributes, Operations, etc). Therefore, the UML profile provides the appropriate way to define required semantics for architectural and design decisions or patterns. As a consequence, the developer is able to suggest architectural or design decisions and intentions via application of the stereotypes from the UML profile onto specific model elements. For example, in the following Figure 8, we can see the suggestion of the Observer pattern via stereotype `<<Observes>>` application onto the association.

Suggested instances of patterns are concretized by the transformation of a model to a model in the next step. The transformation generates missing structural participants of pattern instances and it also marks each pattern participant with the appropriate stereotype. So the transformation propagates and expands applied marks (in our case

the two stereotypes <<Observes>>) on particular elements in the concretization process as the pattern instances are concretized. The result of the transformation of the model from Figure 8 is illustrated in the next Figure 9.

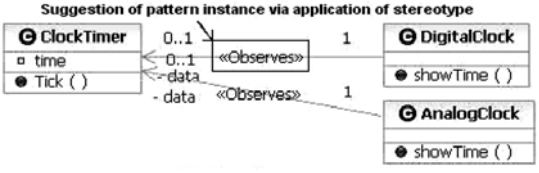


Fig. 8. Suggestion of the Observer pattern instances via stereotype <<Observes>> application

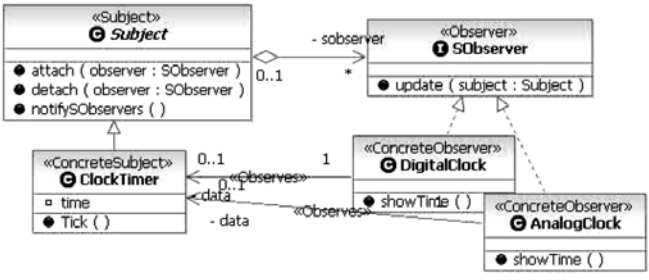


Fig. 9. The result of the transformation of the model from Figure 8. The transformation marks each pattern participant with an appropriate semantic mark (i.e. stereotype) from UML profile.

We have published the realization of the transformation and the UML profile in our previous work. For more details about the authored UML profile and realization of model to model transformations please see the paper [14].

The models with concretized instances of patterns are transformed into the source code in the next step. In order to propagate the visibility of the applied patterns from the model into the source code we have used proposed annotations (see Figure 1). In the following Figure 10 the source code snippet of Subject generated from the model in the Figure 9 is illustrated. Each generated pattern participant is annotated with the proposed definition of annotation. The transformation of the model into the source code is realized in form of JET templates which generate the pattern participants with correct annotations as well. For classes marked with a stereotype, the template with the same name is used. For example, for the model classes marked with the stereotype <<Subject>>, the template with the name `subject.javajet` is used, etc.

Consequently, in this approach we propagate and expand the two applied stereotypes from higher level of abstraction (i.e. <<Observes>> from Figure 8) onto many annotations in the source code (e.g. Figure 10 - but it is only a little snippet from one class of all generated source code). So this way, the massive manual annotation of pattern participants in a huge source code is reduced to a few manual suggestions via stereotypes at the highest level of abstraction (e.g. Figure 8).

```
@DesignPattern( patterName = PatterNames.Observer,
        instanceAlias = "obs1",
        roleName = RoleNames.Subject,
        variant = "encapsulateSubjectState = false; managerType = noManager")
public class Subject {

    @DesignPattern( patterName = PatterNames.Observer,
            instanceAlias = "obs1",
            roleName = RoleNames.ObserversCollection)
    ArrayList<SObserver> observers = new ArrayList<SObserver>();
    //...

    @DesignPattern( patterName = PatterNames.Observer,
                    instanceAlias = "obs1",
                    roleName = RoleNames.Attach)
    public void attach(SObserver observer){
        observers.add(observer);
    }
    //...
```

Fig. 10. Source code snippet of Subject class generated from the model illustrated in Figure 9

5 Evaluation

The presented method and the implemented tool have been evaluated in various ways. The evaluation in form of case studies has been performed, by which we have compared the results obtained by the tool with our targets. Furthermore, the method and the tool have been evaluated by experiments in which we have monitored the time of carrying out the assigned tasks with and without usage of the tool. The tasks consisted of implementing specified instances of Observer pattern in specified form. Average results of the experiments on a group of three programmers and three master degree students of software engineering are summarized in the Table 1. The results of the experiments indicate a significant improvement gained by the usage of the tool.

Table 1. Average results of the experiments of Observer pattern implementation with and without usage of the tool

Time with tool usage – t1	Time without tool usage - t2	Speed up - t2/t1	Number of generated code lines - Ng	Number of added code lines – Na	Improving coefficient - (Ng / Na) + 1
< 15 min	> 35 min	> 2,3	123	19	7,5

6 Conclusion and Future Work

The presented method fits in wider context of pattern support based on semantics and subsequent model transformations or source code generation. The proposed definition of annotations introduces the semantics and clear visibility of pattern instances in the source code and in consequence it opens new opportunities to support various aspects of patterns, or even for correct reverse transformations of the code with the pattern detection. Available feature models of patterns also enable a possibility of live detection of pattern instances advanced defects. Because the manual annotation of the source code by developers is very lengthy and tiresome, we have proposed the approach of manual annotation elimination based on the idea of design information

propagation and expansion from models into the source code. Although it does not deal with the problem of existing or legacy software systems, it provides a very useful way how to propagate and expand design information and how to prevent the problem of pattern instances invisibility in the source code towards the future. Besides, it does not have to be used only for patterns, but it can be simply adjusted for other architectural or design decisions as well.

Acknowledgments. This work was partially supported by the grants VG1/0508/09, APVV-0208-10 and it is the partial result of the Research & Development Operational Programme for the projects Support of Center of Excellence for Smart Technologies, Systems and Services SMART and SMART II, ITMS 26240120005 and ITMS 26240120029, co-funded by the ERDF.

References

1. Gamma, E., et al.: Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley professional computing series (1995)
2. Havlice, Z., et al.: Knowledge Based Software Engineering. In: Computer Science and Technology Research Survey, elfa, Košice, pp. 1–10 (2009)
3. Fülleborn, A., Meffert, K., Heisel, M.: Problem-Oriented Documentation of Design Patterns. In: Chechik, M., Wirsing, M. (eds.) FASE 2009. LNCS, vol. 5503, pp. 294–308. Springer, Heidelberg (2009)
4. Kampffmeyer, H., Zschaler, S.: Finding the Pattern You Need: The Design Pattern Intent Ontology. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 211–225. Springer, Heidelberg (2007)
5. Debnath, N.C., et al.: Defining Patterns Using UML Profiles. In: IEEE International Conference on Computer Systems and Applications, Washington, pp. 1147–1150 (2006)
6. Marko, V.: Template Based, Designer Driven Design Pattern Instantiation Support. In: Benczúr, A.A., Demetrovics, J., Gottlob, G. (eds.) ADBIS 2004. LNCS, vol. 3255, pp. 144–158. Springer, Heidelberg (2004)
7. Dong, J., Yang, S., Zhang, K.: A model transformation approach for design pattern evolutions. In: ECBS 2006, pp. 80–92. IEEE Computer Society, Washington, DC (2006)
8. Java Specification Request: Common Annotations for the Java Platform, `http://www.jcp.org/en/jsr/detail?id=250`
9. Rasool, G., Philippowa, I., Mädera, P.: Design pattern recovery based on annotations. Advances in Engineering Software, 519–526 (2010)
10. Sabo, M., Porubän, J.: Preserving Design Patterns using Source Code Annotations. Journal of Computer Science and Control Systems, 53–56 (2009)
11. Meffert, K.: Supporting Design Patterns with Annotations. In: Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based System, ECBS 2006, pp. 437–445. IEEE Computer Society, Washington, DC (2006)
12. Kirasić, D., Basch, D.: Ontology-Based Design Pattern Recognition. In: Lovrek, I., Howlett, R.J., Jain, L.C. (eds.) KES 2008, Part I. LNCS (LNAI), vol. 5177, pp. 384–393. Springer, Heidelberg (2008)
13. Vranić, V., Šnirc, J.: Integrating Feature Modeling into UML. In: NODe 2006, GSEM. Lecture Notes in Informatics, vol. P-88, pp. 3–15. Gesellschaft fur Informatik, Bonn (2006)
14. Kajsa, P., Majtás, L.: Design Patterns Instantiation Based on Semantics and Model Transformations. In: van Leeuwen, J., Muscholl, A., Peleg, D., Pokorný, J., Rumpe, B. (eds.) SOFSEM 2010. LNCS, vol. 5901, pp. 540–551. Springer, Heidelberg (2010)

On the Formalization of UML Activities for Component-Based Protocol Design Specifications

Prabhu Shankar Kaliappan and Hartmut König

Department of Computer Science
Brandenburg University of Technology Cottbus
P.O.B. 10 13 44, 03013 Cottbus, Germany
{psk,koenig}@informatik.tu-cottbus.de

Abstract. Formal description techniques, such as LOTOS and SDL, have been proven as a successful means for developing communication protocols and distributed systems. Meanwhile the Unified Modeling Language (UML) has achieved wide acceptance. It is, however, less applied in the field of protocol design due to the lack of an appropriate formal semantics. In this paper we propose a formalization technique for UML activity diagrams using the compositional Temporal Logic of Actions (cTLA). We use cTLA because it can express correctness properties in temporal logic and can also be verified formally using several model checking mechanisms. The approach consists of two steps. First, we predefine the formal semantics of the most commonly used UML activity nodes using simple cTLA. In the second step we derive the *functional semantics* of the activity diagram by mapping it to a compositional cTLA process. We illustrate our approach for a connection set up as an example. Finally we present with the *Activity to cTLA generator* a tool to automate this process.

Keywords: communication protocols, distributed systems, UML modeling, formal description techniques (FDTs), formal semantics, cTLA.

1 Motivation

The Unified Modeling Language (UML) provides a variety of diagrams to model the structure and behavior of a system to be designed. Compared to the formal description techniques (FDTs), such as the *Specification and Description Language* (SDL) [1] or the *Language of Temporal Ordering Specification* (LOTOS) [2] etc., the UML has been less applied to the design of distributed systems and communication protocols so far. This is due to the in-adequate option to specify a formal semantics. The FDTs enforce a strict formal semantics. They support straightforward mechanisms to verify designs and to perform model transformations. Many significant contributions on the FDT-based design and validation of communication protocols and distributed systems, respectively, have been published in the last two decades. Babich and Deotto give a survey on this in [3]. In contrast to FDTs, UML has found a wide acceptance in software design. There are mainly two reasons for this: (1) it uses profiles [4] to model domain specific concepts and (2) the UML specification document [5] does not

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 479–491, 2012.

provide a formal semantics. Instead, it allows the designer to define a semantics that is suitable for the specific application development. So several approaches have been proposed which formalize UML diagrams applying programming language semantics, such as C++, JAVA, etc. This sometimes directs the UML towards an implementation language rather than a modeling language which limits its system modeling capabilities and weakens the UML benefits, such as user-free design. Hence, there is a necessity to define implementation independent formal semantics. Besides, the semantics should support the verification of the UML models to prove the design correctness.

We have developed a component-oriented modeling approach for the design of communication protocols and distributed systems [6]. The approach aims at the reuse of components, represented as UML diagrams, in various designs. Components may be typical protocol mechanisms, such as handshake procedures, flow control, and others. Each component consists of two representations: a communication- and a behavior-oriented description. The *communication-oriented description* illustrates the interactions between the communicating entities. This presentation is to be used to visualize the protocol flow, i.e. how the entities interact and in which order. The communication-oriented description is preferred and applied in the design phase as well as for documentation purposes. UML sequence diagrams are applied for this. The *behavior-oriented description* describes the behavior of each communicating entity, e.g. as a finite state machine. It shows the execution flow of the entity when reacting to occurring events. Behavior-oriented description is typically represented in UML by activity diagrams. Accordingly, the communication- and behavior-oriented descriptions complement each other. They have to be synchronized [7] in the modeling process (see Figure 1).

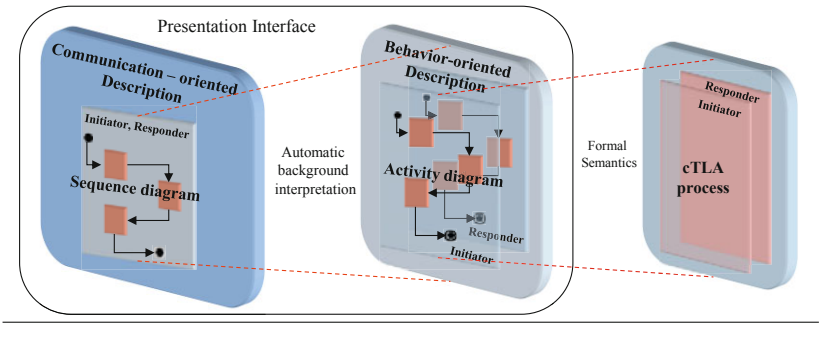


Fig. 1. Different system design perspectives

A UML-based design specification can only be used as a basis for further protocol or system development if it is defined in a formal way. Only this enables an exact interpretation of the design and allows model transformations into other formal presentations. By investigating the functions of the communication- and behavior-oriented description, we assign a formal semantics to the behavior description (see Figure 1). We chose the behavior-oriented description for this because it specifies how the occurring events are handled by the entities. This is the basis for subsequent

coding. Besides, the subsequent development stages verification and testing relate to the protocol design given as activity diagrams. We use the *compositional Temporal Logic of Actions* (cTLA) [8] for defining the formal semantics of UML activity diagrams. It is designed for the specification and verification of transfer protocols. Today it is applied for the design of reactive systems [9]. cTLA was chosen because it is a constraint-oriented specification language. It allows one to formulate desired properties of the systems under design in a temporal logic and to prove formally their correctness. In this paper we describe how UML activity diagrams can be mapped onto cTLA. For this, first we predefine the formal semantics of the most commonly used UML activity nodes. In the next step we show how the *functional semantics* of the activity diagram can be derived by combining the predefined semantics and the activity diagram data. The remainder of the paper is organized as follows. In Section 2 we briefly introduce cTLA. Thereafter in Section 3, we first define the semantics of the activity nodes. For the latter, we present a transformation method to map activity diagrams onto a compositional cTLA process. We also present the *Activity to cTLA process generator* tool to automate this process. Section 4 discusses related approaches. Some final remarks conclude the paper.

2 cTLA – Compositional Temporal Logic of Actions

cTLA is a formal specification language developed for the specification and verifycation of transfer protocols [8]. It is based on Lamport's Temporal Logic of Actions (TLA) [10]. cTLA distinguishes two types of processes: simple and compositional ones. Simple cTLA processes are used to model single system resources, while compositional ones model systems and sub-systems as compositions of simple cTLA processes which cooperate by means of synchronously executed process actions. An example is shown in Table 1.

Table 1. A simple and a compositional cTLA process

Simple cTLA Process	Compositional cTLA Process
PROCESS communicate(pdu_type: ANY) **CONSTANTS** FREE ∈ pdu_type **BODY** **VARIABLES** channel: pdu_type **INIT** ≜ channel = FREE **ACTIONS** send(sd: pdu_type) ≜ channel = FREE ∧ channel′ = sd; receive(rd: pdu_type) ≜ channel ≠ FREE ∧ channel = rd ∧ channel′ = FREE; **END**	**PROCESS** connect_s **IMPORT** DT-PDU **BODY** **VARIABLES** state: {"idle","wait connection","connected"}; **INIT** ≜ state = "idle"; **PROCESSES** C: communicate(pdu: pdu_type); t: Timer(to: natural); … **ACTIONS** Con-Init(pdu: pdu_type) ≜ pdu ∧ pdu.type = "DT" ∧ pdu.sequ = 1 ∧ state = "idle" ∧ state′= "wait connection" ∧ C.send(pdu) ∧ t.start(5); … **END**

cTLA specifications have a program-like structure with *process* as main function, *constants* and *variables* as declarations, *init* as the initial process state, *processes* as sub-functions with an index, and *actions* to define the process behavior. The process states are defined by the current variable values. The simple cTLA process shown in the example above describes a process communication to send and receive a message. It is modeled by a *channel* of *pdu_type*. Initially, the state variable is set to *FREE* indicating that the channel is free. When a message, i.e. a protocol data unit (PDU), is sent the channel must be free. After execution the channel contains the data unit. The actions *send* and *receive* are specified after **ACTIONS**. The identifier *channel* refers to the current state, the primed variable *channel'* accordingly to the next state. Similarly, the compositional cTLA process defined in Table 1 represents a fragment of a connection set up procedure. The processes that compose the compositional process are listed after **PROCESSES**. For each process, an instantiation is created which is represented by a name. For instance, *C*: *communicate*(*pdu: pdu_type*) which is defined as simple cTLA process. The **ACTIONS** definition part describes the actions belonging to the compositional process. An action describes a transition transferring the modeled system from a given state to another state.

The reasons for using cTLA to formalizing the semantics of activity diagrams are threefold: (i) Foremost, the formal semantics of cTLA enables an exact interpretation of the system design specification. (ii) As a temporal event-based system, it is possible to specify cTLA processes in a canonical form [8]. This form can be used to verify the system behaviors by introducing an appropriate verification mechanism, e.g. model checking. It is also possible to formulate time-ordering events as properties to prove whether it holds in the cTLA process or not. (iii) Due to its program-like structure, it is appropriate for model transformations into other FDTs, such as LOTOS, and into verification languages like PROMELA (PRocess MEta LAnguage) [21]. The approaches [8,11] also address the importance of using cTLA for protocol verification.

3 Formalizing the Semantics of UML Activity Diagrams

Our approach of formalizing the semantics of UML activity diagrams consists of two steps. First, each activity diagram node and a control flow are defined by simple cTLA processes. Then, we derive the *functional semantics* of the specification to unify the simple operations of the activity nodes and to interpret the specification

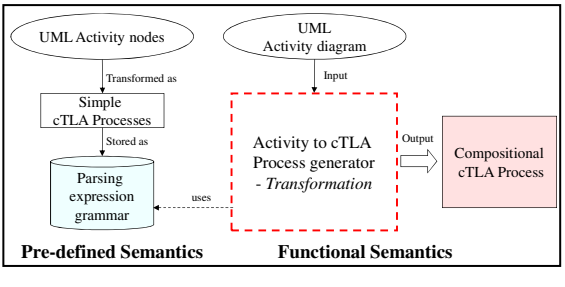


Fig. 2. Semantics definition process

formally. In principle, the *functional semantics* is based on the transformations of a design/program in a standard way for a given data set. Defining such kind of semantics has the advantage that the designer can model the system with any standardized UML tool. The resulting model can then be translated into a cTLA description by using our transformation method. In order to predefine the semantics of an activity diagram the behavior of each activity node is represented by a simple cTLA process. These processes are stored as rules in form of a *parsing expression grammar* (see Figure 2). The parsing expression grammar uses the simple cTLA process name as identifier for the respective UML activity node. The grammar is manually derived, but it must be done only once. Afterwards, the functional semantics of each activity diagram can be derived based on the predefined semantics of the activity nodes by generating the compositional cTLA process. We implemented this generation process in a tool - *Activity to cTLA Process generator.*

3.1 Semantics Definition for Activity Nodes

To get an idea of how to formalize the semantics of the UML activity nodes we outline here as an example the semantics definition of the activity node *action*. According to the UML document, the activities in UML have a Petri net like semantics, i.e. the semantics is based on a token flow. An *activity* node is described by token movements as transitions, and the placement of the tokens in the graph as states. In cTLA the token movement and tokens have to be represented by means of process parameters. The parameter *activity token* AT is assumed as a set of triggering tokens for activities. These tokens represent the data flow among activities.

The *action* node may possess multiple operations based on parameters which trigger the activity. For example, three different operations are defined for the *action* node: *pack* to build the protocol data unit, *unpack* to extract data from the PDU, and *analyze_data* to check the extracted data. Functions, such as *pack_PDU*(), *unpack_PDU*(), and *analyze_data*(), are external processes, i.e. sequential compositions. The *flag* acts as a guard for executing the three operations.

```
PROCESS Action(pdu_type: Any, flag: Any)
IMPORT pack_PDU, upack_PDU, analyze_data;
BODY
   VARIABLES pdata;

   INIT ≙ pdata = NULL;

   ACTIONS  execute(ad: pdu_type, at: flag) ≙ at ∈ {“pack”, “unpack”, “analyze”}:
         at = “pack” ∧ pack_PDU(ad) ∧ pdata´= at;
         at = “unpack ∧ unpack_PDU(ad) ∧ pdata´= at;
         at = “analyze” ∧ analyze_data(ad) ∧ pdata´ = at;
END
```

Our experience shows that this kind of definitions is straightforward and intuitive to protocol engineers and modelers. The other activity nodes, such as *send*, *fork*, *join*, *merge* and so on are defined in a similar way. A definition of all important nodes for protocol design can be found in [12]. We omit these definitions here for lack of space.

3.2 Functional Semantics

The objective of the functional semantics derivation is to formalize the designed activity diagram based on the simple cTLA process definitions for independent activity nodes introduced above. To identify the behavior of the activity diagram, however, we have to consider the nodes in connection with the whole diagram to be transformed. This issue is handled by our *Activity to cTLA Process generator* (see dashed rectangle in Figure 3). It comprises the following four simple steps.

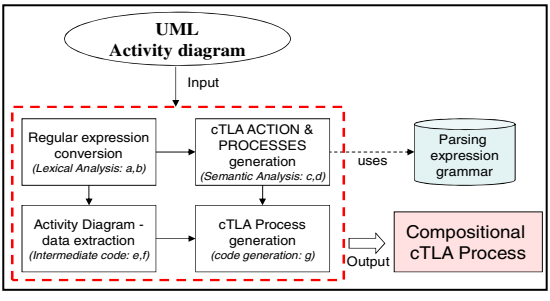


Fig. 3. Compositional cTLA process generation steps

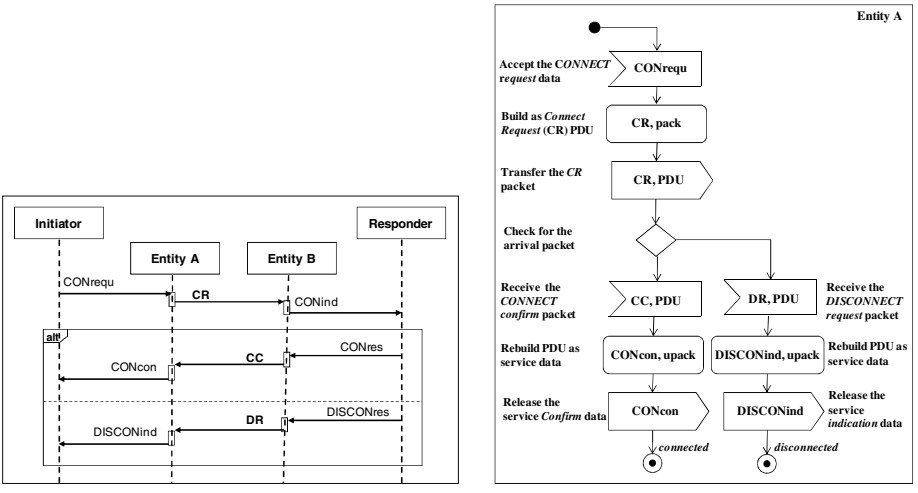
First, the UML activity diagram is converted into a deterministic finite automaton (DFA) to identify the hierarchical structure of the activity diagram. Thereafter, an equivalent regular expression (RE) is derived from the generated DFA. We use the regular expression as a source to generate a formal specification using formation rules. As formation rules we apply the *parsing expression grammar* extracted from the predefined simple cTLA processes. To accomplish this, two algorithms are applied: (a) the *Activity to DFA generation* and (b) the *DFA to RE conversion* algorithm.

Second, we generate the cTLA specification parts *ACTIONS* and *PROCESSES* based on the obtained regular expression to derive all definitions parts of the final compositional cTLA process. This is achieved using the *regular expression to* (*c*) *cTLA ACTION* and (d) *cTLA PROCESSES generation* algorithms.

Third, the skeleton of the compositional cTLA process is generated using the *ACTIONS* and *PROCESSES* definition parts by means of the (e) *cTLA process generation* algorithm. This process is similar to a functional definition in a programming language. In addition, the activity data are extracted from the DFA using the (f) *cTLA ACTION data extraction* algorithm.

Finally we combine all the generated definition parts to one compositional cTLA process using the data definitions of the UML activity diagram. This is achieved by means of the (g) *compositional cTLA process generation* algorithm.

To provide a fixed structure and to interpret the design in a sequential order we apply certain activity syntax constraints, e.g. each activity diagram must have exactly one *init* node. A full list of the constraints can be found in [12]. The observance of these constraints can be checked automatically by the *Activity to cTLA Process generator*. We explain the above four steps using an example: a connection establishment with a two-way handshake (see Figure 4).



a) Sequence diagram b) Activity diagram

Fig. 4. Example: The connection set up phase

The *initiator* and *responder* in the sequence diagram (Figure 4.a) represent the lifelines of the service users; *entities A* and *B* accordingly the lifelines of the protocol entities. At first, the initiator sends the service connection request (CONrequ) to entity A. Next, it is been encoded as a *connect request* (*CR*) protocol data unit (PDU) and sent for setting up a connection to *B*. Entity B receives the request and informs the service user. If the service user accepts the connection and entity B responds with *connect confirm* (*CC*), otherwise it sends a *disconnect indication* (DISCONind). In the latter case, a *disconnect request (DR)* PDU is sent to entity A. For brevity, we show the formalization for only entity A in Figure 4.b.

Deterministic Finite Automata Conversion: As first step in the cTLA process generation, an equivalent DFA is created from the given activity diagram. Here, the activity diagram nodes are marked with identifiers, such as *Initial – I*, *Action – A*, *Accept Event – R, Final* – F, etc. Thereafter, a directed graph is constructed by considering the activity diagram nodes as *states* and the activity labels as *transitions*. The directed graph is considered as a DFA $M = \{Q, \Sigma, \delta, q_0, F\}$ with Q - states, Σ - transition inputs, δ - state transition relations, q_0 – start state, and F - final state. The DFA tuple elements are identified. It may be possible that the directed graph is non-deterministic. In this case, it has to be converted into a deterministic automaton as required for protocol development. In the following we use the first letter of the activity nodes in Figure 4.b as abbreviation instead of the full name. Figure 5 shows the respective directed graph from which the tuple elements are constructed. Thereafter, we derive the regular expression (RE) from this DFA. In our example the depicted DFA directed graph algorithm requires 18 steps to obtain the regular expression: IRASD(RASF/RASF) (see Figure 5).

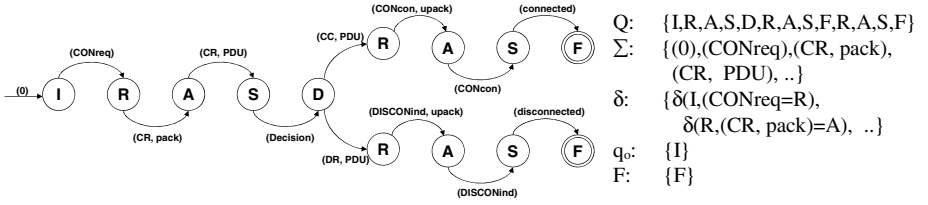


Fig. 5. Activity diagram to finite automaton conversion

cTLA Action and Process Generation: To obtain the cTLA *ACTIONS* definition part, another algorithm is introduced to read the regular expression and the parsing expression grammar (simple cTLA process). The cTLA *ACTIONS* are specified using propositional logic operators. In our example first a parse tree is constructed for the regular expression *IRASD(RASF/RASF)* (cf. Figure 5). The tree is updated by replacing the elements with the parsing expression grammar to obtain the cTLA *ACTIONS* part with no data value, for e.g. I → In.start(st: AT). Finally the cTLA *ACTIONS* part is specified in form of logical expressions as shown in Table 2.

Table 2. Generation of the cTLA ACTIONS and PROCESSES part

cTLA ACTIONS	cTLA PROCESSES
In.start(st: AT) ∧	In: Initial(AT: Any)
Acc.receive(rd: pdu_type) ∧	Acc: AcceptEvent(pdu_type: Any)
Act.execute(ad: pdu_type, at: flag) ∧	Act: Action(pdu_type: Any, flag: Any)
.....	.....
Fi.stop(fin:AT) ∨	Fin: Final(AT: Any)

The *cTLA PROCESSES* part is generated in two steps. First, the generated regular expression is analyzed by ignoring the special and repetition characters. Thereafter, the regular expression is parsed with the process expression grammar to generate the set of *PROCESSES*. In our example the obtained regular expression has been *IRASD*(*RASF/RASF*). The non-repeated non-terminals are extracted from the regular expression as *IRASDF*. Now the regular expression is parsed to obtain the corresponding cTLA *PROCESSES* part as shown in Table 2.

Generation of the Non-functional cTLA Process: Another algorithm is applied to obtain the skeleton for the compositional cTLA process as intermediate code. The algorithm uses a predefined compositional cTLA process frame (cf. Section 2) and the previously generated cTLA *ACTIONS* and *PROCESSES* parts.

We explain this step with respect to our example with the help of Table 3. The process name is the specification name, e.g. *ConEst_Entity_A*. The parameter *Entity B* indicates the partner entity (see line *A* in Table 3). The process state status (PSS) variable is declared to identify the current state and the successor state (PSS′). *PSS* is initialized with NULL of no value, since the state actions are not executed yet. Thereafter, the cTLA *ACTIONS* and *PROCESSES* parts from c) and d) (see Table 2) are included into the compositional cTLA process skeleton. *END* is added for process termination. The italic text in Table 3 indicates the data extracted from the activity

diagram. In principle, the activity data, e.g. CONrequ (cf. Figure 4), are extracted from the DFA and the regular expression and include them into the cTLA *ACTIONS* part of the compositional cTLA process. The *ACTIONS* in line *F* of Table 3 depicts the extracted data for our example. They are not yet included by this algorithm, but in the next step.

Table 3. Compositional cTLA process for entity *A* (fragment)

compositional cTLA process	
A. **PROCESS** ConEst_Entity_A(Entity_B)	/* Process name */
B. **BODY**	/* cTLA process structure begins */
C. **VARIABLES** PSS: {"connected", "disconnected"};	/* Process state status variables */
D. **INIT** ≙ PSS = NULL;	/* Variable(s) initialization */
E. **PROCESSES**	/* Processes declaration */
In: Initial(AT: Any);	/* Initial node */
Acc: AcceptEvent(pdu_type: Any);	/* Accept event node */
…….	
Dec: Decision(dec: Any, action: Any)	/* Decision node */
Fi: Final(AT: Any);	/* Final node */
F. **ACTIONS**	/* Process actions */
con(*du:SDU*) ≙	/* cTLA execution part begins */
(In.start(0) ∧	/* Initial state */
Acc.receive(*CONrequ*) ∧	/* Waiting for the SDU to arrive */
Act.execute(*CR, pack*) ∧	/* Encoding the SDU as data packet CR */
Sen.send(*CR, PDU*) ∧	/* Transferring the packet to receiver entity */
Dec.decide(*CC/DR*) ∧	/* Check for the arrival packet */
……	
Fi.stop(*disconnected*)));	/*End of unsuccessful connection set up */
G. **END**	/* cTLA process terminates */

Generation of the Compositional cTLA Process: In a final step the compositional cTLA process is generated. It reads the formerly generated cTLA process frame and updates the process state variable and the cTLA *ACTIONS* with the data extracted in the previous step. For example, *Acc.receive*(*rd:pdu_type*) is updated with *CONrequ* (see Table 3). This replacement establishes equality with the activity diagram. In principle, the token passing among the activity nodes is achieved by a variable *du* (data unit) (see Table 3 under *ACTIONS*). At the beginning, the initial node is triggered by means of *du=0* which then waits for a *CONrequ* to arrive from the initiator. Once arrived the variable *du=CONrequ* is forwarded to the activity node action. Here the actual data is encoded as *CR*-PDU and sent to entity B. Now the cTLA process waits for a reply from entity B, i.e., *CC/DR* to connect or disconnect the set up. Based on the reply, *du* is forwarded to an appropriate activity node. Similarly the entire cTLA ACTIONS carries the variable *du* for mapping data among the activity nodes.

The algorithmic steps described above are implemented, as already mentioned before, in a tool - the *Activity to cTLA Process generator* – to automate the transformation into cTLA. The algorithms are coded in JAVA. A screenshot of the tool is depicted in Figure 6. The transformation method given in this paper resembles to a compiler process, in which each part can be extracted for further development, e.g. generating test cases from a DFA.

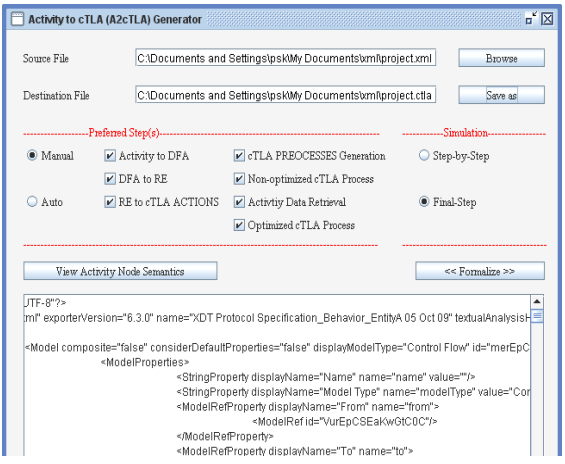


Fig. 6. Activity to cTLA generator – tool screenshot

To prove whether the transformed specification is correct or not, we provide a verification method in our protocol design approach [6]. Illustrating an approach for design verification goes out of the scope of this paper. Here, we only outline the principle. As mentioned in the introduction, we apply two descriptions in the design process, the communication- and the behavior-oriented representation. The communication-oriented description is used to visualize the protocol flow between the communicating entities. It acts as a design document. Based on this document and the given protocol requirements, the specifier (verifier) may formulate correctness properties, i.e. safety and liveness properties, the protocol has to meet. The properties are expressed in a temporal logic. The communication-oriented description is mapped automatically onto the behavior-oriented description, i.e. the activity diagrams [7]. In the next step the behavior-oriented representation is automatically transformed into a cTLA compositional process, as shown in this paper, to prove the semantic requirements. For verification, we map the generated cTLA meta-representation onto PROMELA and apply the SPIN model checker [21] to trace whether the properties hold in the obtained cTLA specification or not. The designer may use multiple protocol design components to specify a protocol in our approach. To verify both, model checking is carried out in two steps: (1) Each component possesses a concrete abstraction. They are first verified independently. Here, the formulated correctness properties are checked whether they hold in the PROMELA model or not. (2) Next, the composition is verified in two stages. First, it is proved whether the components are adapted properly by checking the output of the components with the specified input. This can automatically be checked for all components of the composition during adaptation phase. Second, the correctness of the composition is proved. For this, the component adaptations are tracked by means of temporal events, i.e., henceforth (□) and eventually (◊) operators followed by the component name, for e.g. *ConEst_Entity_A*. This temporal property is used to check the component adaptation path. The specification of the henceforth and eventually notations are based on the protocol requirements which is stated by the designer.

Now we derive the safety and liveness properties from the component's temporal property using inference rules, such as invariance, precedence, response and correlation from [21]. Finally we verify the constructed properties along with the (composed) PROMELA specification for correctness using SPIN. Since we verify all possible occurrences of events and options including related reactions, the completeness to the generated specification is eventually verified.

4 Related Work

We focus here on the formalization of UML activity diagrams. For this, Eshuis et al. [13] propose an approach by predefining temporal events (logic) based semantics to each activity diagram node. Likewise, [14] applies properties like safety, guarantee, etc. as predefined rules to generate a temporal logic based specification from the activity diagrams. However, both approaches do not address the use of temporal events during the activity diagram transformations onto other representations. Störrle and Hausmann [15] investigated the possibility of semantics mapping from activity diagrams into Petri nets. Their study concludes that the mapping is intuitive for the basic activities. Unfortunately, no such intuitive connection exists for high-level activities, such as exceptions, loops, etc. Abstract state machines (ASMs) are also applied to formalize UML activity diagrams. The notable approaches for this are from [16, 17] to formalize an entire activity diagram and to formalize the state charts, activities, and sequence diagrams [18]. Independent UML diagram formalization using ASM is realistic, but in compositions bulky ASMs are difficult to interpret. The *object constraint language* (OCL) [19] specification provides an easy way to express the semantics as constraints, but the problem is to interpret them between tools because the interpretation of the semantics may differ. The formal specification language Z [20] is used to specify real systems. However, it is not ideal for all problems, e.g. dealing with concurrency is clumsy. Besides, the toolset for Z is not very well advanced by industrial standards. The related work shown here is not exhaustive. The reason for using cTLA as formal semantics over the existing technologies is the following. (i) cTLA supports modular formal specifications [8] of distributed systems and applies state transition system based modeling like standardized FDTs, such as SDL. This fosters easy-to-read formal descriptions of the systems. (ii) cTLA has been designed with the objective of explicit verification support through TLA-based verification. (iii) cTLA is a constraint-oriented language where the designer can define the constraints which the system has to meet explicitly during design. For instance, the limitations to model an activity diagram can be predefined in the formalization tool. The *Activity to cTLA Process generator* automatically detects the error cases in the design and an error notification is displayed.

5 Final Remarks

In this paper we have presented a procedural approach to assign a formal semantics for a widely applied class of UML diagrams, namely activity diagrams, using the

compositional Temporal Logic of Actions (cTLA). The formalization forms the basis of our component-oriented method for a UML based design and development of communication protocols and distributed systems [6]. The method applies two perspectives: the communication-oriented using UML sequence diagrams and the behavior-oriented representation using UML activity diagrams. Both views are complementary and synchronized with each other as shown in [7]. The communication perspective supports the intuitive design by representing the interactions between the protocol entities. The behavior perspective describes how these interactions are "implemented" by the entities. The activity diagram specifications form the basis for the further development stages and require therefore a formal basis.

The use of UML in the protocol and system development has several advantages compared to traditional FDT-based approaches. Unlike FDTs which enforce a semantic-oriented description, UML better supports an intuitive modeling of protocols and distributed systems which allows in particular a reuse of the designs. In contrast to FDTs, however, UML does not possess a formal semantics which is required for a unique interpretation of the specification and the transformation into other formal representations. For this reason, we developed this formalization procedure. We chose cTLA because of its constraint-oriented specification which is in particular helpful for the subsequent verification of the specification. The formalization is performed in two steps. First, we introduce predefined semantics for the activity diagram nodes using simple cTLA processes. Next, the functional semantics of the specification is derived by mapping the activity diagram onto a compositional cTLA process. The transformation is implemented as a tool. Unlike the referenced approaches of Section 4 which define formal semantics for a specific UML tool, our approach allows designing UML models in any UML 2.x supported tool and to formalize the same using the *Activity to cTLA Process generator* tool. This presents a general approach for UML designers to formalize their specifications in temporal logic. The main advantage of the translation shown in this paper is to support automated design verification for the UML modeling. Here we explicitly refer to our work [22] which provides an automatic transformation from cTLA onto PROMELA. For design verification, we outlined the approach in Section 3.2. Formalizing the semantics of UML diagrams raises the UML based design to the level of FDT-based protocol and system development. The designer can benefit, on the one hand, from the intuitive modeling features provided by UML and, on the other hand, use the rich experience of the FDT-based design applying established methods for verification, performance evaluation, code generation, and testing.

Currently, we apply our the *Activity to cTLA Process generator* to deriving formal semantics to frequently used protocol mechanisms, such as connection set up, flow control, automated repeat request, connection release, and others which are provided in our component library. It is further planned to connect our tool with SPIN as add-on. This allows the designer to apply the tool to formalize and verify his/her specification even without a profound knowledge on logics because of the enriched graphical interface.

References

1. ITU-T Recommendation Z.100: Specification and Description Language (2000)
2. ISO LOTOS: A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, IS 8807 (1988)
3. Babich, F., Deotto, L.: Formal Methods for Specification and Development of Communication Protocols. IEEE Comm. Surveys and Tutorials 4, 2–20 (2002)
4. Object Management Group: Catalog of UML Profile Specifications (2011)
5. Object Management Group: UML Superstructure Specification Document (2009)
6. Kaliappan, P.S., König, H., Schmerl, S.: Model-Driven Protocol Design Based on Component Oriented Modeling. In: Dong, J.S., Zhu, H. (eds.) ICFEM 2010. LNCS, vol. 6447, pp. 613–629. Springer, Heidelberg (2010)
7. Kaliappan, P.S., König, H.: An Approach to Synchronize UML-Based Design Components for Model-Driven Protocol Development. In: 34th Annual IEEE Software Engineering Workshop. IEEE, Limerick (2011)
8. Herrmann, P., Krumm, H.: A Framework for Modeling Transfer Protocols. Computer Networks 34(2), 317–337 (2000)
9. Kraemer, F.A.: Arctis and Ramses: Tool Suites for Rapid Service Engineering. In: Proc. of the Norwegian Informatics Conference, Oslo, Norway (2007)
10. Lamport, L.: Specifying Systems. Addison Wesley (2002)
11. Graw, G., Herrmann, P., Krumm, H.: Verification of UML-Based Real-Time System Designs by Means of cTLA. In: Proc. of the 3rd IEEE Int. Symposium on Object-Oriented Real-Time Distributed Computing. IEEE (2000)
12. Kaliappan, P.S.: cTLA-based Semantics Specification for UML Activity Diagram. Technical Report, Computer Science Department, Brandenburg University of Technology Cottbus (2010), http://www-rnks.informatik.tu-cottbus.de/de/node/334
13. Eshuis, H., Wieringa, R.J.: A Formal Semantics for UML Activity Diagrams - Formalizing Workflow Models. CTIT technical reports series (2001)
14. Araújo, J., Moreira, A.: Integrating UML Activity Diagrams with Temporal Logic Expressions. In: Proceedings of the 10th International Workshop on Exploring Modelling Methods for Systems Analysis and Design, Portugal (2005)
15. Störrle, H., Hausmann, J.H.: Towards a Formal Semantics of UML 2.0 Activities. In: Proc. of the German Software Engineering Conference, vol. P-64 (2005)
16. Börger, E., Cavarra, A., Riccobene, E.: An ASM Semantics for UML Activity Diagrams. In: Rus, T. (ed.) AMAST 2000. LNCS, vol. 1816, pp. 293–308. Springer, Heidelberg (2000)
17. Sarstedt, S., Guttmann, W.: An ASM Semantics of Token Flow in UML 2 Activity Diagrams. In: Virbitskaite, I., Voronkov, A. (eds.) PSI 2006. LNCS, vol. 4378, pp. 349–362. Springer, Heidelberg (2007)
18. Jürgens, J.: Formal Semantics for Interacting UML Subsystems. In: Proc. of the IFIP 5th Intl. Conf. on Formal Methods for OODS, vol. 209, pp. 29–43. Kluwer B.V (2002)
19. Object Constraint Language: Object Management Group (2011)
20. Information Technology — Z Formal Specification Notation — Syntax, Type System and Semantics (ISO/IEC 13568:2002 ed.), p. 196 (2002)
21. Holzmann, G.J.: The SPIN Model Checker. Addison-Wesley (2006)
22. Kaliappan, P.S., König, H.: Model Transformation from cTLA onto Promela for Model Checking the Protocol Designs. Technical Report, Computer Science Department, Brandenburg University of Technology Cottbus (2011), http://www-rnks.informatik.tu-cottbus.de/de/node/334

Tree Based Domain-Specific Mapping Languages

Elina Kalnina, Audris Kalnins, Agris Sostaks, Edgars Celms, and Janis Iraids

University of Latvia, IMCS, Raina bulvaris 29, LV-1459 Riga, Latvia
{Elina.Kalnina,Audris.Kalnins,Agris.Sostaks,
Edgars.Celms,Janis.Iraids}@lumii.lv

Abstract. Model transformation languages have been mainly used by researchers – the software engineering industry has not yet widely accepted the model driven software development (MDSD). One of the main reasons is the complexity of metamodelling principles the developers are required to know to actually use model transformations in the way the OMG has stated. We offer the basic principles how to create domain-specific model transformation languages which can be used by developers relying only on familiar modelling concepts. We propose to use simple graphical mappings to specify the correspondence between source and target models which are represented using trees based on the concrete syntax of underlying modelling languages. If such principles were followed, then the range of potential users of model transformation languages would increase significantly.

Keywords: mappings, domain-specific languages, UML, model transformation languages.

1 Introduction

Model transformations are the key feature to deal with models in any model driven (MD) technology. However, usage of model transformation languages requires highly skilled specialists with deep knowledge of metamodelling. That is one of the reasons why the industry has not yet widely accepted the MD approaches.

Domain-Specific Modelling (DSM) proposes to use modelling languages which use notation and concepts specific to the domain actually being modelled. Similarly, we propose to use domain-specific transformation languages which use elements specific to models being transformed. Most of model transformation languages use abstract syntax (metamodels) to specify model transformation definitions. However users of the modelling languages use only the concrete syntax of the language. Thus, the domain-specific model transformation language should use familiar concepts for modelling experts: the concrete syntax of the modelling language.

Another crucial aspect for a domain-specific model transformation language is the use of convenient means to represent correspondences between source and target model elements in the model transformation definition. The most intuitive option to define model transformations is to use *mappings*. Mappings permit to specify

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 492–504, 2012.

transformations in a simple way, by very intuitive graphics. The expressive power of general purpose mapping languages is limited, however, we show that mappings are expressive enough for transformations in specific domains.

This should lead to the shift of roles of developers in the model driven software development (MDSD) process. The metamodelling experts (highly skilled professionals) would be the developers of domain-specific transformation languages using all the arsenal of technologies they have. The software developers (modellers) would become the actual developers and users of model transformations. Thus the former model transformation users would become model transformation developers (and users), but former model transformation developers would become model transformation language developers.

In this paper we study how to build domain-specific transformation languages based on simple mappings and the concrete syntax of models. One concrete mapping language of such type - MALA4MDSD is proposed. A general approach how to build similar languages is sketched. This approach helps us in reaching simplicity, readability and sufficient expressiveness of the transformation language at the same time.

In §2 we describe the basic principles for obtaining domain-specific mapping languages using our approach. In §3 we describe MALA4MDSD where MDSD-related transformations from a UML subset to UML subset can be specified in a simple way. We compare our approach with the traditional transformation development on a MDSD use case. In §4 facilities required for precise definition of such mapping languages are described. We complete this paper with the related work description in §5.

2 Basic Principles of Mapping Language Family

Though models being transformed by model transformations are graphs, we propose to base our mapping language definition on the simplest model syntax – model trees. A significant part of model structure in modelling languages such as UML is determined by containment relations, therefore it is natural to represent the basic contents of a model by a tree. A transformation in the defined mapping language (a mapping diagram) will consist of source and target trees with mappings represented graphically by arrows from a source tree node to target tree node (see Fig. 1).

Our approach to mapping language definition is based on the concept of *tree type* as a very lightweight equivalent of metamodelling. First, a tree type definition can be considered as a successor of XML Schema language [19] – while a schema defines the permitted tree structure of a XML document, a tree type defines the structure of a model represented as a tree. The tree type for a modelling language defines the possible *node types* in a model tree and the permitted containment relations between them. Each node type typically corresponds to a modelling language element; hence its graphical representation can be borrowed from the concrete syntax of this element. In addition, each node type has a set of *attributes* – proper attributes in the sense of domain metamodel and also references to other metamodel classes such as *type* in UML. A tree type has also some analogy to graph type used by graph transformation languages [18]. In line with this, we introduce *edge types* by adding edges to the tree

type – as lines linking a pair of node types. Edge types may also have attributes. Each concrete model tree is an *instance tree* compliant to its tree type.

The cornerstone of a mapping language is source and target trees. They are based on the corresponding tree types: one if the source and target is in the same language (as it is for MALA4MDSD, see Section 3) or two if they are different.

Mappings in a diagram go from nodes in the source tree to those in the target tree. The *source tree* of a *mapping diagram* represents the way how we want to process an instance tree in the given mapping diagram, it is a sort of pattern for selecting appropriate sets of instance tree nodes to be mapped to the target. More precisely, a source tree can be obtained from the corresponding tree type by adding appropriate filtering conditions (attribute-based expressions) to nodes. This way a node type may be repeated under its parent as many times as required, each time with a different filter expression. The aim of a source tree node as a starting point of a mapping is to represent the set of instance tree nodes to be processed by the given mapping. Edge types defined in the tree type may also appear in the source tree to represent a set of edges to be mapped to the target.

The target tree of a diagram provides the description, how individual target nodes created by each mapping should be assembled into the target tree. Certainly, it must comply with its tree type, and nodes again may be duplicated when mappings require this. The target tree nodes contain assignments for some or all of the attributes defined in the tree type, they specify expressions how the attribute value is computed when a node instance is created.

Source and target tree nodes are connected using named *mapping relations* (arrows). A mapping relation means that if an instance corresponding to the source node is found in the source model then an appropriate instance should be created in the target model (we should think of both models to be represented by their instance trees). For each valid instance of a source node the outgoing mappings are executed (i.e., the target instances created and their attributes set). Validity of an instance is checked using the containment relationship to parent and filter conditions. In addition to this basic mapping kind, mappings with some modifiers are also supported.

Mapping relations in a mapping diagram are *ordered* top-down – according to their start position in the source tree. Mappings are executed according to their ordering. This natural execution order ensures that in typical cases children of a source node are processed after their parent and since the hierarchy is preserved at the target end it is easy to find where the newly created target node instances are to be attached.

The general “create if not exists” semantics is used for mappings – a duplicate target instance defined by the same type of mapping is never created. This is ensured by creating a special named *trace edge* linking the source and target instances.

3 Proposed Mapping Language for MDSD

In this section one concrete mapping language based on our approach is proposed - MALA4MDSD (MApping LAnguage for MDSD). This language is built to transform one UML model to another UML model. Currently only a UML subset describing the

static structure of a UML model is used (however, it could be easily extended to include behaviour-related elements as well). Typical application of such language is transformations from PIM (Platform Independent Model) to PSM (Platform Specific Model) in the MDSD lifecycle.

The only formal element of the language definition from its user point of view is the *simpleUML* tree type (only one since the source and target is of the same type). This definition is inspired by real model trees in UML tools. See Section 4 for some fragment of a more formal definition of this tree type. The root node is always a *Model*. A *Model* can contain *Packages.* A *Package* can contain other *Packages, Classes, Interfaces, Components, DataTypes, Actors,* and *Enumerations. Component* can contain provided *Interfaces. Class* and *Interface* are allowed to contain *Attributes* and *Operations. Operations* contain their *Parameters.* These node types are depicted graphically by shapes and icons inspired by UML tools.

There are also four types of edges: *Association* (from *Class* to *Class*), *Generalisation* (from *Class* to *Class*), *Dependency* (from *Class* to *Class*) and *Realisation* (from *Class* to *Interface*). All node types have the *name* attribute (a *String*). *Class*, *Attribute*, *Operation*, *Interface* and *Association* have also the *stereotype* (also a *String*). *Attribute*, *Operation* and *Parameter* have the *type* attribute (in the sense of UML), for us it can be a *String* (names of UML primitive types) or a reference to *Class*, *Interface* or *Enumeration* instance in the model tree.

3.1 Basics of MALA4MDSD

On the basis of the defined *simpleUML* tree type both source and target trees in a MALA4MDSD mapping diagram can be built. The source tree has to specify in a hierarchical form which nodes in the source instance tree (in fact, the source UML model) must be processed, and by which mappings.

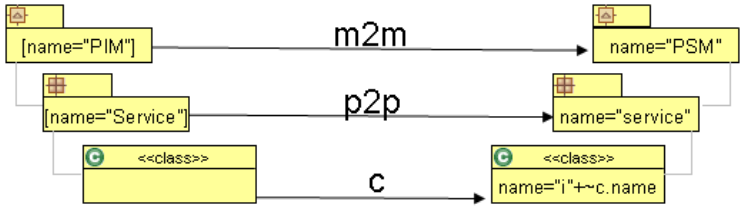


Fig. 1. MALA4MDSD example. UML model "PIM" is transformed to UML model "PSM". Package "Service" in model "PIM" is transformed to package "service" in "PSM" model. Classes from source package "Service" are transformed to classes in target package "service".

A simple mapping diagram example is presented in Fig. 1. The topmost mapping relation is executed first. It maps two UML Models. In the source instance tree a UML model named "PIM" is sought for (there is a filter expression on the *name* attribute in the *Model* node). For each such model a UML model named "PSM" is

created in the target (formally, as the root node in the target instance tree, the *name* attribute is set by a constant expression). Then packages named “Service” in the UML model ”PIM” are sought for. For each such package (in fact only one exists) the corresponding package named “service” in the target UML model “PSM” is created. The third mapping relation processes all classes in the source model package “Service” and creates the corresponding classes in the target model package “service”. The prefix “i” is added to the source model class name – note, how the reference “~c” is used to navigate the mapping named “c” from target to source. Thus the expression “~c.name” gives us the name of the mapping source node (class). In general, the expression syntax is a very simple subset of OCL (referencing the tree type elements).

3.2 More Advanced Mapping Elements

Elements described in the previous sections are the core of the proposed mapping language. To facilitate transformation development in this mapping language some more features are introduced.

For some tasks large source and target trees with many mapping relations must be built, therefore there is a need to divide mappings into smaller sub-diagrams. One mapping program (transformation) consists of several ordered mapping diagrams. They are executed separately in the given order.

There are special mapping modifiers. A mapping with the *copyAttributes* modifier specifies that in the target node for each attribute an implicit assignment is performed setting it to the value corresponding to that attribute value in the source node.

The *copy* modifier is even more powerful. It specifies that implicit mappings for all children types of the node (at any depth, according to the tree type definition) with the *copyAttributes* modifier are performed. This is a powerful feature for copying tree fragments. Certainly, the node types for *copy* must be the same. In Fig. 2 the *copy* modifier is used for class nodes. Child elements for classes are attributes and operations, operations in turn are copied with their parameters.

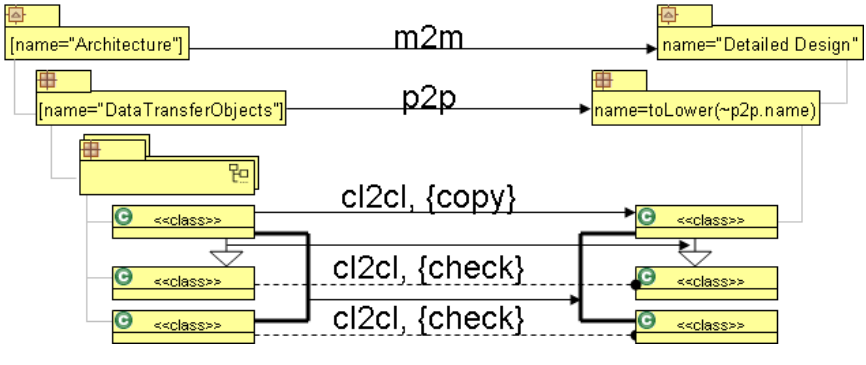


Fig. 2. Mapping example from ReDSeeDS project [13]. Transformation in MALA4MDSD where edge processing and hierarch flattening is demonstrated.

The third mapping modifier *check* means that nothing is created in the target tree only the relevant node is found using the traces between source and target (another kind of arrowhead is used here). Such mappings are necessary, e.g., for locating edge endpoints in the target tree as in Fig. 2.

As it was already mentioned in the previous section a UML package can contain other packages. It means there could be a package hierarchy with arbitrary depth. Sometimes there is a need to process this hierarchy and similar ones in a generic way. Therefore in our mapping language the concept of *transitive (recursive) containment* is introduced. There is a specially marked node, which represents not a single hierarchy level, but the whole set of hierarchy levels containing this node type (it is a *transitive node*). If there is a mapping from such a node it must go to a transitive node in the target tree. In this case the whole source hierarchy represented by this node is reproduced in the target tree. The tree fragment below a transitive node has the standard meaning.

The mapping diagram in Fig. 2 shows an example of transitive node – it is the 3rd node in the source tree. This node represents any level of nested packages below a package represented by the 2nd node of the tree. Here the marking is used only in the source tree (no outgoing mapping from this node), therefore the mappings from the class nodes below create classes in the target tree ignoring the fact how deeply the source instances were found in the source tree.

An edge mapping is also shown in Fig. 2. The edge processing is done after both nodes connected by this edge have been processed. All *Associations* and *Generalizations* between classes in the defined package hierarchy are copied to the target. All classes in this hierarchy have already been copied before the edge processing (by the mapping cl2cl). To find for an association the other end class in the target the mapping cl2cl is duplicated from another class node in the source (the other end of edge in the source) but this time with the *Check* modifier.

Sometimes composition relationships alone are not sufficient to define the mapping application context. Therefore it is possible to use source patterns in order to increase the expressiveness of the mapping language. Patterns are similar to the ones in the transformation languages, however only the node types and edge types defined in the tree type may be used in a pattern. To make mapping language less verbose, mapping relations with an application condition and conditional expressions in the target are introduced.

It is not possible to write an arbitrary transformation between models in this mapping language. Therefore we introduce a special tree node type named “Custom transformation”. This feature permits to combine mappings and transformations.

3.3 Mapping and Transformation Comparison

In this section we will compare UML to UML transformation development in the mapping language MALA4MDSD and in a traditional transformation language.

In IST 6th framework project ReDSeeDS a model driven path from requirements to code is investigated. Two different transformation sets (“styles”) from requirements to code have been developed (for details see [13]). Each set contains a different structure

of PIM and PSM and different transformations between them. These transformations have been developed in the model transformation language MOLA [12]. We have rewritten these transformations in the language MALA4MDSD. Table 1 contains statistics about transformations in MOLA and transformations in MALA4MDSD.

Table 1. Comparison of transformations from PIM to PSM developed using Model transformation language MOLA and mapping language MALA4MDSD. One table row presents comparable elements in both languages.

Basic Style	MOLA procedures	19	3	MALA4MDSD diagrams
	MOLA rules	84	19	MALA4MDSD mappings
	MOLA class elements	265	29(source:11; target:18)	MALA4MDSD nodes
Keyword-based Style	MOLA procedures	51	8	MALA4MDSD diagrams
	MOLA rules	137	41	MALA4MDSD mappings
	MOLA class elements	418	66(source:27; target:39)	MALA4MDSD nodes

In the left hand side of Fig. 3 one mapping diagram from the Keyword-based style transformations is presented. In this diagram copying of Classes, Interfaces and Interface realizations from the PIM model to appropriate place in the PSM model is presented. Classes and Interfaces in the PIM model can be located in the sub-package hierarchy under the packages "businesslogic" and "applicationlogic" (the 3rd node – a transitive one - in the source tree represents the package hierarchy). The same sub-package hierarchy should be reproduced in the target model.

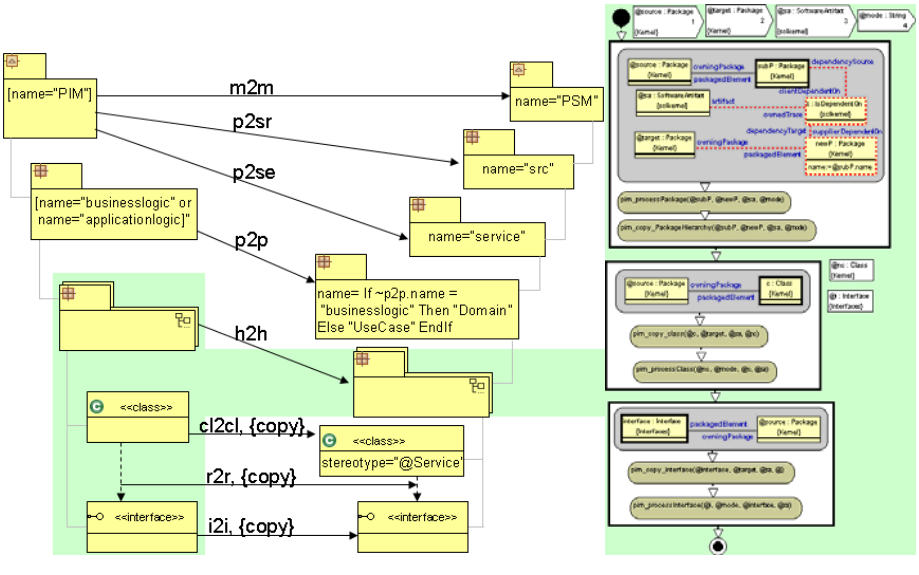


Fig. 3. Mapping example from ReDSeeDS project. Transformation in MALA4MDSD on the left. MOLA transformation for the highlighted part of the same task is presented on the right.

In the right hand side a part of MOLA transformations implementing the same logic is presented. Actually here only the package hierarchy is processed and class and interface copiers are invoked. All the copy logic is defined directly in other MOLA procedures. This copy logic description is quite long as there has to be described that attributes, operations and operation parameters should be copied and how they should be copied in terms of UML metamodel. The mapping part above the package hierarchy symbols is described in another MOLA procedure. Interface realization processing is also not visible in this MOLA transformation.

The reader may get the impression that MOLA is not a suitable language for this task and other transformation languages would do better. However it is not the case. Transformation languages usually deal with UML in its abstract syntax. Therefore all processing of all classes and associations according to the UML metamodel should be precisely defined. In the mapping language UML logical elements (a sort of concrete syntax) are processed and user should not care whether this logical element is represented with instance of one class or with instances of two classes connected with an association (and so on) in the UML domain. One more thing that is easier in a mapping language is the hierarchy processing. It is very easy to define that we have to process elements in some hierarchy or that we have to process relations with source and target ends in some hierarchy.

We have tried to rewrite the same fragment presented in Fig. 3 also in the most popular model transformation language – ATL [10]. The amount of code was similar to the amount of code in MOLA.

To sum up, a significant gain in conciseness of transformation specification using the mapping language compared to traditional transformation languages is obtained by using copy operators, convenient hierarchy processing and logical elements instead of abstract syntax elements. There is also a significant gain in the language understandability and usability because users are familiar with such logical elements typically to be seen in model trees within UML tools. There is no more need for transformation developers to know the coding of these logical elements in UML metamodel.

4 Mapping Language Definition

The precise definition of the general mapping language execution (semantics) as far as provided in previous sections was only from an instance tree to instance tree. However in real life there are only models (compliant to their metamodels). So there must be facilities for going from a model to tree and vice versa.

We propose a uniform solution how to relate models in a modelling language (such as UML) to trees conforming to a tree type based on the language (e.g., the tree type *simpleUML* for MALA4MDSD). Certainly, we assume the *metamodel* of the language in MOF to be given. The solution – *domain-to-tree mapping* is based on the tree type itself. It extends the tree type by OCL expressions based on the metamodel and few predefined keywords. Our mapping definition will directly show how an instance tree could be extracted from a model. But it will provide also an indirect solution for the reverse mapping.

For each *node type* we will specify on which metamodel class it is based (using the *Class* keyword). In addition, a selection expression in OCL can also be provided if not all class instances qualify. Further, for each *attribute* we want to include in the node type an OCL expression extracting the relevant value from a model must be provided. If that expression is to return a reference to another node type in our tree type, the *Node* function is used (with an appropriate class as the argument).

For each *containment* (parent-child) *relation* an OCL navigation expression specifying how child instances can be reached from the parent in a model must be provided (after the keyword *Path*). A node with *transitive containment* (such as Package in UML) must provide a special *Path* expression within it – how the next contained instance of the same type may be reached.

Similarly, for *edge types* the metamodel class they are based on must be specified. Attributes are specified the same way as for nodes. A new element is the *end specification* – the path by which the relevant end node instance can be found.

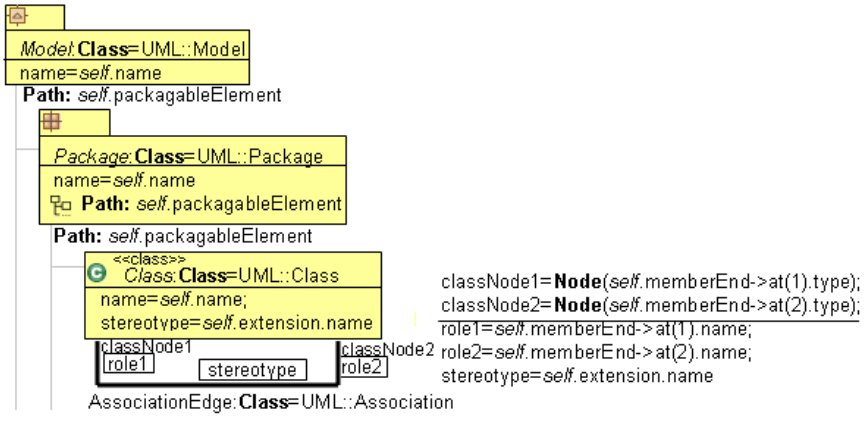


Fig. 4. Mapping language definition. Fragment from MALA4MDSD definition.

Fig. 4 illustrates how the tree type *simpleUML* can be defined on the basis of the slightly simplified UML 2 metamodel. Only the top three node types: *Model*, *Package* and *Class* are visible in the fragment but the continuation is quite similar. For all three node types the *name* attribute is defined in a natural way (the OCL *self* points to the node base class). The containment relation in all cases is defined by the same OCL navigation expression *self.packagableElement*. Only the association edge is visible in the fragment. Role and stereotype attributes are defined for it. Since both ends of an association are attached to classes, two similar *end specifications* are given.

It should be completely transparent now how an instance tree of the given type could be extracted from the model. The reverse operation – to obtain a model from tree is in most cases quite direct as well, but in complicated cases an explicit reverse mapping can be required. This mapping definition serves also as a basis for creating a MALA4MDSD implementation in a generic way. One more element to be defined is

how to "implement" at the model level the special *trace* edge between trees. Typically a special traceability class with associations should be added to metamodels.

5 Related Work

Attempts to create universal mapping languages as a certain alternative to traditional transformation languages have been started sufficiently early.

Hausmann and Kent in [9] used the term mapping to address the general understanding of connection between models and offered a graphical mapping language to specify mappings. However, the precise functionality of mappings had to be defined in OCL thus these relatively simple diagrams actually meant a complicated programming in OCL. In the thesis of Lopes [16] the Hausmann's and Kent's ideas have been developed much further – a mapping specification language has been created and implemented. However the usage of abstract syntax (standard UML metamodel) has led to complicated mappings even for simple tasks. In Malan language [2] mappings may be visualised as simple arrows between metamodel classes, however the actual mappings are specified using declarative textual definitions.

The Atlas Model Weaver (AMW) [6] provides a generic infrastructure and editor to declaratively specify weaving models between two arbitrary models. The weaving models are used to capture different kinds of links between model elements. AMW provides a generic mapping metamodel which should be extended in particular case.

The most recent approach uses composite Mapping Operators (MOps) [21]. The basic mapping operators called kernel MOps provide the basic types of possible mappings (like class to class, attribute to attribute, relation to relation, etc.). Kernel MOps can be composed into more advanced mapping operators – composite MOps.

Another view on mapping languages is given by Guerra et al. [8], where it is proposed to use mappings as requirements specifications for transformations.

All abovementioned mapping languages are general purpose ones, applicable to any domain and are based on the abstract syntax. Unlike the approaches described above, we propose to base the mapping language on a concrete syntax of source and target languages. For transformation languages a similar idea has already been applied, for example in AToM^3[15], and by Grønmo in [7]. The concrete syntax is used directly in model (graph) transformation rules. However, this approach lacks the simplicity and power of representation of correspondences between model elements offered by mapping languages. Transformation examples are also specified as mappings in a concrete syntax within the model transformation by example (MTBE) approach [22]. However, the MTBE approach requires a reasoner for transformation synthesis while in our approach the defined mappings are complete transformation definitions.

Though our approach to transformations is based on models in the form of trees, we have made a brief overview of transformation languages operating on structured textual data represented by trees. XSLT [20] is the transformation language used to transform data in XML format. Although XSLT itself is an XML-based textual language, there are tools which use mappings to represent XSLT transformations, for example Stylus Studio XSLT Mapper [1]. The source and target schemas are

represented by fixed trees and all transformation logic are specified using much more complex mapping features than it has been done in our approach. Another field where data are trees is program rewriting. However, the tools and languages, like Stratego/XT [3] or TXL [4], are intended for the analysis, manipulation and generation of programs.

6 Conclusions

In this paper the use of domain-specific mapping languages is discussed. It is proposed to define model transformations using simple mapping relations and tree syntax of source and target models. As a result it is possible to define typical model transformations in terms familiar to modellers. Therefore these domain-specific mapping languages could be applied by a much wider class of users.

The proposed general principles have been applied to a family of mapping languages where a language for a specific domain is defined by specifying the tree syntax for source and target. One specific mapping language – MALA4MDSD for transformations from PIM to PSM (a UML subset to UML subset) is discussed in some details. A concrete syntax similar to model trees in UML tools is used for source and target models. The transformation development in this language is compared to transformation development in a traditional model transformation language. A significant gain both in transformation size and understandability has been noticed since there is no need to deal with the technical details of the UML abstract syntax.

We propose a generic approach for the creation of domain-specific mapping languages. To define a mapping language, the tree types of source and target trees and their relations to models should be defined. This should be done by an expert in metamodelling and OCL. However, this should be done only once for a mapping language. Of course, the creation of a mapping language pays off only if multiple transformations in the same domain should be defined.

The proposed approach has been successfully checked also on the standard show case for transformations – the UML to RDBMS transformation (according to annex A of [17]). A domain-specific mapping language has been created in which the whole example (except the support for complex attributes) can be specified by one mapping diagram with 10 relations; in this language not only compositions but also associations and generalizations are adequately represented by tree containment. Other appropriate areas for the approach are transformations of UML to XML, RDBMS to ontologies and similar ones where the analysis of source model is simple enough. However, for transformation tasks which involve complicated graph-based source model analysis (for example, BPMN to BPEL [5]) the traditional pattern and rule based paradigm supported by most of transformation languages is better.

Mapping language family is being implemented using the DSL tool definition framework METAclipse [14] and Higher-order transformations in Template MOLA [11]. In METAclipse framework the generic (for the family) behaviour of a mapping language editor and compiler is defined using model transformations in MOLA [12].

The mapping language-specific MOLA transformations for the editor and compiler are generated from the tree type definition using Higher-order transformations.

Acknowledgments. This work has been partially supported by the European Social Fund within the project "Support for Doctoral Studies at University of Latvia".

References

1. XSLT Mapper, http://www.stylusstudio.com/xslt_mapper.html
2. Blouin, A., Beaudoux, O., Loiseau, S.: Malan: a mapping language for the data manipulation. In: Proceeding of the Eighth ACM Symposium on Document Engineering, DocEng 2008, pp. 66–75. ACM (2008)
3. Bravenboer, M., Kalleberg, K.T., Vermaas, R., Visser, E.: Stratego/XT 0.17. A language and toolset for program transformation. Science of Computer Programming 72(1-2), 52–70 (2008)
4. Cordy, J.R.: The TXL source transformation language. Sci. Comput. Program. 61, 190–210 (2006)
5. Dumas, M.: Case study: BPMN to BPEL model transformation. Oryx, 6–9 (2008), http://is.ieis.tue.nl/staff/pvgorp/events/grabats2009/cases/grabats2008translationcase.pdf
6. del Fabro, M.D., Bézivin, J., Jouault, F., Breton, E., Gueltas, G.: AMW: a generic model weaver. In: Proceedings of the 1ère Journée sur l'Ingénierie Dirigée par les Modèles (2005)
7. Grønmo, R., Møller-Pedersen, B.: From Sequence Diagrams to State Machines by Graph Transformation. In: Tratt, L., Gogolla, M. (eds.) ICMT 2010. LNCS, vol. 6142, pp. 93–107. Springer, Heidelberg (2010)
8. Guerra, E., de Lara, J., Kolovos, D.S., Paige, R.F., dos Santos, O.M.: *trans*ML: A Family of Languages to Model Transformations. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010. LNCS, vol. 6394, pp. 106–120. Springer, Heidelberg (2010)
9. Hausmann, J.H., Kent, S.: Visualizing model mappings in UML. In: Proceedings of the 2003 ACM Symposium on Software Visualization, SoftVis 2003, pp. 169–178. ACM (2003)
10. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)
11. Kalnina, E., Kalnins, A., Celms, E., Sostaks, A.: Graphical Template Language for Transformation Synthesis. In: van den Brand, M., Gašević, D., Gray, J. (eds.) SLE 2009. LNCS, vol. 5969, pp. 244–253. Springer, Heidelberg (2010)
12. Kalnins, A., Barzdins, J., Celms, E.: Model Transformation Language MOLA. In: Aßmann, U., Aksit, M., Rensink, A. (eds.) MDAFA 2003. LNCS, vol. 3599, pp. 62–76. Springer, Heidelberg (2005)
13. Kalnins, A., Sostaks, A., Celms, E., Kalnina, E., Ambroziewicz, A., Bojarski, J., Nowakowski, W., Straszak, T., Riediger, V., Schwarz, H., Bildhauer, D., Kavaldjian, S., Popp, R., Falb, J.: Final reuse-oriented modelling and transformation language definition. Project Deliverable D3.2.2, ReDSeeDS Project (2009), http://www.redseeds.eu
14. Kalnins, A., Vilitis, O., Celms, E., Kalnina, E., Sostaks, A., Barzdins, J.: Building tools by model transformations in Eclipse. In: Proceedings of DSM 2007 Workshop of OOPSLA 2007, pp. 194–207. Jyvaskyla University Printing House, Montreal (2007)

15. de Lara, J., Vangheluwe, H.: AToM: A Tool for Multi-Formalism and Meta-Modelling. In: Kutsche, R.-D., Weber, H. (eds.) FASE 2002. LNCS, vol. 2306, pp. 174–188. Springer, Heidelberg (2002)
16. Lopes, D., Hammoudi, S., Bézivin, J., Jouault, F.: Generating Transformation Definition from Mapping Specification: Application to Web Service Platform. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 309–325. Springer, Heidelberg (2005)
17. Object Management Group: Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) Specification, version 1.1, formal/ (January 01, 2011)
18. Taentzer, G.: AGG: A Tool Environment for Algebraic Graph Transformation. In: Münch, M., Nagl, M. (eds.) AGTIVE 1999. LNCS, vol. 1779, pp. 481–488. Springer, Heidelberg (2000)
19. W3C: XML Schema Part 0: Primer Second Edition (October 2004), http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/
20. W3C: XSL Transformations (XSLT) Version 2.0 (2007)
21. Wimmer, M., Kappel, G., Kusel, A., Retschitzegger, W., Schoenboeck, J., Schwinger, W.: Surviving the Heterogeneity Jungle with Composite Mapping Operators. In: Tratt, L., Gogolla, M. (eds.) ICMT 2010. LNCS, vol. 6142, pp. 260–275. Springer, Heidelberg (2010)
22. Wimmer, M., Strommer, M., Kargl, H., Kramler, G.: Towards model transformation generation by-example. In: Proceedings of the 40th Annual Hawaii International Conference on System Sciences, HICSS 2007, p. 285. IEEE Computer Society, USA (2007)

RESTGroups for Resilient Web Services

Tadeusz Kobus and Paweł T. Wojciechowski

Poznań University of Technology
Institute of Computing Science
60-965 Poznań, Poland
{Tadeusz.Kobus,Pawel.T.Wojciechowski}@cs.put.poznan.pl

Abstract. Service resilience, defined as the continued availability of a service despite failures and other negative changes in its environment, is vital in many systems. It is typically achieved by state machine replication using group communication middleware for coordination of service replicas. In this paper we focus on systems that represent critical data as Web resources identified by Uniform Resource Identifiers (URIs). The best examples of such systems are RESTful web services. We present *RESTGroups*: a group communication *front-end* for RESTful web services. Our system is based on Spread – a popular group communication toolkit. Contrary to Spread and other such toolkits, we represent group communication services as resources on the Web, addressed by URIs. In the paper, we describe the system architecture and the API.

Keywords: REST, web services, service resilience, group communication, replication.

1 Introduction

The *Web* can provide a common, language-independent platform for interoperable services that work together to create seamless and robust systems. However, we must ensure that each individual service is *resilient*, i.e. it is able to withstand unpredictable and difficult conditions, such as sudden and significant degradation of network latency or failure of dependant services. In our work, we focus on RESTful web services. *REpresentational State Transfer (REST)* [4,3] embraces a stateless client-server architecture, in which web services are viewed as resources identified by URIs. Clients that want to request these services access their particular representation by transferring application content using a small globally defined set of methods. The methods describe an action to be performed on a given resource (consequently, by the corresponding service). Typically REST uses HTTP [2] and its methods GET, PUT, POST, and DELETE.

A typical way of increasing service resilience is to replicate it. *Service replication* means deployment of a service on several server machines, each of which may fail independently, and coordination of client interactions with service replicas. Each service replica, called a *process*, starts in the same initial state and executes the same requests in the same order. Thus, the replicated service executes simultaneously on all machines. A client can use any single response to its

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 505–517, 2012.

request to the service. A replicated service is available continuously, tolerating crashes of individual server machines. If required, these machines can be located in geographically distant places, connected via a wide-area network.

A general model of such replication is called *replicated state machine* [17]. In this approach, each non-fault replica receives every request (the *agreement property*), and each non-fault replica processes the requests in the same relative order (the *order property*). The key abstractions required to obtain these two essential properties are offered by *group communication systems*. They provide reliable multicast transport protocols with a range of delivery options, e.g. causally- , fifo- and totally-ordered multicasts in a group of processes. The protocols are fully distributed, i.e. they do not depend on any central server and so there is no single point of failure. For the past 20+ years, many group communication systems have been implemented (e.g. JGroups [15] and Spread [18]; see also [9] for other references). Unfortunately, they have quite different APIs, which are language dependent and complex. Moreover, many of group communication systems are monolithic, so it is not possible to easily replace their protocols or add new features. Using these systems to implement resilient web services makes the code of the services neither easily reusable nor interoperable with other services, which is a counterexample to the openness of the Web. Moreover, none of the system that we know offers a REST/HTTP-based interface.

In [14], the authors discuss some examples of service replication middleware systems developed at the academia (e.g. WS-Replication [16]); however, none of these systems supports RESTful web services. The industry solutions for web service resilience are usually based on the SOAP-based WS-* standards that were not designed for service replication. The concrete implementations of service resilience are usually built on queueing or publish-subscribe systems. This approach does not benefit from the group communication protocols that have been optimized for state machine replication. When it comes to RESTful approaches to web service implementation, group communication solutions that offer all associated properties and guarantees are unknown to us. At the moment there is a lack of standards in the domain of group communication intended for the REST style. This provided motivation for our work described in this paper.

We introduce a group communication *front-end* for replication of RESTful web services – *RESTGroups*. A brief announcement of our work appeared in [7]. In this paper, we describe the system architecture and the API. Our system is based on Spread [18,1] – a popular group communication toolkit implementing protocols for reliable, ordered multicasts and group membership. Contrary to Spread and other such toolkits, we represent group communication services as resources on the Web, addressed by URIs. Thus, RESTful web services and their clients can use group communication services in the same style as they communicate among themselves. Moreover, since firewalls usually do not block the HTTP protocol, RESTGroups supports communication across firewalls. The system has been implemented and the distribution files are available [5].

2 RESTGroups Design and Service Replication

RESTGroups is a group communication *front-end* for RESTful web services. Our current implementation is based on Spread Toolkit. Spread [18,1] is a monolithic group communication system, consisting of a daemon program, client libraries, and a system monitor. RESTGroups represents group communication services provided by Spread, as resources on the Web, addressed by URIs. Spread's API consists of many functions with bindings available for several programming languages: C/C++, Java, Perl, Python, and Ruby. RESTGroups has a small but powerful API that consists of just four methods of the HTTP protocol: GET, POST, PUT, and DELETE. They can be used for detection of malfunctioning/crashed processes, reliable point-to-point transfer of messages, formation of processes into groups, the structure of which can change at runtime, and reliable message multicasting with a wide range of guarantees concerning delivery of messages to group members (e.g. causally-, fifo- and totally-ordered delivery).

2.1 System Design

A system built using RESTGroups consists of four types of communicating components: Web Service, Client, RESTGroups Server (RESTGr Server in short), and spreadd (which is a daemon of Spread Toolkit). Web Service is a user-defined RESTful web service. The RESTGr Server acts as a *proxy* between Web Service and group communication protocols that are implemented by spreadd. The communication between Client and Web Service, as well as between Web Service and RESTGr Server uses the REST/HTTP style. RESTGr Server and spreadd communicate using TCP and may or may not run on the same machine.

Group communication services (provided by Spread) are represented as Web resources identified by URIs. Instead of calling Spread methods, a user-defined Web Service invokes only four methods of the HTTP protocol (i.e. GET, PUT, POST or DELETE). Then, a suitable HTTP request, possibly containing an XML document, is sent to RESTGr Server that translates it into a group communication call to spreadd. A crash of RESTGr Server results in the disconnection of all Web Services that are using this server. They can establish connection with another RESTGr Server which is available within the same group. Next, we present an architecture of a system, in which the RESTGroups components will be replicated for resilience.

2.2 Service Replication

In Figure 1, we show an architecture of an example system in which RESTGroups has been used for replication of a RESTful web service. There are three service replicas (each one called *Web Service replica*) perceived by the clients as a single web service, represented as a large circle. Clients can issue REST/HTTP requests to any of them. Each Web Service replica connects to two RESTGr Servers using HTTP, so it can tolerate a crash of one server. Each RESTGr Server has its own

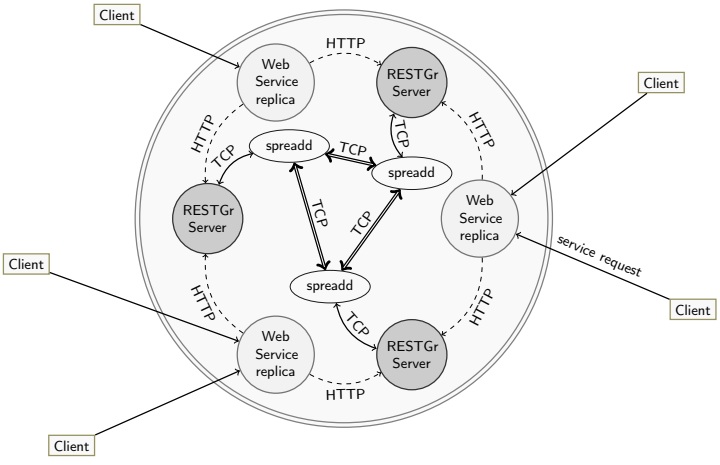


Fig. 1. Replication of a RESTful web service

spreadd so that partial failure of the Spread group communication system used as the back-end is also tolerated.

In general, replicating a web service to tolerate at most $\lfloor (n-1)/2 \rfloor$ machine crashes, requires the following steps:

- spawning n Spread daemons (spreadd) on n independent machines;
- spawning n RESTGroups servers on different machines; each server communicates only with one Spread daemon (usually located on the same machine);
- spawning n instances of the RESTful web service on different machines (in this case, they would run on the same machines as Spread daemons); each service replica can communicate with one or many RESTGroups servers.

The service developers can use the replicated state machine approach [17,20] to implement a resilient RESTful web service, as follows. After system start up, a group is created to which all Web Service replicas must join. A client can issue a request to any known replica which then forwards the request to the RESTGr Server that is alive; the latter broadcasts the request in the group. All client requests issued to (any replica of) the web service are delivered within the group totally ordered. Thus the requests will be processed by each replica in exactly the same order; the client will obtain only one reply to each request. We require the web service to be deterministic, so that all replicas will make transition to the same state in response to the same sequence of requests issued by clients. In the case of a replica crash, the clients may have to repeat its request to another Web Service replica after a timeout.

2.3 Statelessness

RESTGr Servers are almost *stateless* – they only store data that are necessary to maintain group communication sessions for the connected Web Service

replicas. Moreover, RESTGr Server does not have any representation in the group communication system that is the back-end of RESTGroups. However, unique client IDs generated by Spread are used by RESTGroups.

Various authors pointed out limitations of the REST architectural style. For example, Khare and Taylor [6] discussed some of the limitations and proposed extensions of REST, collectively called *ARRESTED*. They allow to model the properties required by distributed and decentralized systems. Similarly to them, we are not bound by the rules of the original model since REST cannot model group communication well (as the RESTGr Server is not 100% stateless). Therefore our goal was rather to design the REST-inspired interface to group communication, albeit sacrificing strict conformance to the original REST model.

3 RESTGroups API Calls

In this section, we explain the calls of RESTGroups API using a few simple examples. A complete description of the API is in the User Guide, available electronically [5]. The following methods of the HTTP protocol are used, where resources represent some group communication services or data structures (such as a mailbox):

- GET is used to perform a query on a resource, e.g. to retrieve messages from the mailbox (in a blocking or non-blocking manner);
- PUT is used to create a new resource, e.g. to extend a process group with a new process; the server responds with a status indicating success or failure;
- POST is used to update existing resources, e.g. to connect to the server on system start-up (this operation is executed only once) or to send/broadcast a new message;
- DELETE is used to remove a resource, e.g. to remove a process from a process group; in some cases, the update and delete actions may be performed with POST operations as well.

Consider the RESTGroups server located at `http://mydomain.com:8182` and a RESTGroups client (or client, in short), denoted `userA`, located at some other site. For example, `userA` could be the Web Service replica in Figure 1. Below we describe the following operations: connecting to the server, sending messages, and message retrieval.

3.1 Connecting to the Server

HTTP is a stateless protocol for client-server communication. In order to execute a given action by a server, a client initializes connection with the server and sends a request to it. The request contains all the information that are needed by the server to process the action. After processing the action, the server sends back a response message and the connection is closed. Therefore, using HTTP as a transport protocol in the group communication system does not seem natural.

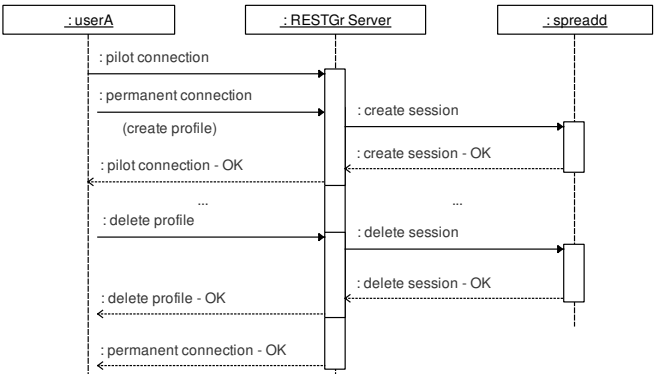


Fig. 2. Successfully connecting and disconnecting from the RESTGroups server

A permanent connection would be more useful, since it can allow the system to detect client's failure when the connection is broken.

Therefore, the connection with the RESTGroups server is accomplished using two requests to the server. The first one, called the *temporary* (or *pilot*) request, is used to ask the server to set up a resource which represents a new communication session. The session is created using the second request, called the *permanent* request. The server does not respond to this request, so the connection opened to process it remains open. Breaking of the latter connection is interpreted by the server as crash of the client. Both requests should be separated in time by no more than 5 seconds; the order of the requests is irrelevant. In Figure 2, we illustrate making a successful connection and disconnection with the RESTGroups server.

Connection with the RESTGr Server is identified by a unique identifier `pilotConnectionToken`, created with the use of random UUID numbers [19]. The UUID number created by a client is sent in the XML format in the bodies of both the pilot and permanent requests. A pilot request may look as follows:

```
POST http://mydomain.com:8182/groups/userA/pilotConnection

<?xml version="1.0" encoding="UTF-8"?>
<restgroups>
 <pilotConnectionToken>dec7b89c-1f08-447e-952f-9c441ec92e5c<</pilotConnectionToken>
</restgroups>
```

Processing of this request is suspended until a corresponding permanent request is received or a timeout occurs. The `schemes/profilesPilotMessage.xsd` file is used for validation of the temporary request's body.

A permanent request may contain information about client preferences, e.g. a request of discarding the group membership messages, as below.

```
POST http://mydomain.com:8182/groups/userA

<?xml version="1.0" encoding="UTF-8"?>
<restgroups>
 <pilotConnectionToken>dec7b89c-1f08-447e-952f-9c441ec92e5c</pilotConnectionToken>
 <groupMembership>false</groupMembership>
</restgroups>
```

The `schema/profileMessage.xsd` file is used for validation of the permanent request's body.

If a new session has been created successfully, the response message to the temporary request is returned with the `204 'Success No Content'` status. The response contains: `sessionID` – a session identification number, stored in the response 'cookie'; from now on, all requests to the RESTGroups server must include `sessionID`, which will allow the server to identify clients, and `identifier` – URI of the client's private group, stored in the response field that is used for identification; since the names of private groups must be unique across the whole group communication system, the `identifier` value can be different from the name of the client, which is specified in the pilot and permanent requests. For example, the following values could be received:

- `sessionID: d10b88e7-74f3-424a-b306-c47440a818d9`
- `identifier: http://mydomain.com:8182/groups/@userA@mydomain`

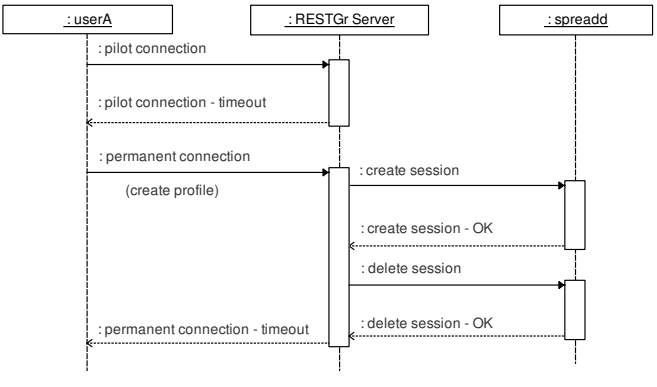


Fig. 3. Unsuccessful session creation due to a connection timeout

If connection with the RESTGroups server fails, suitable error messages are received, e.g. in response to the pilot or permanent request, HTTP's `408 'Request Timeout'` error can be received if one of the two requests has not been received in a predefined period of time (see Figure 3).

3.2 Sending Messages

There are two possible ways of sending messages to a group of users or to a single user, identified by the URI of the private group to which it belongs (only one user can belong to a given private group). The first way (see Figure 4) can be applied in every case; it uses a predefined resource `/multicast` and requires to specify (in the body of a message) the name of the message recipient, i.e. an identifier of a group or a user to whom the message will be sent. When using the second way, there is no need to specify the message recipient in the body of the message. However, each potential recipient of the message, i.e. a group or an individual user, must be represented by a resource, identified by URI. This approach is convenient if messages are addressed to a single user only.

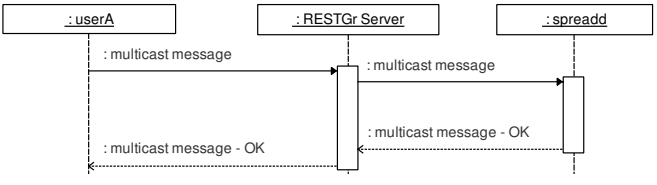


Fig. 4. Sending a message

Consider a user-defined group named `customGroup`. Sending a message to this group by referring to the `/multicast` resource, requires an XML document. The structure of this document is verified based on the `schemes/clientMessage.xsd` file which defines the proper XML schema. The following sections (or tags) of the structure must be defined: `guarantee` – the reliability and ordering guarantees of message delivery, `type` – a message type, `groups` – a list of addresses, and finally `data` – the message payload.

The following guarantees of message delivery are supported:

- `unreliable` – no guarantee of message delivery,
- `reliable` – reliable broadcast,
- `fifo` – fifo broadcast (first-in-first-out),
- `causal` – causal broadcast, consistent with Lamport's definition of causality,
- `safe` – total order broadcast,
- `agreed` – total order broadcast that is consistent with causal broadcast, i.e. messages are delivered to all recipients in the same order, and the order agrees with the causal relation between messages.

```
POST http://mydomain.com:8182/multicast

<?xml version="1.0" encoding="UTF-8"?>
<restgroups>
  <messages>
    <message type="regular">
      <guarantee>safe</guarantee>
      <type>0</type>
      <groups>
        <group>customGroup</group>
      </groups>
      <data>Sample message</data>
    </message>
  </messages>
</restgroups>
```

Using the second approach for sending a message to the `customGroup`, requires to specify an XML document. The structure of this document is verified using the `schemes/clientMessageSingleGroup.xsd` schema file.

The request should appear as below:

```
POST http://mydomain.com:8182/groups/customGroup/mailbox/safe

<?xml version="1.0" encoding="UTF-8"?>
<restgroups>
  <messages>
    <message type="regular">
      <type>0</type>
      <data>Sample message</data>
    </message>
  </messages>
</restgroups>
```

Note that the request's URI refers to a private mailbox located at the specified address. The last part of the URI defines the chosen guarantee of message delivery; this guarantee can take any of the six values described earlier.

Upon successful message sending, the RESTGr Server returns a response message with the `204 'Success No Content'` status code. In the case of an error, the server returns the HTTP error message, e.g. `400 'Client Bad Request'` – if the client with the `sessionID` identifier in the request's 'cookie' does not have an active RESTGroups session, or `503 'Service Unavailable'` – if an error occurs during the disconnection from the group communication system.

3.3 Reception of Messages

The RESTGroups system offers two types of mechanisms for reception of messages: *blocking* (synchronous) and *non-blocking* (asynchronous). A user can also

check if there are any unread messages waiting on the RESTGroups server without fetching them. Below we describe the blocking reception mechanism; the non-blocking reception mechanism has a similar syntax.

Performing the following `GET` request by a client is suspended until a new message (or messages) will be received by the client:

```
GET http://mydomain.com:8182/groups/@userA@mydomain/mailbox/blocking
```

A response to this request is an XML document which contains aggregated messages that have been sent (or broadcast) to the client; each message contains the names of the broadcast group and of the message sender. Messages are sent to a client as soon as they arrive to the RESTGroups server. The structure of responses in the case of non-blocking messages is similar, except that the "no messages" response can also be returned.

In order to stop receiving messages, the client should issue the `DELETE` request:

```
DELETE http://mydomain.com:8182/groups/@userA@mydomain/mailbox/blocking
```

4 Related Work

In this section, we describe related work on group communication support for web services. We begin from discussing the industry standards, followed by example research projects. This work is for SOAP-based web services only; we are not aware of similar work done for RESTful web services.

In SOAP-based web services, distributed processes communicate messages, typically wrapped in the XML format, using the *Simple Object Access Protocol (SOAP)*. Essentially, SOAP means sending remote procedure calls (RPC) through standard HTTP ports, using an XML envelope. REST emphasizes the element of using standardized URIs, and also giving importance to the HTTP verb used (i.e. GET, POST, PUT, or DELETE). Benefit of the RESTful interface is that requests and responses can be short – in contrary to SOAP that requires an XML wrapper around *every* request and response. On the other hand, SOAP can easier transport any attached files and has better tool support. Since group communication protocols exchange many control messages, shorter processing time of messages in REST means better performance of these protocols. However, we do not present evaluation results since our contribution is mainly the design of the group communication interface for RESTful web services, not a group communication system. In particular, we might use some other back-end system instead of Spread and obtain different performance.

WS-ReliableMessaging [13] is an OASIS standard describing the protocol for reliable unicast-only message communication using SOAP. The standard does not specify multicast (or broadcast) communication. It defines two components: *Remote Messaging Source (RMS)* on the sender side and *Remote Messaging Destination (RMD)* on the receiver side. These two components communicate using SOAP-messages. For this, a communication link is created between RMS and RMD, and all messages exchanged using this link are given a sequence number.

The programmer can choose among the following levels of *delivery assurances*, e.g. at-least-once, at-most-once, exactly-once and in-order. The above properties can provide building blocks for implementing group communication abstractions above the WS-ReliableMessaging. In RESTGroups, the unicast communication with analogous guarantees can be achieved by defining a group to which only the sender and the receiver belong, and using a POST method with the required semantics (unreliable, reliable, or fifo).

WS-BaseNotification [11] and *WS-BrokeredNotification* [12] are OASIS standards describing the protocols for one-to-many communication of SOAP messages. The communication is based on the *publish-subscribe* model. The system users can create *topics* of messages, to which the message recipients (or *consumers*) can subscribe. The standard allows to have a separate *subscriber* that subscribes a number of consumers to a given topic. When a message sender (or a *publisher*) publishes a message on a given topic, the message is propagated to all consumers who subscribed (or have been subscribed) for that topic and their subscription remains active. The *WS-BrokeredNotification* standard also introduces a *broker*, who is responsible for recording published messages of a given topic, and resending them to all *consumers* that have subscribed to that topic. However, the WS-BaseNotification and WS-BrokeredNotification standards focus on the information exchange protocol only, leaving the issues of reliable communication to "a delivery mechanism for transmission", where transmission properties are unspecified: "depending on the actual delivery mechanism, this transmission might be reliable or might be done on a best-effort basis" [11].

In commercial applications, *message queueing* systems are often used as the mechanism for reliable message transmission, including the one-to-many interaction. They provide an asynchronous communication protocol between distributed, loosely coupled processes. The sender and a receiver of a message do not need to be accessible at the same time, for the message to be delivered (by default, in the FIFO order). When a receiver of a message is not accessible, the message will be stored in a queue until it can be delivered. This property is called *durability*. Many implementations of queueing systems support resilience to system failures. This is usually done by implementing a message property called *persistence*. After receiving a persistent message, a queueing system sends a message receipt acknowledgment but only after the message had been stored in non-volatile memory. The persistence property guarantees that a message characterized by this property will never be lost, even in the case of runtime failures. Many queueing systems have been developed for the last few decades. This resulted in a variety of protocols and APIs (including REST/HTTP). However, many consider the *Java Message Service (JMS)* [10] to be a *de facto* standard of a queueing system. JMS assumes both the one-to-one and one-to-many mode of communication, where the latter uses the publish-subscribe model. In [8], the authors show that it is possible to build a group communication system based on JMS, offering a notion of a group, a membership service and a total-order broadcast. However, such approach adds additional layers of abstraction when compared to RESTGroups. On the contrary, the group communication protocols designed for the replicated state machine are more efficient.

The closest work to ours is *WS-Multicast* [16] that has been designed as a broadcasting service for SOAP-based web services. The service is built on the JGroups group communication system [15]. It uses its own transport layer module for message communication based on SOAP. A WSDL interface has been defined, making WS-Multicast a web service itself. Since WS-Multicast only replaces the transport layer of the JGroups system, leaving the rest of the protocol stack unchanged, all the assurances offered by JGroups remain in place. Nonetheless, as was noted, the use of SOAP involves sizable cost stemming from the character of this protocol. Thus, in the final version of the proposed service, parsing XML data was being avoided.

5 Conclusion

In this paper, we demonstrated that group communication middleware, such as Spread, can be easily extended to support RESTful web services. RESTGroups wraps functionality of group communication middleware and exposes it through a uniform interface based on the HTTP protocol. We have discussed an example application of our system – replication of RESTful web services. We also emphasized that systems like RESTGroups cannot be 100% RESTful since some REST principles, such as client-server stateless interaction, cannot be captured in this type of application.

Acknowledgments. This work has been partially supported by the Polish Ministry of Science and Higher Education within the European Regional Development Fund, Grant No. POIG.01.03.01-00-008/08.

References

1. Amir, Y., Stanton, J.: The Spread wide area group communication system. Technical Report CNDS-98-4, Dep. of CS, Johns Hopkins Univ. (1998)
2. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1. Internet Engineering Task Force (1999)
3. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine (2000)
4. Fielding, R.T., Taylor, R.N.: Principled design of the modern Web architecture. ACM TOIT 2(2), 115–150 (2002)
5. IT-SOA. RESTGroups (2011), http://www.it-soa.pl/restgroups
6. Khare, R., Taylor, R.N.: Extending the Representational State Transfer (REST) architectural style for decentralized systems. In: Proc. ICSE 2004 (2004)
7. Kobus, T., Wojciechowski, P.T.: A 90% RESTful group communication service. In: Proc. of the DCDP Workshop 2010 (2010)
8. Kupšys, A., Pleisch, S., Schiper, A., Wiesmann, M.: Towards JMS compliant group communication-a semantic mapping. In: Proc. NCA 2004 (2004)
9. Mena, S., Schiper, A., Wojciechowski, P.T.: A Step Towards a New Generation of Group Communication Systems. In: Endler, M., Schmidt, D.C. (eds.) Middleware 2003. LNCS, vol. 2672, Springer, Heidelberg (2003)

10. Sun Microsys. Java Message Service (2009), http://java.sun.com/products/jms/
11. OASIS. Web Services Base Notification 1.3 (2006)
12. OASIS. Web Services Brokered Notification 1.3 (2006)
13. OASIS. Web Services Reliable Messaging 1.1 (2007)
14. Osrael, J., Froihofer, L., Goeschka, K.M.: What service replication middleware can learn from object replication middleware. In: Proc. of MW4SOC: the 1st Workshop on Middleware for Service Oriented Systems (2006)
15. Red Hat. The JGroups toolkit (2009), http://www.jgroups.org/
16. Salas, J., Pérez-Sorrosal, F., Patiño-Martínez, M., Jiménez-Peris, R.: WS-Replication: A framework for highly available Web services. In: Proc. of WWW 2006 (2006)
17. Schneider, F.B.: Implementing fault-tolerant services using the state machine approach: A tutorial. ACM CSUR 22(4), 299–319 (1990)
18. Spread Concepts LLC. The Spread toolkit (2006), http://www.spread.org/
19. The Internet Society. A Universally Unique IDentifier (UUID) URN Namespace (2005), http://www.ietf.org/rfc/rfc4122.txt
20. Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., Alonso, G.: Understanding replication in databases and distributed systems. In: Proc. of ICDCS 2000 (2000)

Leveraging Microblogs for Resource Ranking

Tomáš Majer and Marián Šimko

Faculty of Informatics and Information Technologies, Slovak University of Technology, Ilkovičova 3, 842 16, Bratislava 4
tomasmajer@gmail.com, simko@fiit.stuba.sk

Abstract. In order to compute page rankings, search algorithms primarily utilize information related to page content and link structure. Microblog as a phenomenon of today provides additional, potentially relevant, information – short messages often containing hypertext links to web resources. Such source is particularly valuable when considering a temporal aspect of information, which is being published every second. In this paper we present a method for resource ranking based on Twitter data structure processing. We apply various graph algorithms leveraging the notion of a node centrality in order to deduce microblog-based resource ranking. Our method ranks a microblog user based on his followers count with respect to a number of (re)posts and reflects it into resource ranking. The evaluation of the method showed that micro-based resource ranking *a*) can not be substituted by a common form of an explicit user rating, and *b*) has the great potential for search improvement.

Keywords: microblog, Twitter, resource ranking, web search.

1 Introduction

Nowadays, people benefit from the Internet and its most important service – the Web, which no longer consists only of static pages that people browse and search information in. User generated content has become more important than ever, what caused that a new medium for publishing short posts has emerged – microblog.

Microblog is a lightweight form of traditional blog, where users publish only brief reports. This phenomenon has become popular mainly thanks to microblog Twitter. Therefore, the most common name for a user post is "tweet". Microblog introduced a completely new form of communication. Users became able to share their actual feelings, experiences or opinions. Their posts are often related (and explicitly linked) to entities, which typically represent personalities, products or events. By aggregating different posts on various topics coming from a plethora of users and various platforms, microblog became a valuable source of data.

It is possible to monitor current users' opinions all around the world on microblog. Different world events can be tracked [4] such as elections, an introduction of a new product to the market, or revolutions. Typically it is a very current and extensive user content. Such content can be processed and utilized in order to improve access to information, e.g. by improving search. Currently the search

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 518–529, 2012.

takes into consideration criteria, where the most important are the content and links structure. There are also other areas such as product review or rating, where ranking can be utilized for filtering and sorting. However, those can be subsequently (and unfairly) amended by owners of e-shops. Thinking of Twitter as of a separate service, it contains un-moderated user data, which typically are not as biased and are more objective.

In this paper we propose a method for leveraging microblogs for ranking resources (typically web pages). The method is based on Twitter graph analysis that results into resource ranking computation. The method is domain independent as it processes a graph representation abstracting of the content itself.

The rest of the paper is structured as follows. Section 2 summarizes work related to microblogs and resource ranking. In section 3 we introduce a novel ranking measure called TweetRank. The evaluation and a dataset acquired for experiments are described in section 4. In section 5 we conclude our work.

2 Related Work

There are many areas where microblog mining is beneficial. By textual processing of user posts we can gather specific keywords for a particular user [22], or use text classification methods to classify posts and users to specific topics [15]. It is possible to use methods and techniques of opinion mining to find user opinions [13] such as polarity of tweet.

Much work has been done in the area of microblog search and microblog posts ranking [12,20]. Also the well known search engines like Google include tweets in their search results, based on full text search. Twitter itself contains a custom search engine allowing to search effectively through full text or tags. Twitter can be used to find relevant recent resources over the Web and to identify new and current trends in the world [5]. However, little is known about how Twitter posts influence the relevancy of resources contained in the posts and how this information can be utilized outside microblog. To our best knowledge, we are not aware of any method directly using microblog posts for deriving resource ranking and improving search in resources contained within those posts.

Viewing Twitter as a huge network, it is natural to apply graph algorithms to analyze its structure. General graph ranking algorithms such as PageRank, HITS or NodeRanking can be applied to a graph representing microblog structure. PageRank identifies nodes that have the greatest relevance in the graph. It can be used to rank users by analyzing relationships [3]. NodeRanking also builds on the random surfer model and, in addition to PageRank, considers weights of links and the damping factor is not a constant but a node variable [14].

Some ranking algorithms have been already adopted for microblogs. TwitterRank is based on PageRank [21]. Besides link structure, it considers topical similarity between users. TunkRank is basic ranking method for social networks, which are based on microblog principles [7]. It is directly correlated with an expected

number of readers of a twitterer and inversely correlated with the number of users he follows. TunkRank algorithm can be used to identify influential users.

3 Microblog-Based Resource Ranking

Twitter is a service combining elements of both social networking and microblogging with certain specifics [8]. While user profiles in social networks are connected bi-directionally, connections on microblog have only one-way orientation. User may follow other users in order to track their posts, but it does not necessarily mean that also reverse links exist, i.e., a followed user may choose not to follow ones that follow him. Twitter data analysis revealed that 80.5 % of users is followed by 80 % of their followers [21]. It was also showed that posts that users publish mostly depend on the number of users who follow them [8]. In contrast with traditional blogs, post comments cannot be assigned directly to a post. However, a new post linking to existing posts can be created, referred to as *retweet*. Users have the option to setup visibility for each post and restrict the "audience" of his tweets.

Twitter is mostly used by users who actively use the web. It is therefore natural that a 22 % of their tweets contain links to websites [2]. Due to the limited length of the contributions many of the services that shorten the web address are used, called URL-shortening services.

The aforementioned characteristics can be used to construct a graph representation (see Fig. 1). Twitter graph contains three types of entities: users, tweets and resources. There are some tweets on Twitter, which are not connected to resources (typically web pages). The edges between users represent a unidirectional followership relation. Users are also connected with tweets they post. Tweets can be linked to users, another tweets and pages. Relations between tweets represent a re-tweet of a "parent" tweet.

3.1 TweetRank Computation

We believe the structure of microblog posts (represented by the graph) can be leveraged to determine ranking of resources referenced in microblog tweets.

We propose a method for computing a resource rank, which we call *TweetRank*. Basically, TweetRank is derived from tweet topology. The method is based on the existing graph analysis algorithms, while extending them with microblog specific features. The principle of the method lies in user ranking estimation and propagating such rank via graph relationships into web resources.

In order to obtain resource ranking, we first compute user ranking based on modified TunkRank algorithm [6]. In contrast with TunkRank, we do not use static constant transition probability, but calculate dynamic coefficient for each user based on a number of his followers and tweets. For a user rank calculation we proposed the following equation:

$$UserRank(u) = \sum_{f \in followers(u)} \frac{1 + \frac{|followers(u)|}{|tweets(u)|} * UserRank(f)}{|followers(f)|} \qquad (1)$$

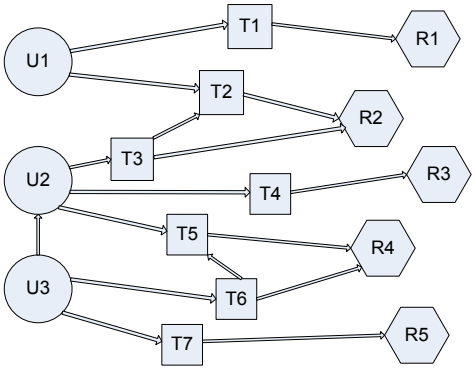


Fig. 1. Basic Twitter graph representation. Users are represented by circle on the left hand side (labeled Ux). Squares in the middle represent tweets (labeled Ty). Hexagons on the right hand side represent resources (labeled Rz).

where $UserRank(u)$ represents rank of a user u, $followers(u)$ represents set of followers of user u and $tweets(u)$ is set of tweets of user u.

After computing a rank for each user, we use the $UserRank$ to rank user tweets. Users with higher $UserRank$ post tweets that are more important and, as a result, resources they share in those tweets are more relevant. The computation is iterative: $UserRank$ is based on $UserRank$ of all user followers. The more high-ranked followers a user has, the higher $UserRank$ of this user is. In the beginning we assign same $UserRank$ value to every user. In order to obtain final $UserRank$, we perform more $UserRank$ computations until changes in user ranks do not exceed a specified (very low) threshold.

For calculating a relevance of a tweet, we use the following computation:

$$TR(t) = TweetRelevance(t) = \frac{UserRank(Author(r))}{|tweets(Author(t))|} \tag{2}$$

where $TR(t) = TweetRelevance(t)$ is relevance of tweet t, $Author(t)$ is author of tweet t. $TweetRelevance$ represents relative rank of every tweet. It is based on $UserRank$ which is adjusted according to the number of tweets of the author.

After obtaining a rank for each tweet, we compute ranks of tweets, which link to resources, and derive resource rankings. When a tweet is re-tweeted, we increase a rank for a resource, as we explained earlier. Finally, $TweetRank$ for a resource r is defined as an aggregation of ranks of all tweets that point to the resource r:

$$TweetRank(r) = \sum_{t \in tweets^{URL}} \Big(TR(t) + \sum_{rt \in retweets(t)} TR(t) * TR(rt) \Big) \tag{3}$$

where $TweetRank(r)$ is TweetRank of a resource r, $tweets^{URL}$ is set of tweets containing URL and $retweets(t)$ is set of retweets of tweet t.

For example, consider graph on Fig 1. Assuming user rankings $UserRank(U1)$ =6, $UserRank(U2)$=2, $UserRank(U3) = 10$ we calculate TweetRank of resource R4 as follows: $TweetRank(R4) = TR(T5)+TR(T5)*TR(T6)+TR(T6) = 2/3+ 2/3 * 10/2 + 10/2 = 8.9$ (normalisation omitted). Similarly, $TweetRank(R2) =$ 5.6. The higher user ranking reflects into the higher ranking of a resource.

The proposed method is a novel graph analysis method for computing rankings for microblog-based networks such as Twitter. For computing resource ranks it considers microblog specific concepts: unidirectional relation of followership and ability to re-tweet already tweeted post.

4 Evaluation

In order to evaluate the proposed method, we conducted two experiments. We collected Twitter dataset with tweets containing links and users relations. In the first experiment we rank YouTube videos and compare results based on TweetRank and user rank from YouTube.

In the second experiment we evaluate our ranking method by incorporating it in searching. We present an approach where resources are sorted based on combination of fulltext search with microblog-based rating.

4.1 Dataset

We collected and joined two datasets to be used in experiments. First, we prepared a dataset of Twitter data by using 140kit service[1]. The dataset contains tweets and user profiles from 18th to 24th September 2009. There are 1,997,446 tweets and 367,824 user profiles in the dataset. User profiles contain basic user data such as the count of his followers or the count of his total tweets. Dataset contains tweets in 6 different languages. 85 % of tweets are in English. To calculate ratings of users it is necessary to create connections between the users (for the calculation of user rank we need to know user rank of all of his followers). However, there were no such relations between users in obtained dataset. In order to calculate the ratings of users, Twitter API can be used directly for downloading a list of followers for a particular user. Unfortunately, this process is slow and Twitter limits the number of requests (350 per hour). Therefore, we used additional dataset, which contains information about user relations [10]. The dataset contains only links between users based on followers relationship and was created for three months from early June 2009 until the end of September 2009. This dataset contains users, who are part of the first dataset. The dataset contains 1,468,365,182 connections between 40,103,281 users. The dataset together with a full description can be downloaded from the website of the work[2].

There are 1,150,168 unique web links in the dataset. By using our method for resource ranking, we computed TweetRank for each resource determined by a

[1] http://140kit.com

[2] http://an.kaist.ac.kr/traces/WWW2010.html

link in the dataset. The distribution of the number of resources having particular TweetRank value is depicted in Fig. 2. It follows a power-law distribution.

Further analysis of the dataset revealed that as much as 3 % of links points to YouTube videos. YouTube videos also contain explicit rating given by users. In the first experiment we assess to what extent TweetRank relates with rating coming from the YouTube site.

4.2 Comparison with YouTube User Rank

In addition to watching a video on YouTube, users can also comment the video, they respond to comments of others and they vote. A vote can be positive or negative. For the purpose of the experiment we downloaded the user votes of 605 videos. We transformed each vote into a rating calculated as a difference between positive and negative votes and normalized it according to TweetRank rankings. After that, we compared user rating to TweetRank ranking. We assumed that the ranking of videos using TweetRank will be similar to that of the votes on YouTube, as both ratings come from the crowd. However, we did not find a correlation between both rankings (see Fig. 3).

The performed experiment showed that the evaluation of video-based ratings of users on YouTube is not correlated with the TweetRank (correlation coefficient $r = 0.02$). Although both approaches are basically based on rating of users, results differ. We see the main difference in results in unequal length of ranking time period. Videos on YouTube are available from the moment of their publication until the author decides to remove the video (which occurs very rarely). Long period YouTube rating is based on continuous rating through years, while TweetRank ranks are computed based on user posts collected during a period in September 2009 as described earlier. As we were not able to asses TweetRank rankings, we decided to compare the rankings with user rankings in the period closer to that from dataset in terms of both time distance and time length.

We created an application for collecting user ratings by enabling them to see video and enabling them to enter a rank. The application offered to visitors 5

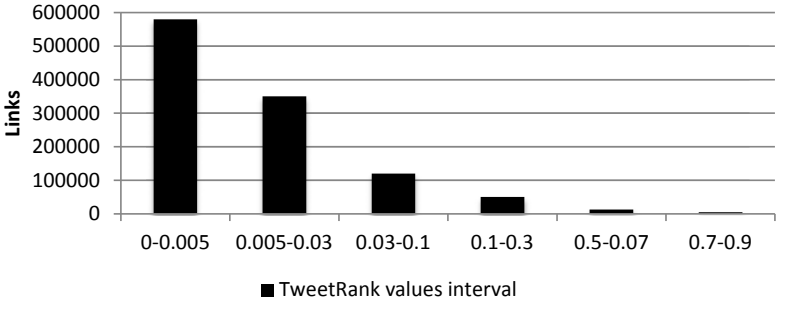


Fig. 2. Distribution of computed TweetRank values

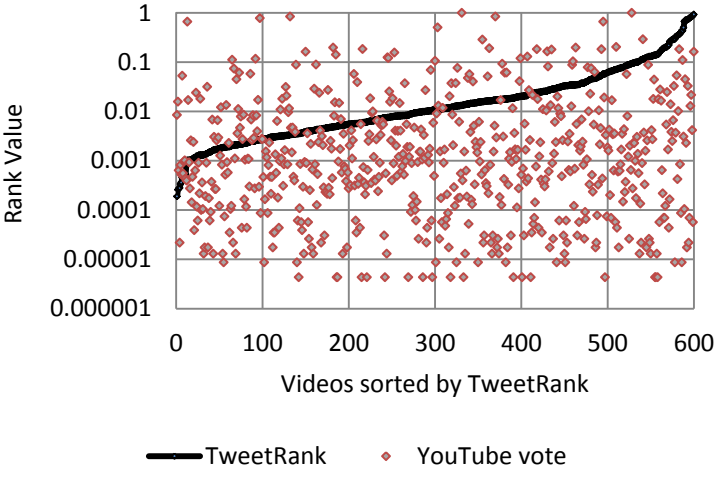


Fig. 3. Comparison of YouTube user rank and TweetRank

YouTube videos coming from the acquired dataset. The visitors were instructed to rate the videos on a scale from 1 to 5, where 1 stood for best and 5 for the worst rate. Users were given simple guidelines for evaluating videos: they should rate videos according to how they *like* them. There were no other criteria suggested they should consider. 70 different users participated in the experiment. They were non-technically oriented, ordinary web users and thus, in the most cases, also users of YouTube. We collected 680 responses together. The participants were approached through social networks Facebook and microblog Twitter.

Based on user voting, we created two comparisons. We sorted videos based on average video rating computed according the following formula:

$$AVR(v) = \frac{\sum_{H(v)} Rating(v)}{|H(v)|} \tag{4}$$

where $AVR(v)$ represents video rating computed as an average rating from a set of all video ratings $H(v)$ of video v.

The results of voting showed that videos got rather lower user rank values in general (see Table 1). When compared to the TweetRank results (Fig. 2), we see similar characteristics: in both cases many videos got low rates and only a few videos were assigned high rates. The user voting results were compared with TweetRank rankings. We computed the Kendall rank correlation coefficient τ in order to measure the association between both results. We obtained τ = -0.12519 (the resulting value is negative because user rank evaluation method has been reversed). We also calculated the correlation coefficient $r = 0.387$.

Table 1. Number of videos with a particular user rank falling into rank interval

Average rank in interval	Number of videos
⟨1.0, 1.5⟩	40
(1.5, 2.5⟩	45
(2.5, 3.5⟩	92
(3.5, 4.5⟩	106
(4.5, 5.0⟩	107

As each user had to rate five videos at once in the created application, he created own subjective ordering of the videos. Average difference between two adjacent videos in ordering by TweetRank is 0.0025. We also considered a deviation of rating in order to observe the change of two orderings (5-tuplets) match ratio. One ordering was based on user ranks and the other was based on Tweet-Rank (of the same videos). Deviation we define as a number, which we need to add to any TweetRank of video from 5-tuplet of videos in order to make changed TweetRank ordering match the 5-tuplet of videos composed by a user (Fig. 4).

We see that finding a match in 5-tuple of user rating and TweetRank is difficult, because disagreement even in one place makes a comparison of orderings unsuccessful. Therefore, we also made another comparison, where we compared pairs of videos rather than 5-tuples. Similarly, we considered deviation in order to observe match ratio change (Fig. 5).

The number of matched pairs is far higher than in the case when comparing 5-tuples. Results show that there is more then 60 % of matches without considering deviation. Although a match ratio increases with a growing deviation, there is still much difference in agreement on orderings. We believe the explicit user rating lacks information (let us name it a context) that was induced from microblog. Microblog topology and relationships between all entities represents potentially

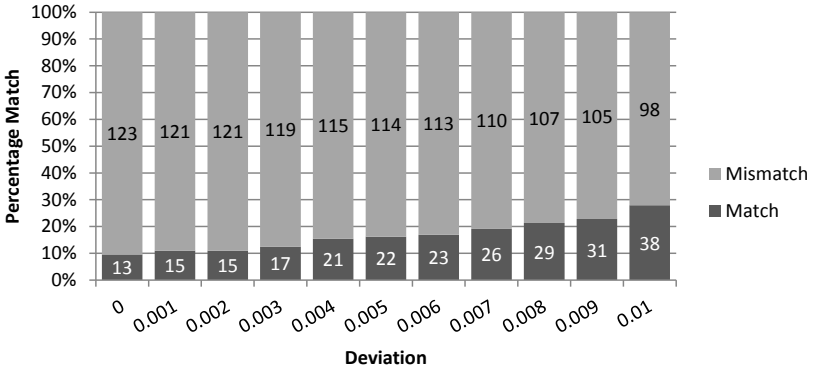


Fig. 4. Comparison of identical sort of five videos with different deviation

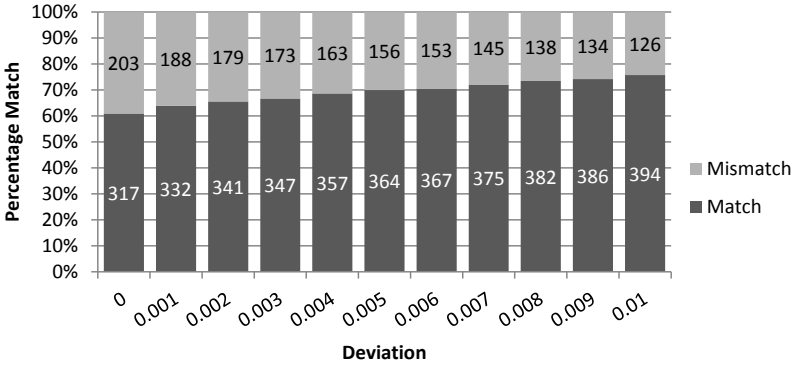


Fig. 5. Comparison of identical pairs with different deviation

valuable information with regards to resources contained in blog posts. In order to further analyze obtained TweetRank rankings, we conducted an experiment related to search.

4.3 Sorting Search Results with TweetRank

In the experiment we used TweetRank for sorting search results. For the experiment we selected a part of the acquired dataset, which contained the randomly chosen 20,000 unique links to web resources. We indexed the resources using the SOLR search platform, which utilizes the core Apache Lucene search[3]. We extended document representation in order to incorporate calculated TweetRank, allowing us to sort search results by that rank.

We proposed a hybrid approach for scoring computation combining internal ranking of SOLR search engine and TweetRank. As a result, search yields relevant documents (containing keywords from a query), which are sorted by TweetRank. Top-k documents are selected for presentation on a result page.

We performed search with 20 randomly selected queries from well understood domain and manually evaluated the results. In order to demonstrate the results we have selected the keyword "apple". Top-5 documents ranked by both Lucene original ranking and our approach utilizing TweetRank are shown in Table 2. We see from the example that the hybrid approach yields ordering, which prefers pages containing information about releasing new products. This kind of pages also ranked better with other queries we tried. Such behavior of rank originates in the nature of microblogs, where people often react to world or local news [5]. However, an important observation is that TweetRank-based ordering does not match a chronological rank of resources, i.e., publication time of information on a page. This leads us to conclusion that microblog-based ranking is especially useful when ranking a set of resource created within a specific time window, as it can not be replaced by temporal metatada of resources itself (creation time, change time, etc.).

[3] http://lucene.apache.org/java/2_4_0/scoring.html

Table 2. Top pages in search result by SOLR index and by TweetRank

#	Sorted by SOLR	Sorted by TweetRank
1	iPhone 4 Available in China on Sep. 25	Adobe's Premiere Elements now available ...
2	Bad Apple!! (HQ, Download, ...	iPhone 4 Available in China on Sep. 25
3	Apple Pushing the Art of iPhontography	VLC for iPad is finally out on the App ...
4	eBay - New & used electronics ...	Can Apple's Time Machine old ...
5	eBay - New & used electronics ...	Cloud Computing – Look forward ...

5 Conclusions

The amount and nature of non-moderated user generated data emerging from microblog made it a potentially relevant and powerful data source that can be utilized for a variety of tasks. We believe that microblog data mining can improve approaches to domain modeling [16,17] and user modeling [1,9] potentially resulting in improved web search [18] or personalization [19,11]. It is important to note that microblog has certain restrictions, typically related to a limited size of a post or too specific language, which make microblog content processing more difficult. However, its popularity and spread result into increased number of posts every day. Nearly one quarter of posts contains URL to a web page [2]. Increasing popularity of microblog also increases a number of microblog posts, which can be utilized for web search improvement. This makes microblog data extremely perspective. Not only for search improvement at the Web scale, but also in domains, where microblog emerges as a tool for instant discussions among communities of the specialized interest.

In this paper we proposed a novel method for ranking resources referenced by microblog users in their posts. We particularly focused on microblog Twitter as it is both public and widely adopted. The principle of our approach lies in ranking microblog users and reflecting this rank in resources present in posts they publish. We utilize user ranking method TunkRank and extend it by considering additional microblog specific features like re-tweet, which improves user ranking computation in relation to posts a user publishes.

In order to evaluate the proposed method for resources ranking we conducted two experiments. The first experiment we performed in the domain of user videos, where we compared ratings obtained from YouTube website with ranking obtained by our method. We found no correlations in both measures, what can be explained by temporal characteristics of underlying data: while microblog-based ratings were created based on one month sample of tweets, original user voting at YouTube site is present for years. Therefore, we created own application for collecting a different form of explicit user rating: relative ordering. The final user ratings were compared with the rank obtained from microblog Twitter using our method. We were able to see similarities between the measures, but there still

was not sufficient number of agreements on rankings. We interpret this as a presence of additional potentially valuable information originating in microblog topology that can not be reflected in explicit user ratings.

In the second experiment we used TweetRank for sorting search results. We combined document score computation by considering both a traditional statistical measure and TweetRank obtained by the method we proposed. By performing a set of search scenarios we assessed how the search results were reordered. Results showed that combined scoring computation prefers pages containing relatively new information. Such behavior of rank originates in the nature of microblogs, where people often react instantly. On the other hand, TweetRank-based ordering does not match a chronological rank of resources, i.e., resources in top-k search results are not new in terms of time only, but also sorted with regards of user attitudes expressed through microblog posts they write.

In conclusion, we consider results of our research very promising and perspective. As a part of a future work we plan to conduct an experiment in a real world setting by incorporating users, who will assess obtained search results. Links in microblog posts are typically accompanied with user comments, which can be considered as resource annotations. Our long term goal is to extend our method with microblog posts content analysis in order to further improve obtained resource rankings and possibly combine it with semantic search approaches [18].

Acknowledgments. This work was partially supported by the grants VG1/ 0675/ 11/2011-2014, KEGA 028-025STU-4/2010, APVV-0208-10 and it is the partial result of the Research & Development Operational Programme for the project Research of methods for acquisition, analysis and personalized conveying of information and knowledge, ITMS 26240220039, co-funded by the ERDF.

References

1. Barla, M., Bieliková, M.: Ordinary Web Pages as a Source for Metadata Acquisition for Open Corpus User Modeling. In: Proc. of WWW/Internet, pp. 227–233. IADIS Press (2010)
2. Boyd, D.M., Golder, S., Lotan, G.: Tweet, Tweet, Retweet: Conversational Aspects of Retweeting on Twitter. In: 43rd Hawaii International Conference on System Sciences, pp. 1–10. IEEE (2010)
3. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. In: Proc. of the 7th Int. World Wide Web Conf. (1998)
4. Diakopoulos, N.A., Shamma, D.A.: Characterizing Debate Performance via Aggregated Twitter Sentiment. In: Proc. of the 28th Int. Conf. on Human Factors in Computing Systems, pp. 1195–1198. ACM (2010)
5. Dong, A.: Time is of the essence: improving recency ranking using Twitter data. In: Proc. of the 19th Int. Conf. on World Wide Web, pp. 331–340. ACM (2010)
6. Gayo-Avello, D., Brenes, D.J.: Overcoming Spammers in Twitter: A Tale of Five Algorithms. ir.ii.uam.es, pp. 41–52 (2010)
7. Gayo-Avello, D.: Nepotistic Relationships in Twitter and their Impact on Rank Prestige Algorithms. In: Arxiv preprint, arXiv:1004.0816 (2010)

8. Huberman, B.A., Romero, D.M.: Social networks that matter: Twitter under the microscope. In: Arxiv preprint, arXiv:0812.1045v1 (2009)
9. Kramár, T., Barla, M., Bieliková, M.: PeWeProxy: A Platform for Ubiquitous Personalization of the "Wild" Web. In: UMAP 2011: Adjunct Proc. of the 19th Int. Conf. on User Modeling, Adaptation and Personalization. Demo., pp. 7–9 (2011)
10. Kwak, H., Lee, C., Park, H.: What is Twitter, a Social Network or a News Media? In: Proceedings of the 19th International Conference on World Wide Web, Raleigh, pp. 591–600. ACM (2010)
11. Labaj, M.: Information Sciences and Technologies Bulletin of the ACM Slovakia. Special Section on Student Research in Informatics and Information Technologies 3(2), 76–78 (2011)
12. Nagmoti, R., Teredesai, A., De Cock, M.: Ranking Approaches for Microblog Search. In: Proc. of the 2010 IEEE/WIC/ACM Int. Conf. on Web Intelligence and Intelligent Agent Technology, vol. 01, pp. 153–157. IEEE Computer Society, Washington, DC (2010)
13. Pandey, V., Iyer, C.: Sentiment analysis of microblogs (2009), http://www.stanford.edu/class/cs229/proj2009/PandeyIyer.pdf (accessed October 05, 2011)
14. Pujol, J.M., Sangesa, R., Delgado, J.: Extracting Reputation in Multi Agent Systems by Means of Social Network Topology. In: Proc. of the First Int.l Joint Conf. Autonomous Agents and Multiagent Systems, pp. 467–474 (2002)
15. Ramage, D., Dumais, S., Liebling, D.: Characterizing Microblogs with Topic Models. In: Proc. of Int. AAAI Conf. on Weblogs and Social Media, pp. 130–137. AAAI Press (2010)
16. Šimko, J., Tvarožek, M., Bieliková, M.: Little Search Game: Term Network Acquisition via a Human Computation Game. In: HT 2011: Proc. of the 22nd ACM Conf. on Hypertext and Hypermedia, pp. 57–61. ACM, New York (2011)
17. Šimko, J.: Augmenting Human Computed Lightweight Semantics. Information Sciences and Technologies Bulletin of the ACM Slovakia, Special Section on Student Research in Informatics and Information Technologies 3(2), 116–118 (2011)
18. Šimko, M., Bieliková, M.: Improving Search Results with Lightweight Semantic Search. In: Grobelnik, M., Mika, P., Douc, T.T., Wang, H. (eds.) Proc. of the Workshop on Semantic Search, SemSearch 2009 at the 18th Int. World Wide Web Conference, WWW 2009, Madrid, Spain. CEUR, vol. 491, pp. 53–54 (2009)
19. Šimko, M., Barla, M., Bieliková, M.: ALEF: A Framework for Adaptive Web-Based Learning 2.0. In: Reynolds, N., Turcsányi-Szabó, M. (eds.) KCKS 2010. IFIP AICT, vol. 324, pp. 367–378. Springer, Heidelberg (2010)
20. Teevan, J., Ramage, D., Morris, M.R.: TwitterSearch: a comparison of microblog search and web search. In: Proc. of the Fourth ACM Int. Conf. on Web Search and Data Mining, WSDM 2011, pp. 35–44. ACM, New York (2011)
21. Weng, J., Lim, E.P., Jiang, J., He, Q.: TwitterRank: Finding Topic-sensitive Influential Twitterers. In: Proc. of the Third ACM Int. Conf. on Web Search and Data Mining, pp. 261–270. ACM (2010)
22. Wu, W., Zhang, B., Ostendorf, M.: Automatic generation of personalized annotation tags for Twitter users. In: Proc. HLT 2010 Human Language Technologies: The 2010 Annual Conf. of the North American Chapter of the Association for Computational Linguistics, pp. 689–692. ACM (2010)

Inner Architecture of a Social Networking System

Jaroslav Škrabálek, Petr Kunc, and Tomáš Pitner

Lab Software Architectures and Information Systems,
Faculty of Informatics, Masaryk University, Brno, Czech Republic
{xskraba1,xkunc7,tomp}@fi.muni.cz

Abstract. Social networks, their increasing popularity reaching hundreds of million users, demand advance software architecture. Countless requests per second necessitate flexible and utmost efficiency and high performance. This article is focused on development of such a web-based service offering social functionality to end users, but from the technology point of view represents state-of-the-art in current usage of the latest technologies. Those technologies mentioned further are often used for the first time in such a complex project. High volume data distribution is handled by Apache Hadoop[1] framework together with Hadoop Distributed File System (HDFS) and MapReduce. Therewithal, non-relational distributed database HBase and Memcached tool ensures scalability and high throughput helping with often accessed information. Inner architecture of the social subsystem has been implemented within three-layer structure (services/data access/transmission). Social subsystem among others deals with one-way (unsymmetrical) relationship generation or cancellation between users but events either. Particular system entities are allowed to add comments, follow or "like" others. In the end, testing phase, deployment and practical utilization (although the resulted solution is completely independent) is demonstrated on practical example of case study *Takeplace* – complex tool for event management.

1 Introduction

Web 2.0 [8] meant the revolution of creating and using webpages. The content was not anymore created by the administrator but by the users. A webpage turned to be just an interface.

Probably the most popular Web 2.0 applications are social network services that allow creating user accounts, managing relations with other users, creating groups of interest, communicating and sharing information. For example *Facebook* has more than seven hundred million users in July 2011 [7].

This paper focuses on design and architecture of a server-side social network service (based on analysis of existing services), which aims on creating professional relations.

[1] Apache Hadoop, Hadoop, HDFS, Avro, Cassandra, Chukwa, HBase, Hive, Mahout, Pig, Zookeeper are trademarks of the Apache Software Foundation.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 530–541, 2012.

Although Facebook and *Twitter* start to assimilate technologies mentioned in abstract, social network service has never been built on these storage systems aiming on performance and sustainability.

The benefit is an easy data analysis and distribution using Hadoop Framework[2]. The main task was to create suitable services, which will cooperate with simple framework over HBase to make development easy. The most challenging task is to store users' walls and news feeds without redundancy but still with great performance, as these channels are most important in social network service.

The social network services are heavily data-oriented and are based on model write once, read many as is expected that once submitted post will be read many times. Relational databases are designed for often updating, inserting and reading small chunks of structured data but can fall behind when joining huge tables that can be distributed over the network. As we lose relations among data while using non-relational databases we get great performance, which does not decrease, with amount of data stored in our database.

The application was created as a subsystem and therefore can be implemented in any existing web application that can communicate with social network service interface.

1.1 The Wall

The wall is an important term that will be used often in this paper. It was used on social network service Facebook for the first time. The wall is a webpage on any user's profile where he/she and friends can leave messages, photos, links, and the like.

Friends' posts (sorted by time) are displayed on News Feed – it is an aggregation of other users' posts on one webpage so that user can see important events taking place on social network.

One post consists of user's photography and name, text of the post and time when it has been posted. Visitors can comment and like using a simple interface.

1.2 Entity

System is not limited on physical users only – the user can be for example event or conference – more exact term would be an entity. In this paper the terms entity and user are interchangeable. Same rules stand for every entity.

2 Analysis

Well-known social network services were analyzed before implementing and on the results and other available data basic requirements were set.

[2] Apache Hadoop, `hadoop.apache.org`

2.1 Existing Social Networks

Facebook is a social network service helping people to communicate with friends and family. Facebook influenced application presented in this paper especially in the concepts of posts and comments and also presents two main channels: wall and news feed.

Facebook used *Cassandra*[3], which was developed by its team and later open-sourced, for messaging system until 2010. In 2010 Facebook engineers decided to start using HBase instead as Cassandra's consistency model proved to be insufficient [6].

Twitter works on the main principle of *microblog*. It is a more public service than Facebook that connects people with friends. Twitter allows people to communicate with anyone and is more focused on interests. Twitter inspires our described application mostly at asymmetric relations among users.

Twitter uses several *NoSQL technologies* [3] such as Hadoop and *HBase*[4] for data analysis or people search and also Cassandra for online systems and backups. Cassandra is also planned to be used as the Tweet storage but currently Twitter is not developing this system yet as it would mean a vast intervention into its data model.

LinkedIn is a service focused on professional relations but aims more at creating the relations and building own professional profile than at communicating with each other. LinkedIn is often used by headhunters and with their specialization is closest to the created social network service. LinkedIn uses Hadoop and non-relational database *Voldemort*[5].

2.2 Technological Requirements

- System has to work on any Linux operating system as most recent and innovative technologies are built on this platform.
- Java programming language has to be used (Java EE platform, especially Spring Framework) to ensure cross-platform availability of the software itself. Java also provides professional frameworks and interfaces for most technologies.
- Persistent storage must focus on high throughput because the application is data-oriented. Many concurrent requests can happen any time and the storage has to deal with heavy loads.
- System must use a caching tool to increase performance as the most needed data is stored in a fast memory.

2.3 Functional Requirements

- Entities can create asymmetric relations (*Entity A can follow Entity B but it does not mean that Entity B also follows Entity A*). It is a more professional approach than symmetric relations when speaker does not want to read posts from his/her listeners but wants to let them read his/her posts.

[3] Apache Cassandra, `cassandra.apache.org`
[4] Apache HBase, `hbase.apache.org`
[5] Project Voldemort. A distributed database. `project-voldemort.com/`

- Entities can block any entities so they will not be able to follow them.
- Entity can view walls of entities they are following and each of them has its own wall and news feed. Entities can put posts on their own wall. System can put posts on any wall.
- The author of post or people following the author can put comments. Comments can be deleted by their authors or by post author.
- Entities can like posts (with the very same restrictions as comments).
- System can create recommendations. For example:
  - "This event is favored by X entities you are following."
  - "Y entities you are following follow Entity A."
  - "Z people you are following plan to visit event M."

3 Used Technologies

In order to develop scalable and high performance social network subsystem, the choice of proper technologies is crucial. First of all, the Java Platform Enterprise Edition became the foundation of the solution. Java ensures high level of security and robustness as well as multithread support. On the other hand, certain simplification of low-level routines handling like memory management causes performance risks. Another facilitation of development has been given by Spring[6] framework utilization. Spring is an open-source solution combining other single-layer frameworks into one complex forming resulting system architecture. In this concrete example of social network, Spring is used for Plain Old Java Objects (POJO) adaptation to enterprise solution. POJO thus provides all benefits of Java Enterprise Edition but the programming code is also simple enough for reusability, sustainability or testing [5]. Social networks are well known for processing large amount of data continuously and thanks to huge amount of users accessing the social system simultaneously in very short time as well. For that reason, the Hadoop software framework creates the backbone for distributed environment of presented solution. Hadoop is currently the most important project of Apache Company and it consists of two parts:

1. Hadoop Distributed File System (HDFS)
2. MapReduce

Persistent data layer HBase is strictly independent on Hadoop framework. Processing data model MapReduce (introduced by Google) is built on the ideas of functional programming, dividing processing into two phases both reading the input and writing the output in the form of key-value. In the map phase the main node (master) in distributed network divides problem into several smaller one and then they are assigned to other nodes. These nodes are giving results back to the main node. Reduction phase processes returned results within the main (master) node and assemble them into the original problem [4]. This parallelism helps to avoid dropouts. If particular node would not give the result

[6] SpringSource, www.springsource.org/

back, it could be easily substitute by another one. MapReduce model is suitable for cases when once written data has to be read and process often, especially in huge amounts. Unlike relational databases that were designed for frequent small data blocks writing and updating [10].

HDFS – distributed file system – has been designed for storing large amounts of data with general hardware requirements. HDFS divides files to 64 MiB blocks and in contrast to standard file systems smaller files do not occupy the whole block. The size of 64 MiB helps to decrease seeking time for particular blocks and increase transmission speed of large files. The division of file to those blocks also enables distribution of very large files between several computers with disk drives smaller than original file.

HBase is another component of Hadoop framework, yet independent. It serves as a non-relational database built upon HDFS for quick reading and writing vast data collections. There is no SQL (Structured Query Language), foreign keys, triggers or views because of absence of relations between the data. HBase is inspired by the BigTable[1] model. Particular tables are automatically distributed to so call regions. Region is a subset of rows defined by the first and last row and randomly generated identification. In the beginning, while the database is small enough, it is stored on a single server. When the database limit is reached, HBase is split into two regions distributed through the network to other servers using HDFS [10].

HBase consists of tables in which information is stored in four dimensions: row, column family, column and version. Basically it is a multidimensional sorted persistent distributed map (key-value). Keys and values are array of bytes so any data can be stored. Column families are stated as the table is created and should not be changed later. Columns can have any name and in any column family can be a various number of them. They can be added or changed any time and the database is ready to store millions of them. Information in column can have versions so data can be automatically backed up. As is said database does not support query language over the data but to obtain more rows a developer can use scanner which can fetch rows of some key interval (for example “aa” to “cz” row names).

Data model demonstration (JSON):

```
{ // table
    // previous sorted rows
    "aa" : { // row
        "cf1" : { // column family
            "jedna" : 1, // column and data
            "object" : serialized_data
        },
        "cf2" : {
            "" : "w" // the only column
        }
    },
    // next sorted rows (i.e. "ad", "ba", "cz")
}
```

For the best use of persistent databases, Memcached[7] is one of the most suitable recent technologies that can be utilized. Memcached is a distributed system as well as the previous ones, focused primary on very high performance enabling to cache any information in the form of key-value. Memcached is keeping the data in RAM improving performance of dynamic web applications [2]. A typical Memcached usage is demonstrated below:

```
public Data getData(String query) {
    Data data = memcached.get(query);
    if (data == null) {
        data = database.get(query);
        memcached.set(query, data);
    }
    return data;
}
```

First, the data is retrieved from Memcached layer. If they are not present there, the data is loaded from database, stored in Memcached for future re-usage and sent to the web application. In the beginning, there is a obvious delay caused by accessing Memcached uselessly but each next data call brings significant improvement thanks to accessing directly RAM. Another benefit represents the ability to store Memcached distributed through network on more cooperating servers. Disadvantage of this solution lies in a way how RAM works. It is energy dependent repository and therefore the risk of data lost in case of any failure is considerable (but within the next call we can retrieve the data again from persistent database).

4 Architecture and Design

The system has been designed to be integrated into existing applications. It uses the interfaces, which defines services. These services can be called directly or developer can implement communication layer, which handles remote calls (REST, XML-RPC, JSON-RPC, SOAP and many others) and is connected with system services.

4.1 Inner Architecture

The system has three basic layers connected by interfaces. Service layer can call data access layer that access the storage using the data transmission layer.

Service layer serves as the interface for direct or remote callings from the application. It processes data and authorizes operations.

Data access layer fetches or stores data from/in database and transforms it into business objects and back.

[7] Free and open source, high-performance, distributed memory object caching system, memcached.org

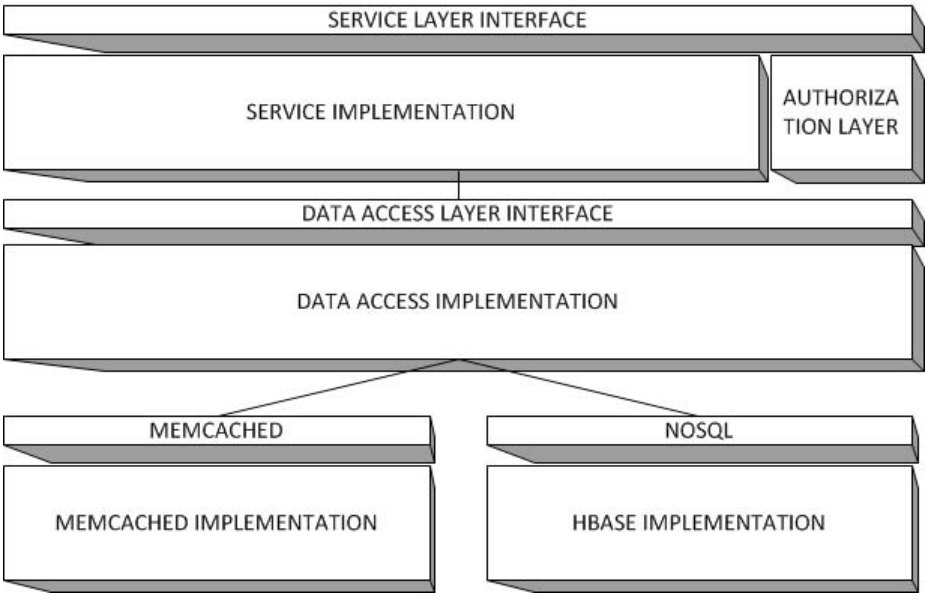


Fig. 1. Three layers of inner architecture

The last layer consists of basic operations like add, delete, get, replace on database or cache and creates a simple framework over any storage.

There are four basic services currently in the application:

1. *Follow service* is responsible for creating and managing relations. It also enables to get data about followers and following (complete or random list which can be paginated).
2. *Wall service* performs actions related to posts and walls. It provides methods for creating, deleting and viewing either the entire post or only minimized one (for example on wall we do not want to show all comments connected with the post).
3. *Like service* allows entities to like and unlike the posts.
4. *Discussion service* manages the comments of specific posts.

4.2 Data Model

The system consists of three tables entities, walls and discussions (their names can be changed before initialization to avoid the problem of identical names).

Table entities consist of five column families and saves basic information about them. Column families followers, following and blocked contains unique ids in columns. The news column family includes post ids to the posts on news feed (stored as a backup when Memcached fails to store news). The last column family count stores redundant information about numbers of followers, following etc. so system does not have to recalculate them every time.

Table walls stores all posts of each entity in the system. Column family info stores basic information like author, type, number of comments and number of likes. In text is only one column containing the text of the post and likes contains unique ids of people who like the post.

Table discussions is very similar to the table walls but does not contain type, likes, number of likes and comments.

4.3 Storing Data

While using HBase, developer has to think about row identifiers as they affect performance. Rows are sorted lexically (identifiers are arrays of bytes) and the only possibility how to obtain more lines is a sequence scanner, therefore there results the need to store similar data in a batch to easily fetch them.

In table entities is unique identifier (UID) the key which provides correct sorting in other tables that uses this id as part of their keys. The key should have constant length to ensure that data are always lexically similar.

Table walls uses concatenation of UID and time of the post (Java SimpleDateFormat `yyyyMMddhhmmssSSS`). Thanks to that there exists only one length of the key so posts are grouped by entities and then sorted by time – so each entity has posts sorted from oldest to newest so fetching data is fast for each entity and there is higher probability that it will be stored on the same region server. Also entities can view complete history of posts on the wall.

In fact the posts are stored in reverse order (newest to oldest) so developer can very fast obtain only certain number of newest posts (in the opposite sorting it would be really hard as the scanner can read only in direction first to last row). The solution is quite clear – all bytes in formatted date are inverted.

There are only weak relations among data. The entities have their posts and they have their comments. These relations are displayed in names of keys. Other relations are not clearly seen in the database because they are not important.

4.4 Wall and News Feed

As is said in previous paragraph, posts are stored in table walls for each entity so it is simple and fast to obtain wall for any entity. The news feed is stored in HBase only like a backup of Memcached data (for each entity post identifiers are stored so we can get the news). Fetching the news from HBase is relatively slow because we need to load the posts one by one. It is the only solution of this problem – otherwise we would create a huge redundancy (as many copies as many users are following). In this case Memcached can avoid redundancy and also make access to data fast.

While sending a new post to the server all entities following author are discovered. The post is inserted in the cache in minimized form (also time to live is set) and link to it is put in news feed to every interested entity. We can obtain news feed in two simple cache queries. First one fetches the list of posts and the second one (batch query) returns the posts. As Memcached is a simple hash

table in RAM getting a few posts to display is a really fast operation. Once the data gets old or Memcached is full the expired posts are deleted first and after them the least recently used.

4.5 Implementation

System was implemented using Java EE and Spring Framework, client libraries for Memcached is Spymemcached developed by one of founders of Memcached and client library for HBase is *HBase 0.90.2 API for Java*.

5 Case Study

The application was created to ensure social interactions for users of web application Takeplace.

Takeplace[8] is a web-based service designed and implemented on the basis of expert requirements for the particular tool to facilitate efficient organization of events based on meeting, sharing and communication.

Users are provided with a web interface and gain an access to the standard services for organizing events according to the type. Basic types of events include conferences, congresses, symposia, seminars, trainings, consultation meetings, workshops and team buildings.

With increasing number of professional events organized with the Takeplace system, we will be able to evaluate these events. Users themselves will be able to share the experience of participation, evaluate and recommend events to others. Then, virtual communities of professional users will arise in course of time. The emphasis on social and interpersonal interaction is the essential feature of Takeplace [9].

Social network service was implemented for several reasons. One of the most important was to provide an easy tool to comment conferences or individual events and give a feedback to organizers and speakers. Attenders can find problems that hosts do not see and help them make a successful conference. Users can also "like" events, which should compare particular seminars and give an opportunity to people to tell which events are the best.

The second very important reason is to create a professional user network. Attenders of conferences are people with common professional interests and therefore a service allowing finding new contacts, create specialist relations and recommend each other is needed to be implemented.

Several other use cases:

- Simply inform a group of people.
- Find out how many visitors could attend particular event.
- Hear the topics the attenders want to study.
- Share any information among users of service.

[8] Takeplace Event Management Tool, `www.takeplace.eu`

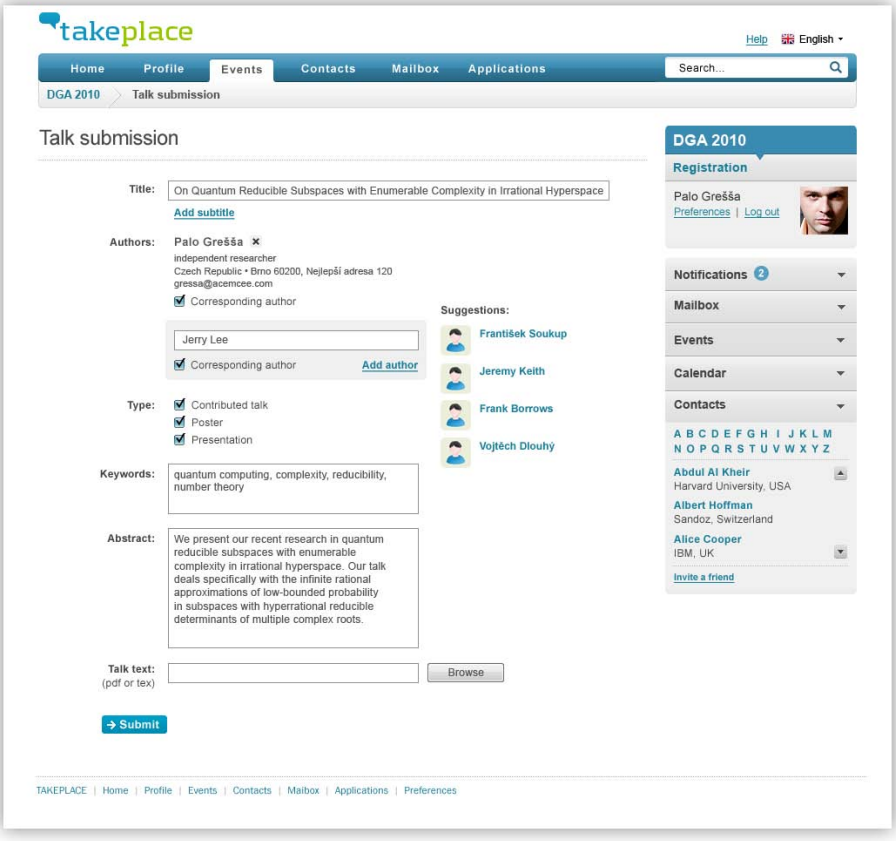


Fig. 2. Takeplace event management tool – submission

5.1 Implementation

Service layer of social network service is connected with Takeplace by communication layer that uses JSON to send needed data. Takeplace can retrieve certain information and connects them with their database of users. User identifiers both in the application and in the subsystem remain the same so there is no need to transform them. Takeplace sends to the social subsystem authenticated unique id so it can authorize all needed operations.

Any data fetched from social network can be post-processed and the application can for example after obtaining list of all followers load names and photographs to display information. The social network subsystem independence, easy integration and configuration are one of key features.

5.2 Testing

This application was tested using Jakarta JMeter, Netbeans Profiler and also private pilot run. The application was deployed to a single very limited machine

(with only one dual core processor at 2 GHz and 1GB RAM) with Tomcat 6.0.26 as it was enough to see how the application performs with 100 hundred users.

In average one simple follow invokation took 0.5 ms, obtaining the list of all one hundred followers and thirty random followers took 2.2 ms (data access time was less than 1 ms). Sending one post to server consumed 2.58 ms in average and loading wall of 35 posts for single entity took about 4 ms. Load time of entity news feed was 7 ms.

On server we got throughput of about 700 requests per second and median of loading a simple page which performed one follow operation was 304 ms. The most important page's (News Feed) throughput is 440 requests per second and median was 354 ms.

Memcached heavily improved the loading time as data load operation of getting all followers improved about 1000 as the operation needs a single request to memcached. Loading the news feed is even faster as Memcached only performs batch query for needed keys and HBase has to find the posts in each user's post (random reads). Memcached proves to be useful immediately and developers should not use it for only two reasons. Either the applications update the data too often so the cache hit is not probable or the single chunk of data is larger than 1 MiB. This application is based on write once read many so Memcached brought significant performance improvment.

The testing showed that Memcached is really improving the load time. The comparison between relational and non-relational databases will be evaluated when high volume of data will be accessed by our users. The testing data from pilot run look really promising. The main advantages of using the HBase is scalability and also flexibility as we can react easily when we need to change the structure of data while implementing new features.

The testing data are only estimation as the application was running on single node, using one Memcached server and one HBase region. In the beginning of 2012 we will deploy our application to cloud service and then we will evaluate the performance again as it can improve or downgrade.

6 Conclusion

The social network service was designed as a subsystem and can be integrated into existing application by implementing the communication layer and connect it with existing database of users.

Social module allows creating asymmetric relations among users and events. Entities can insert posts, which can be "liked" and commented. Posts are displayed on entity's profile wall and user can view all posts in the history. News feed displays the newest posts from entities news feed owner is following. The news is stored for limited time that can be set up in configuration files.

Subsystem uses non-relational database HBase as permanent storage and Memcached as distributed cache to improve performance, as is expected heavy load on fetching data from database. Non-relational database and memory cache should decrease reading time and provide high throughput and scalability in data-oriented web application.

Subsystem was integrated in Takeplace application and tested in a pilot run with limited users. The availability on production servers is planned in the beginning of 2012.

The paper aimed to present a modern layered architecture and propose use of technologies to ensure high volume data distribution, scalability and high throughput.

Described system is unique for connecting the most innovative technologies currently developed and as seen on pilot testing the performance should increase in comparison to ordinary RDBS with amount of data stored in database. At the expense of performance, developer loses the relations among data which creates slightly more responsibility to ensure that data is consistent and valid but with right choice of keys and creating a suitable framework even this problem can be overcome.

References

1. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A distributed storage system for structured data. ACM Trans. Comput. Syst., 26, 4:1–4:26 (2008)
2. Finsel, J.: Using Memcached: How to scale your website easily (2008), http://pragprog.com/titles/memcd/using-memcached
3. Leavitt, N.: Will nosql databases live up to their promise? Computer 43(2), 12–14 (2010)
4. Lin, J., Dyer, C.: Data-intensive text processing with mapreduce. Synthesis Lectures on Human Language Technologies 3(1), 1–177 (2010)
5. Machacek, J., Vukotic, A., Ditt, J., Chakraborty, A.: Pro. Spring 2.5. Springer, Heidelberg (2008)
6. Muthukkaruppan, K.: The underlying technology of messages. Facebook Engineering (2010)
7. Newman, J.: Time: What decline? facebook may have just reached 750 million users (2011), http://techland.time.com/2011/06/24/what-decline-facebook-may-have-just-reached-750-million-users/
8. O'Reilly, T.: What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. O'Reilly Media (2007)
9. Škrabálek, J., Ludík, T., Slabý, J., Pitner, T.: Web-based service for collaborative organization of academic events–case study of takeplace. In: Tetsuo, I., Viorel, N., Jebelean, T., Petcu, D., Watt, S., Zaharie, D. (eds.) 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), pp. 577–580. IEEE Computer Society (2010)
10. White, T.: Hadoop: The Definitive Guide. O'Reilly Media (2009)

State Coverage: Software Validation Metrics beyond Code Coverage

Dries Vanoverberghe[1,⋆], Jonathan de Halleux[2], Nikolai Tillmann[2], and Frank Piessens[1]

[1] Katholieke Universiteit Leuven, Leuven, Belgium
{dries.vanoverberghe,frank.piessens}@cs.kuleuven.be
[2] Microsoft Research, Redmond, WA, USA
{jhalleux,nikolait}@microsoft.com

Abstract. Currently, testing is still the most important approach to reduce the amount of software defects. Software quality metrics help to prioritize where additional testing is necessary by measuring the quality of the code. Most approaches to estimate whether some unit of code is sufficiently tested are based on code coverage, which measures what code fragments are exercised by the test suite. Unfortunately, code coverage does not measure to what extent the test suite checks the intended functionality.

We propose *state coverage*, a metric that measures the ratio of state updates that are read by assertions with respect to the total number of state updates, and we present efficient algorithms to measure state coverage. Like code coverage, state coverage is simple to understand and we show that it is effective to measure and easy to aggregate. During a preliminary evaluation on several open-source libraries, state coverage helped to identify multiple unchecked properties and detect several bugs.

Keywords: state coverage, test adequacy metric, test oracle.

1 Introduction

As software becomes a central part of society, the impact of software defects on the economy is huge. For example, in 2002, software failures were estimated to cost the US economy about $60 billion annually [17]. Currently, testing is still the most important approach to reduce the amount of software defects.

During the testing process, the code under test is exercised in various ways while a test oracle (e.g. assertions or pre- and post conditions) checks that the code behaves according to its specification. Defects are reported and fixed and the testing process restarts. In principle, this process can continue forever since testing usually cannot show the absence of software defects. In practice however, only limited resources are available and testing needs to stop at some point. Software quality metrics help to prioritize where additional testing is necessary by measuring the quality of the code under test.

⋆ This work was done during an internship at Microsoft Research. Dries Vanoverberghe is a Postdoctoral Fellow of the Fund for Scientific Research - Flanders (FWO). This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, by the IWT, and by the Research Fund K.U.Leuven.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 542–553, 2012.

Currently, most *software validation metrics*, i.e. metrics that estimate whether the code is sufficiently tested, are based on code coverage. Code coverage estimates the fraction of the execution paths of the code under test that are exercised by the test suite. Since code coverage metrics are simple to understand and efficient to compute, the use of code coverage metrics during the testing process is well-established. Furthermore, automatic tools have been created to help testers achieve high code coverage (e.g. random testing, symbolic execution[13]). Unfortunately code coverage alone is not sufficient to measure software quality since it only measures whether the code has been sufficiently exercised. It does not measure the strength of the test oracle, the properties that must be satisfied by the code.

In this paper, we focus on the use of assertions, one of the most basic ways to instrument the code with the test oracle. Whenever the execution reaches an assertion, the execution state must satisfy the given boolean expression. Although the use of assertions is far from new [10,23,12,20] and experimental evidence [15] shows there is a correlation between the number of assertions and the amount of software defects, little work has been done to measure the quality of assertions in a test suite.

We propose the use of *state coverage* [14], a software validation metric based on the hypothesis that every update to the execution state must eventually be followed by an assertion that reads the updated value. State coverage is orthogonal to code coverage: they measure different concerns. While state coverage measures the strength of the test oracle, code coverage measures how well the code is exercised. Nonetheless they are intertwined, for example adding extra assertions to the test suite may decrease code coverage and exercising more paths of the program may discover new state updates and decrease state coverage. Therefore code coverage and state coverage work best in combination. In addition, the thought process of developers to achieve high state or code coverage is also orthogonal: While code coverage makes a developer think in terms of branches, state coverage makes a developer think in terms of properties that are established by state updates.

For a good software validation metric, the following criteria are essential:

- **easy to understand**, for developers and testers who write code and tests to achieve certain metric numbers, and for managers to decide when a project is ready to be shipped,
- **composable**, i.e. results from individual test cases can be combined to an overall result for an entire test suite,
- **effective to measure**, i.e. adding only a reasonable overhead during the software development and testing process.

We show in this paper that state coverage fulfills all of the above criteria.

Except for Mutation Testing [8], state coverage is the only technique to measure the quality of the test oracle. Unfortunately, the mutation adequacy score is hard to understand because deciding whether a live mutant is equivalent can be complex and often requires human intervention. In addition, it suffers from a high performance penalty caused by executing the test suite with millions of mutants.

We have implemented a prototype of the state coverage metric for the .NET platform, and have applied it to several open-source libraries. While adding extra assertions

to increase state coverage, we have found several bugs in DSA[2,1], a library with complementary data structures for the .NET platform. In total, we found seven properties which were not or insufficiently checked in the existing test suite.

To summarize, the main contributions of this paper are:

- We propose a general definition for *state coverage*, a software validation metric that goes beyond code coverage. Our definition improves on existing work by Koster et al.[14] by allowing more dynamic state updates and lifting the restriction on the structure of test cases.
- We present efficient algorithms to measure *object sensitive* and *object insensitive* state coverage, two variants with different granularity.
- We propose a technique to make *object sensitive* state coverage composable.
- We evaluate the metric in a case study on several open-source libraries, using a prototype implementation of our algorithm.

The remainder of this paper is structured as follows. First, Section 2 introduces state coverage and discusses how it can be computed. Then, we evaluate state coverage in Section 3. Finally, we discuss related work and conclude in Sections 4 and 5 respectively.

2 State Coverage

In this section, we propose state coverage, an approach that measures the percentage of the state updates that are verified by an assertion. We start with its definition, and then give a simple algorithm to track state coverage of a single test case at runtime. We describe how state coverage data of individual test cases can be combined into overall state coverage information, in order to measure state coverage of an entire test suite. Finally, we extend the algorithm with dependency tracking to avoid low state coverage ratios due to intermediate state updates and missing context information.

2.1 Definition

We define *state coverage* as the ratio of the number of *state updates* which are read by assertions to the total number of state updates.

This definition of state coverage depends on the definition of state updates. Just as there are different characterizations of code coverage (statement, basic block, arc, etc.), there are different possible characterizations of state coverage, depending on the chosen granularity of state updates.

In this work, we propose two such granularities of state coverage:

- *Object insensitive* state coverage considers as a state update the *code location* in the source code where an update is performed.
- *Object sensitive* state coverage considers as a state update a pair of *object identifier and code location*, where the object identifier is derived from the actual object reference that is involved in a state update at runtime.

Object insensitive state coverage is quite similar in nature to the idea of statement coverage. It simply relates a number of covered code locations to a total number of code locations. While easy to understand, statement coverage is often not fine-grained enough to give confidence that the code has been sufficiently exercised. Similarly, object insensitive state coverage is rather coarse. While it provides some basic insights into the quality of a test suite, we have found cases where only striving for object sensitive state coverage could uncover certain software defects.

We have implemented a prototype to compute state coverage based on runtime monitors. To get the state coverage, all test cases of a given test suite are executed with a special monitor, which gets callbacks during the execution, for example, whenever a field is read or written. Sections 2.2 and 2.3 discuss the implementation of the monitors for object insensitive and object sensitive state coverage. Both monitors collect a set of state updates (*writes*) and a subset that is read in assertions (*reads*). The resulting value is computed by dividing the number of reads by the number of writes. More details about this algorithm can be found in an extended technical report of this paper [25].

2.2 Object Insensitive State Coverage

Figure 1 shows the basic algorithm to compute the object insensitive state coverage metric. The class *ObjectInsensitiveStateCoverageMonitor* is a runtime monitor which gets notified whenever a field is written (*WriteField*) or read (*ReadField*) and upon entering (*EnterMethod*) and leaving methods (*LeaveMethod*). The state coverage monitor uses *EnterMethod* and *LeaveMethod* to track whether the execution is currently inside the assert method. Whenever a field is written, the current code location is added to a set *writes*, which tracks all code locations where write operations are performed by the program. A code location represents a method and the offset of an instruction in the body of that method. In addition, the written object and field is associated with the current code location in the map *lastWrite*, which tracks the last location where each object field has been written. When the execution is inside an assert method and the execution reads an object field, the last write location for that object field is added to the set *reads*.

For simplicity, the presented algorithm only deals with writes to object fields. However, other parts of the state, such as static fields, array elements, or struct fields, can be handled similarly.

In general, computing the set of object fields that influence an assertion corresponds to information flow analysis [21]. For simplicity, the algorithms in this paper simply track all fields read during the computation of the assertion. However, our implementation uses runtime information flow monitoring to give more precise results.

Composing the results from all test cases of an entire test suite is easy: It simply amounts to computing the union of the respective reads and writes sets.

2.3 Object Sensitive State Coverage

The basic idea of the object sensitive algorithm is similar to the object insensitive algorithm. The main difference is that instead of just tracking code locations for the reads

```
class ObjectInsensitiveStateCoverageMonitor
extends Monitor<CodeLocation> {
 Map<Pair<object,Field>,CodeLocation> lastWrite;
 bool InAssert;
 ...
 void WriteField(object o, Field f) {
   var loc = CurrentCodeLocation();
   lastWrite[new Pair(o, f)] = loc;
   writes.Add(loc);
 }
 void ReadField(object o, Field f) {
  if(inAssert) {
     var loc = lastWrite[new Pair(o, f)];
    reads.Add(loc);
  }
 }
 void EnterMethod(Method method) {
  if(method == Assert.IsTrue)
   inAssert = true;
 }
 void LeaveMethod(Method method) {
  if(method == Assert.IsTrue)
   inAssert = false;
 }
}
class Assert {
 static void IsTrue(AssertExpr expr) {
  if(!expr())
   throw new AssertionViolationException();
 }
 delegate bool AssertExpr();
}
```

(a) Object Insensitive State Coverage Monitor

```
class Frame {
 int idCounter;
 Map<object, int> ids;
 int GetLocalId(object o) {
  if(!ids.ContainsKey(o))
   ids[o]= idCounter++;
  return ids[o];
 }
}
class ObjectSensitiveStateCoverageMonitor
extends Monitor<Pair<int,CodeLocation>> {
 Map<Pair<object,Field>,
  Set<Pair<int,CodeLocation>>> lastWrites;
 bool InAssert;
 ...
 void WriteField(object o, Field f) {
  var s = new Set<Pair<int,CodeLocation>>();
  foreach(var frame in CurrentFrames) {
    var loc = frame.CodeLocation();
    var id = frame.GetLocalId(o)
    var p = new Pair(id, loc);
    s.Add(p);
    writes.Add(p);
   }
   lastWrites[new Pair(o, f)] = s;
 }
 void ReadField(object o, Field f) {
  if(inAssert) {
     var ps = lastWrites[new Pair(o, f)];
     reads.AddRange(ps);
  }
 }
}
```

(b) Object Sensitive State Coverage Monitor

Fig. 1. State coverage monitors

and writes sets, we will track more information, allowing us to distinguish writes which might have happened at the same code location, but were performed on different objects.

The obvious and most general approach would be to track pairs representing the actual object reference together with the code location where it was written. However, this is impractical. While it may allow computing very precise state coverage for individual test cases, joining the information of individual test cases to obtain an overall state coverage ratio for a test suite becomes challenging: It is not clear how to relate the actual object references of different test runs.

Composing state coverage information is important. It is quite common that multiple unit tests check different properties of the same code unit. Some people even consider it bad practice to write multiple assertions in one test case [18]. For example, the methods *PairTest.Test1* and *PairTest.Test2* in Figure 2 check that the constructor correctly initialized the field x and y respectively. Since the allocated pair p will not be identical during the execution of both test cases, a simple union of the read and written fields leads to a joined state coverage of only 50 percent. On the other hand, it also happens that one single test checks multiple properties on different objects.

As discussed above, the global nature of object references poses a challenge to join the results of multiple executions of a particular method or for all methods of a particular type. To enable joining for object sensitive state coverage, the state coverage monitor needs to maintain context-insensitive object identifiers. Figure 1 contains an

```
class Pair {
  int x, y;
  public Pair(int x, int y) {
    this.x = x;
    this.y = y;
  }
}
```

```
class PairTest {
  void Test1() {
    var p = new Pair(27, 33);
    Assert.IsTrue(p.x == 27);
  }
  void Test2() {
    var p = new Pair(27, 33);
    Assert.IsTrue(p.y == 33);
  }
}
```

Fig. 2. Example to illustrate composition

updated version of the object insensitive algorithm. Each frame maintains a map from the actual object references to frame-local object identifiers. Whenever an object identifier is requested and the frame does not yet have an identifier, it allocates the next identifier, represented by returning the *idCounter* and increasing its value by one. All reads and writes use the frame-local identifier instead of the actual object reference. In addition, the last write also tracks the frame-local identifier. For brevity, we omitted the code to push and pop frames when entering and exiting a method.

The resulting set of reads and writes can be unioned together over multiple tests, which makes it possible to report state coverage values for all tests in a particular test fixture or test assembly.

There are alternative approaches to to assign context-insensitive object identifiers. We have chosen the current approach, because it is simple and effective without sacrificing precision.

3 Evaluation

We have implemented a prototype for the state coverage metric as an extension for Pex [24], an automatic test input generation tool for .NET developed at Microsoft Research, based on the idea of dynamic symbolic execution: During the execution of a program, Pex maintains a symbolic state and uses it to generate new inputs that drive the execution to some unexplored path of the program. We implemented our runtime monitor on top of Pex, leveraging its extensible instrumentation framework. As shown in the algorithms in Section 2.2 and Section 2.3, the prototype focuses on object field updates.

The aim of this paper is to present and investigate the notion of state coverage. We want to avoid the influence of automatic test generation on the evaluation of state coverage. Therefore, we use existing projects with manually created test suites, and we do not generate additional test cases with Pex. Since state coverage is computed using a run-time monitor, it is essential that these projects already have a test suite with high code coverage. In addition, state coverage is only useful when the test cases use assertions to specify the test oracle. Based on these criteria, we applied our prototype on the following open-source libraries:

- Quickgraph [7] is a managed C# port of the Boost Graph Library.
- Data Structures and Algorithms (DSA) [21] features implementations of data structures and algorithms that complement the data structures in the .NET 3.5 base class libraries.

Ultimately, the main research question is whether code bases with low state coverage are more likely to have bugs, while code bases with high state coverage are unlikely to have bugs. Unfortunately, answering this question is troubled because of two reasons: First, all projects have been tested reasonably well which implies that the likelihood of findings bugs is low. By consequence, the evaluation is biased. Second, we do not have historical information about older bugs. Therefore it is not possible to compute a reliable correlation between the amount of bugs and the state coverage values.

Since a quantitative analysis is challenging, we perform a more qualitative analysis. Our experiment answers the following research question: "for code bases with good structural coverage, how does an increase in state coverage impact the number of bugs found?"

We measure the initial state coverage of each project and manually add new assertions to read object fields that were written but not read in an assertion. When we can no longer improve the state coverage score, we report the amount of added assertions and the number of bugs we discovered. In this process, we give preference to the simplest assertions that increase state coverage over well-known more complicated invariants. In a realistic test setting, more time could be spent to come up with more valuable invariants. This implies that the results are conservative, i.e. it is in some sense the weakest set of invariants that maximizes state coverage.

For one of the projects, DSA, we perform a detailed analysis of the added properties. We discuss the most useful invariants, and assess whether they add value to the test suite. For this project, we also evaluate the level of false positives/negatives. False positives show up as uncovered state updates, and are therefore easy to detect. False negatives are harder to detect. Therefore, we manually inspected the code bases to find patterns that can cause false negatives.

Most of the assertions are added as invariants or post-conditions using Code Contracts [9]. By consequence, the impact on the existing code base is minimal and the added properties are checked at multiple locations.

3.1 General Results

Table 1 contains the results of executing our prototype on the original unmodified projects. Since all projects have been reasonably well tested, they have high basic block coverage (column 2). Columns three and four report the object insensitive and object sensitive state coverage. All projects have a high score on the object insensitive state coverage, and therefore require few additional properties to reach the maximum ratio. This is not surprising since they had high code coverage and a significant number of assertions (See Table 2). In fact this represents a significant (conservative) bias of our evaluation. We expect that object insensitive state coverage is more useful on average projects. The object sensitive state coverage ratios are lower, and highlight the need for some useful properties. Table 1 show the results for QuickGraph and DSA after adding additional assertions. For both projects, we achieved the maximal ratio.

Column five and six show the performance overhead of object insensitive and object sensitive state coverage. However, we made no attempt to reduce the execution overhead of the prototype, therefore there may be room to improve these numbers. What is

essential about the overhead is that it clearly is just a constant factor off the original performance, not unlike what one would expect from measuring code coverage overhead.

One may object that writing additional assertions just to increase a new metric is a burden for the developer. Column 3 in Table 2, shows the number of lines of code that the traditional test suites required in order to achieve high code coverage. The code added to increase state coverage was negligible compared to the size of the existing test suite.

Table 1. State coverage results before and after adding extra assertions

Project	basic block coverage	State coverage		Performance overhead	
		Obj. insens.	Obj. sens.	Obj. insens.	Obj. sens.
DSA (before)	1580/1608 (98.26%)	69/71 (97.18%)	552/805 (68.57%)	22.89%	29.92%
QuickGraph (before)	553/658 (84.04%)	17/19 (89.47%)	1006/1307 (76.97%)	374.91%	291.17%
DSA (after)	1973/2074 (95.13%)	71/71 (100.00%)	801/801 (100.00%)	45.33%	57.65%
QuickGraph (after)	673/779 (86.39%)	19/19 (100.00%)	1304/1304 (100.00%)	353.93%	276.60%

Table 2. Added assertions

Project	Assertions (Added/Total)	LOC (Original/Total)	Bugs
DSA	33/461	999/1036	5
QuickGraph	22/56	373/426	0

3.2 Detailed Evaluation of DSA

First, we evaluate the bugs we found in DSA while adding properties, and we describe the process that led us to them. The relevant code fragments for these bugs can be found in the extended version of this paper [25]. First, some locations in the code write to the root field of the binary search tree, and the left and right fields of the nodes, but these writes were never read in an assert. The simplest invariant for reference fields is checking whether they are non-null. Since the root, left and right fields can be null, we needed a more complicated invariant. The simplest invariant we could find was that the amount of nodes in the tree must equal the count field of the BinarySearchTree. After inserting this invariant, one of the existing tests failed. Upon closer inspection, it revealed that a value was ignored when it was already in the tree, but the count was still increased.

Interestingly, when increasing state coverage, a developer thinks differently about the code than when trying to increase traditional code coverage. In code coverage, when a statement is uncovered, a developer needs to look at the *branch condition* that preceeds this code fragment. In state coverage, when a state update is uncovered, a developer is forced to think about the properties that are established by this state update.

This mindset helped us discover the other four bugs. After reaching full object insensitive and object sensitive state coverage, it was surprising that we did not need to

specify some properties, in particular, that the reachable nodes in a linked list can not be shared between different linked lists. The existing test suite did not check this property, and both *SinglyLinkedList* and *DoublyLinkedList* have the methods *AddBefore* and *AddAfter* which insert a value before or after a given node. None of the existing tests attempted to invoke these methods with a node that was not in the linked list. The implementation of these methods does not validate that the nodes are in the linked list, and throws a null reference exception or invalidates the invariants concerning the structure of the linked list.

The process to find the bugs in the linked lists illustrates that it is not required to have a full functional specification in order to reach full state coverage. This is due to the fact that state coverage is fundamentally an underapproximative measure for the strength of the test oracle. For example, it was also not necessary to check the consistency between previous and next fields of doubly linked list, or sortedness of the values of the binary search tree.

Nonetheless, 33 additional assertions were added to achieve full state coverage. Four of those assertions were introduced in the new test cases that detect the bug in the linked lists. Eleven of the assertions were trivial data structure invariants, which were enforced locally (for example, in the constructor). Those invariant are less useful, but it is likely possible to infer them automatically using existing invariant generation techniques. Now we discuss the most useful properties that were inserted (using 18 out of the 33 assertions):

- We added a post condition that checks if an element that added to a data structure is contained by the data structure (1 post-condition in CollectionBase, 2 more for specialized methods in Deque, a double-ended queue).
- We added an invariant that checks whether the height of all reachable nodes in an AvlTree is consistent with its actual height (1 invariant, 2 post conditions).
- An invariant in BinaryTree checks that the amount of reachable nodes equals the count of the binary tree (1 invariant).
- None of the tests checked that constructors of heap correctly initialize the strategy field (1 postcondition for both constructors).
- Some data structures are a wrapper around other data structure, but do not always check that the Count field is consistent (2 invariants).
- In the linked lists, the tail pointer is null if and only if the head pointer is null. In addition, the next field of the tail and the previous field of the head must be null (5 invariants in total).
- An invariant in the linked lists checks that the amount of reachable nodes equals the count of the binary tree. In addition, it checks that the previous field of all reachable nodes (except the head node) is not null (2 invariants).

In the end, we achieve 100% state coverage, which implies that the potential false positives due to the dependency tracker did not occur in practice. Intuitively, the lack of false positives due to the intraprocedural algorithm can be explained because the results of a method call are usually consumed in the same branch as the invocation. In addition, the false positives due to tracking dependencies at runtime do not occur because the expressions in assertions are typically simple. Finally, we manually examined the source code for patterns that cause false negatives, and we did not find such patterns.

4 Related Work

In the broader sense, state coverage is part of the larger area of software quality metrics. Empirical studies have shown that complexity metrics (e.g. Cyclomatic complexity [16]) and object oriented design metrics (e.g. Coupling [5]) can be used to predict defect density (See Catal et al. [3] for a survey on defect prediction). However, such metrics only indirectly help to reduce the defect rate by measuring the quality of the design.

More narrowly, state coverage is a *test adequacy metric* (See Zhu et al. [26] for a survey on test adequacy criteria), it directly measures how well the software has been validated. Structural coverage metrics (such as statement coverage) are most popular in this area. They all measure to some extent which subset of the execution paths of the program are exercised by the test suite. State coverage is orthogonal to these metrics, since it measures the strength of the test oracle. Therefore, state coverage is most powerful in combination with the existing approaches.

State coverage is most closely related with all-defs [19] coverage. The critical difference between both is that dataflow coverage works with all state reads, whereas state coverage focuses on state reads that influence the result of an assertion. This difference makes state coverage measure the strength of the assertions in a code base, instead of measuring whether the code base is sufficiently exercised.

With respect to Koster et al. [14], our definition of state coverage is more general. We do not require any particular structure for the tests. Furthermore, we allow more dynamic state update identifiers (e.g. by including object identifiers) than nodes in the control flow graph. Our preliminary experiments have shown that a more dynamic version can reveal more faults and is therefore more precise. Finally, we do not restrict the metric to output-defining nodes. Therefore our algorithm gives a more accurate view of the fraction of state updates that have been checked by assertions.

Structural test adequacy criteria have also been applied to specifications instead of programs [4,11,6]. These metrics consider the system as a black box, and evaluate whether the test suite sufficiently exercises a model of the system. Therefore, they measure the quality of a set of tests rather than the quality of the test oracle. Unlike pure program based or specification based adequacy metrics, state coverage uses the structure of the program and the specification.

Next, fault-based test adequacy criteria (mostly mutation testing [8]) measure the fault finding capability of a test suite. Unlike existing structural test adequacy criteria, mutation testing can be used to evaluate the strength of the test oracle. Mutation testing injects faults into the codebase and checks whether the test suite can observe the injected fault. Often mutation testing requires generating and executing millions of mutants. The mutation adequacy score divides the amount of killed mutants by the amount of non-equivalent mutants. Unfortunately, deciding whether a mutation is equivalent is undecidable in general, and therefore often requires human interaction. State coverage achieves some of the benefits of mutation testing, without the performance overhead and complexity of mutation testing.

The number of assertions in a code base have been shown to correlate inversely with the amount of defects in the code [15]. Based on this observation, counting the number of assertions is an obvious metric for the strength of the oracle. State coverage goes beyond assertion count in that it is more constructive: it highlights parts of the state that

are not mentioned by any assertion. In addition, assertion count does not normalize as well as state coverage: The ideal assertion density (the assertion count divided by the size of the code base) may vary from program to program.

From a more technical perspective, state coverage is related to UnitPlus [22], a tool to assist developers in writing unit tests. Based on a static read/write analysis, UnitPlus suggest new assertions. Unlike UnitPlus, our algorithm computes state coverage at runtime and therefore it can be more precise (e.g. we don't have problems due to aliasing). In addition, the algorithm in Section 2 uses dependency tracking to precisely track which state has been read while constructing expressions.

5 Conclusion and Future Work

In this paper, we went beyond traditional code coverage metrics to assess the quality of a test suite. We created state coverage, a novel metric that measures the ratio of state updates that are read by assertions, and we presented efficient algorithms to measure state coverage. We have implemented a prototype to measure state coverage, and evaluated the metric in a case study on several open-source libraries. State coverage helped to identify multiple unchecked properties and detect several defects.

In future work, we will further experiment with different frame-local object identifiers. Although the current identifiers are simple and efficient, they do not provide enough feedback to debug which object is updated and cover this update in a new assertion.

In addition, we plan to measure some notion of redundancy of assertions to avoid trivial assertions such as tautologies or otherwise implied properties. We envision a redundancy notion which gives individual assertions a score, which measures its quality, similar to how state coverage quantifies the quality of a test case or suite, but possibly another orthogonal metric.

References

1. Barnett, G., Del Tongo, L.: Data Structures and Algorithms: Annotated Reference with Examples. NETSlackers (2008)
2. Barnett, G., Del Tongo, L.: Data structures and algorithms, dsa (2008), http://dsa.codeplex.com/
3. Catal, C., Diri, B.: A systematic review of software fault prediction studies. Expert Systems with Applications 36(4), 7346–7354 (2009)
4. Chang, J., Richardson, D.J., Sankar, S.: Structural specification-based testing with adl. In: Proceedings of the 1996 ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 1996, New York, NY, USA, pp. 62–70 (1996)
5. Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Trans. Softw. Eng. 20(6), 476–493 (1994)
6. Dadeau, F., Ledru, Y., du Bousquet, L.: Measuring a java test suite coverage using jml specifications. Electronic Notes in Theoretical Computer Science 190(2), 21–32 (2007); Proceedings of the Third Workshop on Model Based Testing
7. de Halleux, J.: Quickgraph: A 100% c# graph library with graphviz support (2007), http://www.codeproject.com/KB/miscctrl/quickgraph.aspx

8. DeMillo, R.A., Lipton, R.J., Sayward, F.G.: Hints on test data selection: Help for the practicing programmer. Computer 11(4), 34–41 (1978)
9. Fähndrich, M., Barnett, M., Logozzo, F.: Embedded contract languages. In: SAC 2010: Proceedings of the 2010 ACM Symposium on Applied Computing, New York, NY, USA, pp. 2103–2110 (2010)
10. Floyd, R.W.: Assigning meanings to programs. Mathematical Aspects of Computer Science 19(19-32), 1 (1967)
11. Heimdahl, M.P., George, D., Weber, R.: Specification test coverage adequacy criteria = specification test generation inadequacy criteria? In: IEEE International Symposium on High-Assurance Systems Engineering, pp. 178–186 (2004)
12. Hoare, C.A.R.: Assertions: A personal perspective. IEEE Ann. Hist. Comput. 25(2), 14–25 (2003)
13. King, J.C.: Symbolic execution and program testing. Commun. ACM 19(7), 385–394 (1976)
14. Koster, K., Kao, D.: State coverage: a structural test adequacy criterion for behavior checking. In: The 6th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers, ESEC-FSE Companion 2007, New York, NY, USA, pp. 541–544 (2007)
15. Kudrjavets, G., Nagappan, N., Ball, T.: Assessing the relationship between software assertions and faults: An empirical investigation. In: ISSRE 2006: Proceedings of the 17th International Symposium on Software Reliability Engineering, pp. 204–212. IEEE Computer Society, Washington, DC, USA (2006)
16. McCabe, T.J.: A complexity measure. IEEE Trans. Softw. Eng. 2(4), 308–320 (1976)
17. N.I. of Standards and technology. The economic impacts of inadequate infrastructure for software testing. Planning Report 02-3 (2002)
18. Osherove, R.: The Art of Unit Testing with examples in .NET. Manning Publications Co. (2009)
19. Rapps, S., Weyuker, E.J.: Selecting software test data using data flow information. IEEE Trans. Softw. Eng. 11, 367–375 (1985)
20. Rosenblum, D.: A practical approach to programming with assertions. IEEE Transactions on Software Engineering 21(1), 19–31 (1995)
21. Sabelfeld, A., Myers, A.C.: Language-based information-flow security. IEEE Journal on Selected Areas in Communications 21(1), 5–19 (2003)
22. Song, Y., Thummalapenta, S., Xie, T.: Unitplus: assisting developer testing in eclipse. In: Eclipse 2007: Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology Exchange, New York, NY, USA, pp. 26–30 (2007)
23. Taylor, R.N.: Assertions in programming languages. SIGPLAN Not. 15(1), 105–114 (1980)
24. Tillmann, N., de Halleux, J.: Pex–White Box Test Generation for .NET. In: Beckert, B., Hähnle, R. (eds.) TAP 2008. LNCS, vol. 4966, pp. 134–153. Springer, Heidelberg (2008)
25. Vanoverberghe, D., de Halleux, J., Tillmann, N., Piessens, F.: State coverage: Software validation metrics beyond code coverage - extended version (2011), http://www.cs.kuleuven.be/publicaties/rapporten/cw/CW610.abs.html
26. Zhu, H., Hall, P.A.V., May, J.H.R.: Software unit test coverage and adequacy. ACM Comput. Surv. 29, 366–427 (1997)

Factorization for Component-Interaction Automata

Nikola Beneš*, Ivana Černá**, and Filip Štefaňák

Faculty of Informatics, Masaryk University, Brno, Czech Republic

Abstract. Component-interaction automata is a verification oriented formalism devised to be general enough to capture important aspects of component interaction in various kinds of component systems. A factorization problem naturally arises in formalisms that are based on composition. In general, the factorization problem may be presented as finding a solution X to the equation $M \mid X \simeq S$, where $\mid$ is a composition and $\simeq$ a behavioural equivalence. In our framework, the equivalence is the weak bisimulation and composition is parametrized. We provide a solution for the factorization problem which is built on top of the approach of Qin and Lewis [13].

1 Introduction

Developing correct and efficient computer systems is an intrinsically difficult task and the component-based development approach is no exception to this rule. Formal methods prove to be an invaluable ally in the effort to achieve this uneasy goal. They have their place in all parts of the development process, be it the specification stage or the verification stage.

To be able to employ formal methods, we need a formalism to describe the kind of systems we wish to work with. One of such formalisms is the modelling language of *component-interaction automata*, first presented in [2]. This formalism, as its name suggests, focuses on the properties of interaction between components. It has been devised with two main goals in mind: (a) to be able to model component interaction in various kinds of component systems with various kinds of component assembly, and (b) to support various formal methods of analysis, most notably formal verification of temporal properties. Component-interaction automata model component assembly using a flexible form of composition, which can be parametrized by a set of interactions that are feasible/infeasible in the system. This helps to model component-based systems quite precisely, even though the formalism itself is rather simple [4,8,17].

A problem that naturally arises in every formalism that is composition-oriented is the following. Suppose we have some kind of specification describing the desired system we are attempting to develop. Let us call this specification S. Assume further that a part, a component of the system is already built, call

* The author has been supported by Czech Grant Agency, grant No. GD102/09/H042.
** The author has been supported by Czech Grant Agency, grant No. GAP202/11/0312.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 554–565, 2012.

this component M. We then might ask the following question: "Is there any component that can be composed with M in order to obtain a system equivalent to the specification S?" This problem is called the *factorization problem* and the solution to this problem may be seen as an inverse to the composition operation.

In this paper, we are interested in solving the factorization problem in connection with component-interaction automata. In our version of the problem, both the system specification and the given component are assumed to be described in the component-interaction automata language. The equivalence in the factorization problem question is a version of the weak bisimulation. This allows S to be a high-level specification without describing any details of internal communication. Another advantage of using this kind of equivalence is that it preserves a wide range of temporal logics (see e.g. [1]). Our solution to this problem is built on top of the approach of Qin and Lewis [13]. They provide the solution of the factorization problem for finite CCS processes, which they call *finite-state machines*, using a version of CCS parallel composition and the standard weak bisimulation. Our situation is more complex, as the formalism of component-interaction automata employs a generalized notion of composition which is parametrized to allow for different wiring of the components. In our approach, the solution to the factorization problem is not only a component but also a description of the way it should be connected to the existing component of the system.

Our main contribution is an algorithm solving the factorization problem for component-interaction automata. In the structural part of the factorization construction we make use of the original algorithm of Qin and Lewis and provide a two way transformation from component-interaction automata into finite-state machines and back. The transformation itself is not completely straightforward, as we need to take the parametrized composition into account. The reason we choose this approach instead of a modification of the original algorithm is that such a formulation would be too complicated, adding little value to the presentation. On the other hand, the use of transformation leaves us enough space to focus on on the question of the composition parameter, which is not present in the finite-state machines model.

Related work. The factorization problem and its relatives have been studied in various contexts and using various formalisms. We have already mentioned the work by Qin and Lewis [13], which extends previous work done by Shields in [16]. A similar problem (solving CCS process equations with weak bisimulation) has been tackled by [11], yielding a semi-automatic procedure that is not well suited for automatic verification. Solutions to process equations with *strong* bisimulation have been fully described in [7]. This approach even yields a compact description of all possible solutions in the form of a *disjunctive modal transition system*, which is an extension to previously defined *modal transition systems* [6]. The comparison of these approaches together with a discussion about complexity may be found in [5]. Moreover, when using *deterministic* modal transition systems as the formalism for describing components, the factorization problem (here called the residual) has been also recently tackled in [14].

The factorization problem may be also seen as a special kind of a synthesis problem. Another kind of a synthesis problem is that where the specification of desired behaviour is not given as a model but as a logic formula. An example of this is the approach of [12], which produces a system satisfying a given LTL formula (for the definition of LTL, see e.g. [1]). Recently, in [9] this approach has been extended to synthesis from a given library of components. Another problem related to that of factorization is the substitutability problem, investigated e.g. in [15]. This problem concerns the question whether a given component in a given system may be substituted with another component. In the framework of component-interaction automata, this problem has been already tackled in [3]. Note, however, that although the substitutability and the factorization problems are similar, they cannot be reduced to each other.

2 Preliminaries

In this section, we present basic definitions that will be used throughout the paper. We start with the definition of labelled transition systems and weak bisimulation on labelled transition systems. The other formalisms (component-interaction automata, finite-state machines) will then follow as extension of labelled transition systems. The advantage of this approach will be seen further in Section 4 as the transformations presented there will be based on modifications of the underlying labelled transition systems, preserving their state space.

Definition 1. *A* labelled transition system *(LTS) is a tuple* $(Q, q_0, L, \rightarrow)$ *where* Q *is a set of states,* $q_0 \in Q$ *is the initial state,* L *is a set of labels, and* $\rightarrow \subseteq Q \times L \times Q$ *is a transition relation. We write* $q \xrightarrow{\ell} q'$ *instead of* $(q, \ell, q') \in \rightarrow$. *We say that an LTS is* finite *if* Q *and* L *are finite. We say that an LTS is* deterministic *if for all* $q \in Q$ *and* $\ell \in L$ *there is at most one* q' *such that* $q \xrightarrow{\ell} q'$.

In the next definition, we use the following notation: Let $U \subseteq L$ be a set of given *unobservable labels*. We then use $q \xrightarrow{\hat{\alpha}} r$ to denote that either $\alpha \notin U$ and $q \xrightarrow{\alpha} r$, or $\alpha \in U$ and $q = r$. We further use $q \stackrel{\hat{\alpha}}{\Rightarrow} r$ to denote that $q \xrightarrow{\beta_1} \cdots \xrightarrow{\beta_n} \xrightarrow{\hat{\alpha}} \xrightarrow{\gamma_1} \cdots \xrightarrow{\gamma_m} r$, where all $\beta_i, \gamma_j \in U$. We then define weak bisimulation with respect to a given set of unobservable labels as a modification of the original definition of [10].

Definition 2. *Let* S, T *be two LTS,* $S = (Q_S, q_0^S, L, \rightarrow_S)$ *and* $T = (Q_T, q_0^T, L, \rightarrow_T)$. *Let* $U \subseteq L$ *be a set of* unobservable labels. *A relation* $R \subseteq Q_S \times Q_T$ *is a* weak bisimulation with respect to U *if for each* $(p^S, p^T) \in R$ *the following holds:*

- *If* $p^S \xrightarrow{\alpha}_S q^S$ *then there is some* q^T *such that* $p^T \stackrel{\hat{\alpha}}{\Rightarrow}_T q^T$ *and* $(q^S, q^T) \in R$.
- *If* $p^T \xrightarrow{\alpha}_T q^T$ *then there is some* q^S *such that* $p^S \stackrel{\hat{\alpha}}{\Rightarrow}_S q^S$ *and* $(q^S, q^T) \in R$.

We say that $p^S \in Q_S$ *and* $p^T \in Q_T$ *are* weakly bisimilar with respect to U, *denoted as* $p^S \approx_U p^T$ *if there exists a weak bisimulation* R *with respect to* U *such that* $(p^S, p^T) \in R$. *We say that* S *and* T *are weakly bisimilar with respect to* U $(S \approx_U T)$ *if* $q_0^S \approx_U q_0^T$.

Component-Interaction Automata. The formalism of component-interaction automata was introduced in [2] for description of components and their interactions. Informally, a component-interaction automaton is a finite state LTS with structured labels that capture three types of communication – input, output and internal communication. The formalism is equipped with parametrized composition operator which allows for various kinds of composition.

In order to define component-interaction automata formally, we first introduce the following notation: Let Act and P be arbitrary sets. We use the symbol L^P_{Act} to denote

$$L^P_{Act} = ((P \cup \{-\}) \times Act \times (P \cup \{-\})) \setminus (\{-\} \times Act \times \{-\}),$$

where $-$ is a special symbol not in P. The elements of L^P_{Act} are called *component-interaction labels* (CI labels). Every CI label of the form $(r, a, -)$ is called an *output label*, every CI label of the form $(-, a, s)$ is called an *input label*, the remaining CI labels are called *internal*. The input and output labels are also called *external labels*. The internal labels are used to model internal behaviour of a component. It can also represent communication between some of the constituents of a composite component. The external labels are then used to model the readiness of the component to communicate with the environment, thus representing the services that are required or provided by the component.

The following definition is an extension of the original definition of [2]. The original definition uses a notion of *component names* and each component-interaction automaton is equipped with a *hierarchy* of component names, representing its constituent components. The hierarchy can be degenerate, i.e. contain only one component name, which means that the automaton represents a single primitive component. The component names then are used in the structured labels (see r and s) above. We extend this approach by adding a more fine-grained notion of a *port*. Every primitive component may be equipped with one or more (but finitely many) ports. A component name is thus identified with a set of ports. The ports are then used in the structured labels instead of the component names. Clearly, this modification is a strict extension of the original formalism, as any primitive component of the original definition can be seen as a component with just one port.

Definition 3. *A* component-interaction automaton *(CI automaton) is a tuple* $(Q, q_0, Act, \rightarrow, \mathbb{H})$ *where* Act *is a finite set of* actions, $\mathbb{H}$ *is a hierarchy of component names, where each component name is a finite set of* ports *and* $(Q, q_0, L^P_{Act}, \rightarrow)$ *is a finite LTS, where* P *is the union of all sets of ports found in the component names of* $\mathbb{H}$.

For a CI automaton $\mathcal{C} = (Q, q_0, Act, \rightarrow_{\mathcal{C}}, \mathbb{H})$ with P being the set of all ports occurring in $\mathbb{H}$ we further define the set of all *reachable labels* in $\mathcal{C}$ as

$$\begin{aligned} Reach(\mathcal{C}) = \{\ell \in L^P_{Act} \mid\ & \exists k \in \mathbb{N}, \exists q_1, \ldots, q_k \in Q, \\ & \exists \ell_1, \ldots, \ell_{k-1} \in L^P_{Act} : \\ & q_0 \xrightarrow{\ell_1}_{\mathcal{C}} q_1 \xrightarrow{\ell_2}_{\mathcal{C}} \cdots \xrightarrow{\ell_{k-1}}_{\mathcal{C}} q_{k-1} \xrightarrow{\ell}_{\mathcal{C}} q_k\} \end{aligned}$$

Two component-interaction automata are said to be *composable* if the sets of all ports found in the respective automata's hierarchy are disjoint. This is again in accordance with the original definition from [2] which required disjoint component names. We define the (parametrized) composition of two composable automata as follows. The definition is a slight modification of the original definition of $\otimes^{\mathcal{F}}$ in [17], as we only define composition of two automata. Also, the requirement that all internal transitions of the original automata are preserved is made implicit in this definition instead of imposing further requirements on $\mathcal{F}$.

Definition 4. *Let $\mathcal{C}_1 = (Q_1, q_1, Act_1, \rightarrow_1, \mathbb{H}_1)$ and $\mathcal{C}_2 = (Q_2, q_2, Act_2, \rightarrow_2, \mathbb{H}_2)$ be two composable CI automata. Let P_1 and P_2 be the set of all ports occurring in $\mathbb{H}_1$ and $\mathbb{H}_2$, respectively. Let $\mathbb{H} = (\mathbb{H}_1, \mathbb{H}_2)$ and let P be set of all ports occurring in $\mathbb{H}$ (thus $P = P_1 \cup P_2$). Let further $Act = Act_1 \cup Act_2$ and let $\mathcal{F} \subseteq L^P_{Act}$ be a set of* feasible labels. *We define the composition of $\mathcal{C}_1$ and $\mathcal{C}_2$ with respect to $\mathcal{F}$, denoted by $\mathcal{C}_1 \otimes^{\mathcal{F}} \mathcal{C}_2$, as $\mathcal{C}_1 \otimes^{\mathcal{F}} \mathcal{C}_2 = (Q_1 \times Q_2, (q_1, q_2), Act, \rightarrow, \mathbb{H})$, where $(p_1, p_2) \xrightarrow{\alpha} (p'_1, p'_2)$ if and only if $\alpha \in \mathcal{F} \cup (P_1 \times Act_1 \times P_1) \cup (P_2 \times Act_2 \times P_2)$ and one of the following four conditions holds:* (1) $p_1 \xrightarrow{\alpha}_1 p'_1$ *and* $p'_2 = p_2$, *or* (2) $p_1 = p'_1$ *and* $p_2 \xrightarrow{\alpha}_2 p'_2$, *or* (3) $\alpha = (s, a, r)$, $p_1 \xrightarrow{(s,a,-)}_1 p'_1$, $p_2 \xrightarrow{(-,a,r)}_2 p'_2$, *or* (4) $\alpha = (s, a, r)$, $p_1 \xrightarrow{(-,a,r)}_1 p'_1$, $p_2 \xrightarrow{(s,a,-)}_2 p'_2$.

When talking about weak bisimulation in connection with a CI automaton, we use the weak bisimulation on its underlying LTS, taking $U = P \times Act \times P$ (internal actions) as the set of unobservable labels. Whenever we speak about weakly bisimilar CI automata in the following, we shall use $\approx$ instead of $\approx_U$, as U is always clear from the context.

Finite-State Machines. We now present the definition of finite-state machines, their composition and bisimulation, as defined in [13]. Finite-state machines are basically finite-state processes of CCS [10] with composition that contains an implicit restriction of all synchronization actions. A finite-state machine is thus a finite LTS equipped with a finite set of action symbols Σ and a special symbol $\tau \notin \Sigma$.

Definition 5. *A* finite-state machine *(FSM) is a tuple $(Q, q_0, \Sigma, \rightarrow)$, where $(Q, q_0, \Sigma \cup \{\tau\}, \rightarrow)$ is a finite LTS and $\tau \notin \Sigma$ is a special* invisible action *symbol. We define $\hat{\Sigma} = \Sigma \cup \{\bar{u} \mid u \in \Sigma\}$. We further assume that $\bar{\bar{u}} = u$ for every $u \in \Sigma$.*

Definition 6. *([13]) Let $M_1 = (Q_1, q_1, \Sigma_1, \rightarrow_1)$ and $M_2 = (Q_2, q_2, \Sigma_2, \rightarrow_2)$ be two FSM. We define the composition of M_1 and M_2, denoted as $M_1 \parallel M_2$, as $M_1 \parallel M_2 = (S \times T, (q_1, q_2), \Sigma, \rightarrow)$, where $\Sigma = (\Sigma_1 \cup \Sigma_2) \setminus (\hat{\Sigma}_1 \cap \hat{\Sigma}_2)$ and $\rightarrow$ is defined as follows:*

- *If $p_1 \xrightarrow{\alpha}_1 p'_1$, and $\alpha \in (\Sigma \cup \tau)$, then $(p_1, p_2) \xrightarrow{\alpha} (p'_1, p_2)$ for all $p_2 \in Q_2$.*
- *If $p_2 \xrightarrow{\alpha}_2 p'_2$, and $\alpha \in (\Sigma \cup \tau)$, then $(p_1, p_2) \xrightarrow{\alpha} (p_1, p'_2)$ for all $p_1 \in Q_1$.*
- *If $p_1 \xrightarrow{\alpha}_1 p'_1$, and $p_2 \xrightarrow{\bar{\alpha}}_2 p'_2$, then $(p_1, p_2) \xrightarrow{\tau} (p'_1, p'_2)$.*

When talking about weak bisimulation on FSM, we use the weak bisimulation on its underlying LTS, taking $U = \{\tau\}$. In the following, we shall thus use $\approx$ instead of $\approx_{\{\tau\}}$ when talking about weakly bisimilar FSM.

Note: In the following, we shall use calligraphic letters (such as $\mathcal{A}$, $\mathcal{C}$) for CI automata while using ordinary roman letters (such as A, C) for FSM.

3 The Factorization Problem

We start this section with a reminder about the general factorization problem, followed by a formal definition of the factorization problem for CI automata and a general outline of our solution. We then proceed with a short discussion on the factorization problem solved in [13].

The general factorization problem is as follows. Suppose we have a modelling formalism that supports some kind of composition of models (denoted as $|$) and is equipped with a relation that represents behavioural equivalence (denoted as $\simeq$). Let then S be a model of the desired system and M be a model of an already build component of the desired system. The factorization problem is then to find some X such that $M \mid X \simeq S$.

In the context of CI automata, the problem is a bit more complex. As the composition operator for CI automata is parametrized by $\mathcal{F}$, a set of feasible labels, we require the solution to not only consist of a CI automaton $\mathcal{X}$, but also such set $\mathcal{F}$. This requirement on the solution is natural in the framework of CI automata, as the parameter $\mathcal{F}$ represents the connection of components. Having one component $\mathcal{A}$, we therefore look for a second component $\mathcal{X}$ together with the way of connecting it to $\mathcal{A}$, which is represented by $\mathcal{F}$.

Formally, the definition of our factorization problem is as follows:

input: CI automata $\mathcal{A}$ and $\mathcal{C}$ such that $\mathcal{C}$ is deterministic and $\mathit{Reach}(\mathcal{C})$ contains no internal labels.
output: A CI automaton $\mathcal{X}$ and a set of CI labels $\mathcal{F}$ such that $\mathcal{A} \otimes^{\mathcal{F}} \mathcal{X} \approx \mathcal{C}$.

The condition that $\mathit{Reach}(\mathcal{C})$ contain no internal labels is a natural one, as the CI automaton $\mathcal{C}$ is supposed to describe a high level specification of the desired system. Such specification may pose no constraints on the internal interaction of its actual implementation. It only describes the communication with the environment, i.e. the required and provided services. The determinism requirement on $\mathcal{C}$ is then a prerequisite to using the approach of [13] as the basis for our approach. We are not aware of a solution to the factorization problem using weak bisimulation that would allow for nondeterministic specification.

The general outline of the solution to this problem is the following: We first transform the CI automata $\mathcal{A}$ and $\mathcal{C}$ into FSM, preserving their structure. The two FSM will then be used as an input to the approach of [13], resulting in another FSM. This FSM is then to be transformed back into a CI automaton $\mathcal{X}$, which will be the solution to our original problem. Note that this general plan does not address the problem of finding the desired composition parameter $\mathcal{F}$.

This task, together with presenting the transformations, will be tackled in the next section. Note that there is one necessary condition on $\mathcal{F}$ that is already clear, namely that $\mathcal{F} \supseteq \mathit{Reach}(\mathcal{C})$.

Before we come to the next section, which provides the transformation from CI automata to FSM and back, we shortly discuss the solution to the factorization problem for FSM. In context of FSM in [13], the composition is $\|$ and the behavioural equivalence is $\approx$, the weak bisimulation. The factorization problem is then to find some FSM X such that $A \parallel X \approx C$ where A, C are given FSM. The paper [13] then present a solution of this problem, under certain conditions, namely that C is deterministic and *rigid*, i.e. it does not contain any τ-transitions.

The solution provided by [13] is furthermore the most general solution in the following sense. Let the provided solution be called R. Then every other solution to the factorization problem is *strongly included* in R, where the definition of strong inclusion is provided below. Note that the provided solution may contain a special state called the DON'T-CARE state. It is guaranteed that the composition with A will never enter such state.

Definition 7. *([13]) A binary relation T on states is a* strong inclusion relation *if the following holds: Whenever $(p, q) \in T$ and $p \xrightarrow{u} p'$ then either (a) $q \xrightarrow{u}$ DON'T-CARE or (b) $p \neq$ DON'T-CARE, $q \xrightarrow{u} q'$ and $(p', q') \in T$. We say that R* strongly includes *R' if there exists a strong inclusion relation T such that $(q_0, q_0') \in T$ where q_0 and q_0' are the initial state of R and R', respectively.*

4 From CI Automata to FSM and Back

In this section, we shall often use assumptions of the form "Let $\mathcal{F}$ be a set of CI labels." If no further information about $\mathcal{F}$ is given, such an assumption simply means that $\mathcal{F}$ may be an arbitrary set of triples, where either the first or the third element may be the special symbol $-$, but not both. We also ignore the hierarchy of component names and only work with the ports occurring in the hierarchy, as the hierarchy does not influence the solution to the factorization problem. We shall thus write $\mathcal{C} = (Q, q_0, \mathit{Act}, \rightarrow, P)$ instead of using: "$\mathcal{C} = (Q, q_0, \mathit{Act}, \rightarrow, \mathbb{H})$ where P is the set of ports found in $\mathbb{H}$". This allows us to simplify some of the reasoning and reduces the complexity of the presentation in this section.

We introduce functions $f : \mathit{CI} \rightarrow \mathit{FSM}$ and $g_{\mathcal{F}} : \mathit{CI} \rightarrow \mathit{FSM}$ for every set of CI labels $\mathcal{F}$ such that $\mathcal{A} \otimes^{\mathcal{F}} \mathcal{B} \approx \mathcal{C} \iff g_{\mathcal{F}}(\mathcal{A}) \parallel g_{\mathcal{F}}(\mathcal{B}) \approx f(\mathcal{C})$. The f and g transformations are going to preserve the set of states and the initial state of the underlying LTS. Only the transitions are going to change.

The first transformation is the f transformation, which is to be applied on the system represented by $\mathcal{C}$. The transformation changes all internal labels into τ, leaving the rest intact.

Definition 8. *Let $\mathcal{C} = (Q, q_0, \mathit{Act}, \rightarrow_{\mathcal{C}}, P)$. A FSM $f(\mathcal{C})$ is defined as $f(\mathcal{C}) = (Q, q_0, \Sigma, \rightarrow)$, where $\Sigma = (P \times \mathit{Act} \times \{-\}) \cup (\{-\} \times \mathit{Act} \times P)$, for $\alpha \in \Sigma$, $q \xrightarrow{\alpha} q'$ if and only if $q \xrightarrow{\alpha}_{\mathcal{C}} q'$, and $q \xrightarrow{\tau} q'$ if and only if $q \xrightarrow{(r,a,s)}_{\mathcal{C}} q'$ where $r, s \in P$.*

In order to define the g transformations, we need the following definition of a few auxiliary sets.

Definition 9. *Let $\mathcal{C} = (Q, q_0, Act, \rightarrow, P)$ be a CI automaton, let $\mathcal{F}$ be a set of CI labels. We define:*

- $ext^{\mathcal{C}}_{\mathcal{F}} = \{(r, a, -) \in \mathcal{F} \mid r \in P\} \cup \{(-, a, s) \in \mathcal{F} \mid s \in P\}$. *This is the set of all external labels of $\mathcal{C}$ that are allowed by $\mathcal{F}$.*
- $out^{\mathcal{C}}_{\mathcal{F}} = \left\{\overline{(r, a, s)} \mid (r, a, s) \in \mathcal{F}, r \in P, s \notin P \cup \{-\}\right\}$.
  This is the set of all synchronization labels in $\mathcal{F}$ such that $\mathcal{C}$ can be the outputting automaton in the synchronization.
- $in^{\mathcal{C}}_{\mathcal{F}} = \{(r, a, s) \mid (r, a, s) \in \mathcal{F}, s \in P, r \notin P \cup \{-\}\}$.
  Similarly to previous, this is the set of all synchronization labels in $\mathcal{F}$ such that $\mathcal{C}$ can be the inputting automaton in the synchronization.
- $other^{\mathcal{C}}_{\mathcal{F}} = \{(r, a, s) \in \mathcal{F} \mid (r, s \notin P \cup \{-\}) \vee (r, s \in P)\}$. *This is the set of all labels of $\mathcal{F}$ that cannot be created by synchronization, if $\mathcal{C}$ is one of the partners of the synchronization. They are thus either internal labels of $\mathcal{C}$ or they are internal labels with both ports different from those of $\mathcal{C}$.*

Note that the four sets $ext^{\mathcal{C}}_{\mathcal{F}}$, $out^{\mathcal{C}}_{\mathcal{F}}$, $in^{\mathcal{C}}_{\mathcal{F}}$, $other^{\mathcal{C}}_{\mathcal{F}}$ may be seen as a partition of $\mathcal{F}$, except for the fact that all labels in $out^{\mathcal{C}}_{\mathcal{F}}$ are modified with a line above. This is to ensure that $\overline{(r, a, s)}$ and (r, a, s) may synchronize in the FSM that is constructed below.

We now define the g transformation. The transformation is parametrized with $\mathcal{F}$, a set of CI labels. The transformation (i) changes all internal labels to τ and (ii) multiplies all external labels with all possibilities of synchronization.

Definition 10. *Let $\mathcal{F}$ be a finite set of CI labels, let $\mathcal{C} = (Q, q_0, Act, \rightarrow_{\mathcal{C}}, P)$. We define a FSM $g_{\mathcal{F}}(\mathcal{C})$ as $g_{\mathcal{F}}(\mathcal{C}) = (Q, q_0, \Sigma, \rightarrow)$, where*

- $\Sigma = ext^{\mathcal{C}}_{\mathcal{F}} \cup in^{\mathcal{C}}_{\mathcal{F}} \cup out^{\mathcal{C}}_{\mathcal{F}} \cup other^{\mathcal{C}}_{\mathcal{F}}$,
- *for all* $\alpha \in ext^{\mathcal{C}}_{\mathcal{F}}$, $q \xrightarrow{\alpha} q'$ *iff* $q \xrightarrow{\alpha}_{\mathcal{C}} q'$,
- $q \xrightarrow{\tau} q'$ *iff* $q \xrightarrow{(r,a,s)}_{\mathcal{C}} q'$ *for some* $(r, a, s) \in (P \times Act \times P)$,
- *for all* $\overline{(r, a, s)} \in out^{\mathcal{C}}_{\mathcal{F}}$, $q \xrightarrow{\overline{(r,a,s)}} q'$ *iff* $q \xrightarrow{(r,a,-)}_{\mathcal{C}} q'$,
- *for all* $(r, a, s) \in in^{\mathcal{C}}_{\mathcal{F}}$, $q \xrightarrow{(r,a,s)} q'$ *iff* $q \xrightarrow{(-,a,s)}_{\mathcal{C}} q'$.

Example 1. Let $\mathcal{F} = \{(1, a, 3), (1, a, 4), (2, a, 4)\}$ and let $\mathcal{A}$ be a CI automaton with a transition $p \xrightarrow{(1,a,-)}_{\mathcal{A}} p'$ and $\mathcal{B}$ a CI automaton with a transition $q \xrightarrow{(-,a,4)}_{\mathcal{B}} q'$. Then $g_{\mathcal{F}}$ transforms $\mathcal{A}$ into a FSM with transitions $p \xrightarrow{\overline{(1,a,3)}}_{g(\mathcal{A})} p'$ and $p \xrightarrow{\overline{(1,a,4)}}_{g(\mathcal{A})} p'$ and $\mathcal{B}$ into a FSM with transitions $q \xrightarrow{(1,a,4)}_{g(\mathcal{B})} q'$ and $q \xrightarrow{(2,a,4)}_{g(\mathcal{B})} q'$. The composition of these two FSM is then allowed to synchronize on $(1, a, 4)$, producing the internal transition $(p, q) \xrightarrow{\tau} (p', q')$ in $g_{\mathcal{F}}(\mathcal{A}) \parallel g_{\mathcal{F}}(\mathcal{B})$.

We now show that the f and g transformation preserve composition and weak bisimulation and thus satisfy the requirement given at the beginning of the current section. Due to space constraints, the proofs of all lemmata are omitted.

Lemma 1. *Let $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ be CI automata such that $\mathcal{A}$ and $\mathcal{B}$ are composable. Let $\mathcal{F}$ be a set of CI labels such that $\mathcal{F} \supseteq Reach(\mathcal{C})$. Then the following holds: $\mathcal{A} \otimes^{\mathcal{F}} \mathcal{B} \approx \mathcal{C} \iff g_{\mathcal{F}}(\mathcal{A}) \parallel g_{\mathcal{F}}(\mathcal{B}) \approx f(\mathcal{C})$.*

We thus know that our transformations f and g preserve composition and weak bisimulation. Moreover, we need the transformation g to be reversible, which is not possible in the general case. We shall, however, provide a sufficient condition for $\mathcal{F}$ that ensures reversibility of g. We then show that this condition does not restrict the possibility of finding a solution to the factorization problem.

Definition 11. *Let P' be a set of ports and $\mathcal{F}$ an arbitrary set of CI labels. We say that $\mathcal{F}$ is* label injective *with respect to P' if the following statements hold:*

- $\forall r \in P', x, y \notin P' : (r, a, x), (r, a, y) \in \mathcal{F} \implies x = y$
- $\forall s \in P', x, y \notin P' : (x, a, s), (y, a, s) \in \mathcal{F} \implies x = y$

Note that x, y may be $-$.

We first show that label injectivity of $\mathcal{F}$ implies reversibility of $g_{\mathcal{F}}$.

Lemma 2. *Let $A = (Q, q_0, \Sigma, \rightarrow_A)$ be a FSM such that Σ is a set of CI labels. Let P' be a set of ports such that for every $(r, a, s) \in \Sigma$, $s \in P'$ and for every $\overline{(r, a, s)} \in \Sigma$, $r \in P'$. Further, let $\mathcal{F}$ be a set of CI labels that is label injective with respect to P'. Then there exists a CI automaton $\mathcal{A}'$ with set of ports P' such that $g_{\mathcal{F}}(\mathcal{A}') = A$.*

Example 2. Indeed, without the requirement of label injectivity, the inverse of $g_{\mathcal{F}}$ may not exist. Take a FSM A with only one state q and only one transition $q \xrightarrow{\overline{(1,a,2)}}_A q$ and let $\mathcal{F} = \{(1, a, 2), (1, a, 3)\}$. It can be clearly seen that there is no CI automaton $\mathcal{C}$ such that $g_{\mathcal{F}}(\mathcal{C}) = A$.

The following lemma states that if there exists a solution to the factorization problem, there exists a solution with label injective $\mathcal{F}$.

Lemma 3. *Let $\mathcal{F}'$, $\mathcal{X}'$ be a solution to the factorization problem. Then there exists a solution $\mathcal{F}$, $\mathcal{X}$ such that $\mathcal{F}$ is label injective with respect to the set of ports of $\mathcal{X}$.*

We now know that we can focus our attention to solutions $\mathcal{F}$, $\mathcal{X}$ such that $\mathcal{F}$ is label injective with respect to the set of ports of $\mathcal{X}$. From Section 3, we know that a necessary condition for $\mathcal{F}$ is that $\mathcal{F} \supseteq Reach(\mathcal{C})$. Combining these two conditions is, however, not yet sufficient for a solution to exist. Indeed, take $\mathcal{F} = Reach(\mathcal{C})$. This $\mathcal{F}$ is always label injective with respect to any set of ports, yet disallows all communication in the composition. We thus define a new condition imposed on $\mathcal{F}$ that, together with the two conditions already mentioned, will be sufficient.

Definition 12. *Let $\mathcal{A}$ be a CI automaton with set of ports $P_{\mathcal{A}}$. Let $\mathcal{F}$ be an arbitrary set of CI labels. We say that $\mathcal{F}$ is complete with respect to $\mathcal{A}$ if for every $(r, a, -) \in Reach(\mathcal{A})$ there is some $(r, a, y) \in \mathcal{F}$ with $y \notin P_{\mathcal{A}}$, $y \neq -$ and for every $(-, a, s) \in Reach(\mathcal{A})$ there is some $(x, a, s) \in \mathcal{F}$ with $x \notin P_{\mathcal{A}}$, $x \neq -$.*

Lemma 4. *Let $\mathcal{A}$, $\mathcal{C}$ be as in the definition of the factorization problem. Let $\mathcal{F} \supseteq Reach(\mathcal{C})$ be complete with respect to $\mathcal{A}$ and label injective with respect to all ports not in $P_{\mathcal{A}}$. Let further $\mathcal{F}'$, $\mathcal{X}'$ be a label injective solution to the factorization problem. Then we may find a CI automaton $\mathcal{X}$ such that $\mathcal{F}$, $\mathcal{X}$ is also a solution to the factorization problem.*

5 The Algorithm

Before we can proceed with explaining the final algorithm solving the factorization problem, we need to show that there always exists a set $\mathcal{F}$ satisfying the sufficient conditions given by Lemma 4. Let $\mathcal{A}$, $\mathcal{C}$ be the instance of the problem and let $P_{\mathcal{A}}$ be the set of ports of $\mathcal{A}$. Recall that the sufficient conditions are: (i) $\mathcal{F} \supseteq Reach(\mathcal{C})$, (ii) $\mathcal{F}$ is complete with respect to $\mathcal{A}$, and (iii) $\mathcal{F}$ is label injective with respect to all ports not in $P_{\mathcal{A}}$. A trivial $\mathcal{F}$ satisfying these conditions is given as follows: Let for each $r \in P_{\mathcal{A}}$, r' be a unique new port not contained in $P_{\mathcal{A}}$. We then set $\mathcal{F} = Reach(\mathcal{C}) \cup \{(r, a, r') \mid (r, a, -) \in Reach(\mathcal{A})\} \cup \{(s', a, s) \mid (-, a, s) \in Reach(\mathcal{A})\}$. Our solution to the factorization problem for CI automata thus works as follows.

input: CI automata $\mathcal{A}$ and $\mathcal{C}$ such that $\mathcal{C}$ is deterministic and $Reach(\mathcal{C})$ contains no internal labels.

1. Let $\mathcal{F} = Reach(\mathcal{C}) \cup \{(r, a, r') \mid (r, a, -) \in Reach(\mathcal{A})\} \cup \{(s', a, s) \mid (-, a, s) \in Reach(\mathcal{A})\}$
2. Transform $\mathcal{A}$ into $g_{\mathcal{F}}(\mathcal{A})$ and $\mathcal{C}$ into $f(\mathcal{C})$.
3. Solve the factorization problem for FSM $g_{\mathcal{F}}(\mathcal{A})$ and $f(\mathcal{C})$ using the algorithm from [13]. Let B denote the solution. (If no solution exists, claim that no solution for the original problem exists.)
4. Using the construction from the proof of Lemma 2 produce a CI automaton $\mathcal{B}$ such that $g_{\mathcal{F}}(\mathcal{B}) = B$.
5. Output $\mathcal{F}$, $\mathcal{B}$ as the solution of the original problem.

The correctness of this algorithm is a corollary to all previous lemmata.

Theorem 1. *The algorithm is correct and complete, i.e. whenever it produces an output, it is a correct solution, and if there exists a solution, it produces one.*

The complexity of the algorithm is determined by the complexity of Qin and Lewis' construction. It may be easily seen that all our transformations are linear in the size of the transformed models. Therefore, even though we have a more complex formalism, finding solution to the factorization problem in our approach retains the complexity of [13].

Figure 1 illustrates the result of the algorithm. Note that the solution also contains transitions to a DON'T-CARE state, which we omitted here for clarity. It is easy to verify that indeed $\mathcal{A} \otimes^{\mathcal{F}} \mathcal{B} \approx \mathcal{C}$. Note that although the solution $\mathcal{B}$ has three ports $1'$, $2'$, and $3'$, a solution with two ports also exists. We could, for example, rename $1'$ to $2'$ everywhere in $\mathcal{B}$'s labels and thus obtain a solution with ports $2'$ and $3'$. However, no solution with just one port exists.

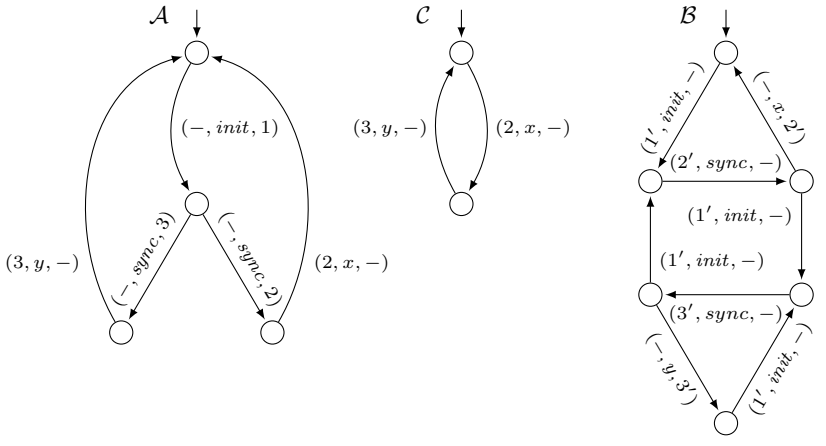


Fig. 1. An instance $(\mathcal{A}, \mathcal{C})$ of the factorization problem and its solution, $\mathcal{B}$

The solution provided by the algorithm is a CI automaton whose number of ports may be in the worst case equal to the number of ports in $\mathcal{A}$. However, it may be the case that it is desirable to produce a CI automaton with as few ports as possible. The algorithm can be augmented with the following heuristic, lowering the number of ports of the result.

Let P_a^+ be the set of ports r such that $(r, a, -) \in \mathit{Reach}(\mathcal{A})$, let P_a^- be the set of ports s such that $(-, a, s) \in \mathit{Reach}(\mathcal{A})$. Let then m be defined as $m = \max\{\max_{a \in Act} |P_a^+|, \max_{a \in Act} |P_a^-|\}$. Clearly, for each a there exists an injective function $i_a^+ : P_a^+ \to \{1, \ldots, m\}$ and an injective function $i_a^- : P_a^- \to \{1, \ldots, m\}$. The improved parameter $\mathcal{F}$ is then defined as:

$$F = \mathit{Reach}(\mathcal{C}) \cup \{(r, a, i_a^+(r)) \mid (r, a, -) \in \mathit{Reach}(\mathcal{A})\} \cup \{(i_a^-(s), a, s) \mid (-, a, s) \in \mathit{Reach}(\mathcal{A})\}$$

We thus obtain a solution with only m ports.

6 Conclusion and Future Work

In this paper, we have tackled the factorization problem in connection with the framework of component-interaction automata. We have based our solution on a previous work by Qin and Lewis [13]. Our solution works by transforming the original problem into a form that is an input to the approach of [13]. The result of this approach is then transformed back into a component-interaction automaton, thus providing a solution to the original problem.

There are many possibilities of future development. One of them is port minimization. Our solution may produce a component-interaction automaton with many ports, even if much smaller number of ports would suffice. We provide a heuristic reducing the number of ports, but we suspect that a general approach

to finding the solution with the *minimal* number of ports may be a much harder problem. A related issue is that of finding the most general solution. The work [13] provides the most general solution in the sense of strong inclusion as explained in Section 3. This is, however, not very satisfactory result, as it is not complete. This means that although every solution is strongly included in the provided one, there may be systems which are strongly included in the provided one, yet are not solutions to the factorization problem. A better description of a most general solution would perhaps use modalities such as in [7,6].

References

1. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
2. Brim, L., Černá, I., Vařeková, P., Zimmerova, B.: Component-interaction automata as a verification-oriented component-based system specification. In: SAVCBS 2005, pp. 31–38. Iowa State University, USA (2005)
3. Černá, I., Vařeková, P., Zimmerova, B.: Component substitutability via equivalencies of component-interaction automata. ENTCS 182, 39–55 (2007)
4. Jia, Y., Li, Z., Zhang, Z.: Timed component-interaction automata for specification and verification of real-time reactive systems. In: CSSE 2008, vol. 2, pp. 135–138 (2008)
5. Jonsson, B., Larsen, K.G.: On the Complexity of Equation Solving in Process Algebra. In: Abramsky, S. (ed.) CAAP 1991 and TAPSOFT 1991. LNCS, vol. 493, pp. 381–396. Springer, Heidelberg (1991)
6. Larsen, K.G., Thomsen, B.: A modal process logic. In: LICS 1988, pp. 203–210. IEEE Computer Society (1988)
7. Larsen, K.G., Xinxin, L.: Equation solving using modal transition systems. In: LICS, pp. 108–117. IEEE Computer Society (1990)
8. Lumpe, M., Grunske, L., Schneider, J.G.: State Space Reduction Techniques for Component Interfaces. In: Chaudron, M.R.V., Ren, X.-M., Reussner, R. (eds.) CBSE 2008. LNCS, vol. 5282, pp. 130–145. Springer, Heidelberg (2008)
9. Lustig, Y., Vardi, M.Y.: Synthesis from Component Libraries. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 395–409. Springer, Heidelberg (2009)
10. Milner, R.: A Calculus of Communication Systems. LNCS, vol. 92. Springer, Heidelberg (1980)
11. Parrow, J.: Submodule construction as equation solving in ccs. Theor. Comput. Sci. 68(2), 175–202 (1989)
12. Pnueli, A., Rosner, R.: On the synthesis of a reactive module. In: POPL 1989, pp. 179–190. ACM (1989)
13. Qin, H., Lewis, P.: Factorization of Finite State Machines Under Observational Equivalence. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 427–441. Springer, Heidelberg (1990)
14. Raclet, J.B.: Residual for component specifications. ENTCS 215, 93–110 (2008)
15. Sharygina, N., Chaki, S., Clarke, E.M., Sinha, N.: Dynamic Component Substitutability Analysis. In: Fitzgerald, J.S., Hayes, I.J., Tarlecki, A. (eds.) FM 2005. LNCS, vol. 3582, pp. 512–528. Springer, Heidelberg (2005)
16. Shields, M.W.: Implicit system specification and the interface equation. The Computer Journal 32(5), 399–412 (1989)
17. Zimmerova, B., Vařeková, P., Beneš, N., Černá, I., Brim, L., Sochor, J.: Component-Interaction Automata Approach (CoIn). In: Rausch, A., Reussner, R., Mirandola, R., Plášil, F. (eds.) The Common Component Modeling Example. LNCS, vol. 5153, pp. 146–176. Springer, Heidelberg (2008)

Optimizing Segment Based Document Protection

Mirosław Kutyłowski and Maciej Gębala

Institute of Mathematics and Computer Science,
Wrocław University of Technology
{miroslaw.kutylowski,maciej.gebala}@pwr.wroc.pl

Abstract. We consider documents with restricted access rights, where some *segments* of the document are encrypted in order to prevent unauthorized reading. The access rights to such a document are described by an *access graph*. It is a directed acyclic graph; each node describing a different access rights level. It is assumed that a user having the rights corresponding to a node v has also all rights corresponding to all nodes w such that there is a directed path from v to w in the access graph. Then, to each node v we assign a key K_v and use this key to encrypt the segment of the document corresponding to the access level v.

We consider key management schemes and encoding auxiliary information in the document which ensure that a user who gets a single key corresponding to his access level v can derive all keys K_w for $w = v$ or w being an ancestor of v in the access graph.

In this paper we show how to minimize the total size of auxiliary keying information stored in the document. We provide an algorithm based on node disjoint paths in the access graph and key derivation based on one-way functions. We show that the algorithm chooses the paths in an optimal way.

Keywords: document protection, access rights, key management, key hierarchy, directed acyclic graph.

1 Introduction

Digitalization of information flow requires in particular creating electronic documents as a counterpart of paper documents and assigning them similar legal role as for paper documents. This process is inevitable, but creates a lot of problems in the transition period. They have not only legal and social background, there are new technical challenges as well.

One of the key technical problems is that once a digital document is created, it can be easily copied (with each copy indistinguishable from the original) and freely distributed. This is not the case for paper documents where one can control the location of a document and where there is only one original of the document. The ease of copying electronic documents becomes a problem since in some areas (like court procedures and personal data protection) there are strict rules about granting rights to read documents.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 566–575, 2012.

Access to an electronic document might be guarded by the operating system: it assigns access rights to each document and afterwards grants access to a document for a given user as long as it is explicitly admitted by access rights. This method is satisfactory as long as a the system has a small size and all entitled readers are using the same computer. However, when we are talking about a large distributed system such an approach leads to a very tedious distributed rights management.

In this situation cryptography may contribute a lot into practical usability: we can encrypt a document (or its appropriate parts) and make it easily available (by posting on the Web, or broadcasting). Then it suffices to distribute the decryption keys to the entitled users. The decryption keys may be delivered in different ways; for instance the users might have hardware tokens with secrets shared by document issuer and can generate decryption keys on demand. Another option are broadcast protocols, where secret material (i.e. decryption keys) is encoded so that only certain recipients may derive it.

1.1 Segment Based Document Protection

Encryption of electronic documents brings also some advantages: instead of using a single key for all sensitive parts one can use different keys to different parts. In this situation different parties may get access to different parts of the same document, while integrity of the document and its the encrypted parts might be secured with means of electronic signatures. Such a policy of separation of roles and limiting the scope of user's view in case of sensitive documents is an idea gaining popularity (e.g. for medical data systems).

The idea of securing multiple parts of a document in different ways has appeared in many contexts in the literature; for recent application oriented work see [1, 2]. The components of the system are the following:

- the document contents as well as auxiliary right management information are encoded in an XML-structure,
- apart from plaintext contents, the document is divided into segments, each segment encrypted with a dedicated key,
- the dependencies between segments are described by an *access graph*, which is a directed acyclic graph (dag); in the access graph, if there is a directed path from a node labelled A to a node labelled B, then a user having the right to decrypt segment A has also the right to decrypt segment B,
- the XML-structure contains public *key information* on decryption keys; it enables a user that knows the key for a given segment A to derive the key of any segment B such that there is a directed path from A to B in the access graph.

Obviously, the access graph must be acyclic in order to derive a hierarchy-like rights management. Many authors talk about an ordering induced by the access

graph: if there is a directed path from node A to node B, we say that $A \succeq B$. The relation $\succeq$ need not to be a linear ordering. In many target applications it is only a partial order.

The question considered in this paper is what auxiliary information concerning the keys has to be stored inside the document in order to provide a solution that is computation and space efficient as well as flexible and scalable.

1.2 Hierarchical Key Structures and Key Derivation Techniques

The idea of efficient key management appeared at early days of modern cryptography. The idea of [3] is to provide each user with a single key so that all keys corresponding to the user's rights can be derived locally by the user.

Linear Schemes. If the access graph defines a linear ordering, say $A_1 \succeq A_2 \succeq \ldots \succeq A_m$, then we may derive the corresponding keys with any one-way function F (see e.g. [4]). Then we choose a key K_1 at random and for $i = 2, \ldots, m$ derive

$$K_i := F(K_{i-1}) \tag{1}$$

Then we assign the key K_i to node A_i, for $i \leq m$.

Obviously, having K_i it is easy to derive the keys $K_{i+1}, \ldots, K_m$ by applying formula (1). On the other hand, an attempt to derive K_{i-1} from K_i is an attempt to derive a preimage of a one-way function. By definition of one-way functions, this is infeasible.

In a practical setting we may use a cryptographic hash function H in place of F. Then we rely upon preimage resistance of the hash function H.

Tree Schemes. If the access graph is a tree, then we can apply a similar solution (see e.g. [5]). Namely, if a node A has child nodes $B_1, \ldots B_k$, and a key K has been assigned to A, then we assign the keys $K_1 \ldots K_k$ to, respectively $B_1, \ldots B_k$ using the formula

$$K_i := F_i(K) \quad \text{for } i \leq k \tag{2}$$

where $F_1, \ldots F_k$ are distinct and unrelated one-way functions. For this scheme we require that there is a easy to determine linear ordering of children of each node.

In a practical setting we may use a cryptographic hash function H instead of $F_1, \ldots F_k$. Namely, we put:

$$K_i := H(i, K) \quad \text{for } i \leq k\ . \tag{3}$$

However, then we rely upon a nonstandard assumption about infeasibility of derivation K_i based on K_j for $j \neq i$, when they are derived by the above formula. However, even if we use distinct one-way functions, we have to prove that one-wayness still holds, if an attacker gets not only a value of a single one-way function $F_i(K)$, but a collection of values $F_1(K), \ldots, F_k(K)$ derived from the sought value K.

Arbitrary Posets. From the early days of hierarchical key management, basically two techniques have been used. The first one, proposed in [3] is based on the strong RSA assumption, but in fact can be used in any multiplicative algebraic structure, where exponentiation is easy but computing roots of a given degree is infeasible (e.g. see [6]). The idea is as follows. First for a given poset P describing the access rights find a mapping $\rho : P \to \mathbb{N}$ such that for any nodes u, v we have

$$A \succeq B \quad \text{iff} \quad \rho(A)|\rho(B)$$

Then choose an element g at random in a given algebraic structure and compute

$$K_A := g^{\rho(A)}$$

as the key for node A. Of course, if $\rho(A)|\rho(B)$, then one can compute K_B from K_A due to equality $K_B = K_A^{\rho(B)/\rho(A)}$. On the other hand, computing K_A from K_B would mean computing root of degree $\rho(B)/\rho(A)$, which is infeasible.

The advantage of the scheme described is that each user has to store a single key and only the mapping need to be described in a document – no key information is necessary in the document. The main disadvantage of this scheme is computational complexity and relatively low flexibility.

The second approach is to use an arbitrary tree scheme (see e.g. [7] and [2]). Of course, this leads to conflicts when a node v has more than one incoming arcs. As the keys in successor nodes are computed by one-way functions, in general there would be more than one value derived for the key corresponding to node v. In order to deal with this problem we assign a kind of offset for each arc of the access graph. Namely, if the key of node v should be K_v and the computation for an arc (u, v) yields K', then we may define the offset as K' XOR K_v. The offset is the essential part of public information corresponding to the arc.

1.3 Document Encoding and Problem Statement

After making choice over hierarchical key management scheme we have to include relevant informations in the segmented document encoding described in Sect. 1.1. Namely, we have to describe

- key derivation method for each arc of the access graph,
- offset, if it is necessary for a given arc.

The mechanisms described so far have the following space requirements related to key information fields:

- If we use a hierarchical key management that can support arbitrary access graph, then no public key information need to be attached to the arcs of the access graph. Consequently, space overhead of the document encoding is low.
- If the tree key management is used, then key information might be eliminated from relatively many arcs of the access graph. Namely, we embed a tree into

the access graph and assign the keys within the embedded tree according to the tree key management method.[1] For the remaining arcs we define the offsets. If the access graph differs from a tree by just a few arcs, then only those arcs must be given offset information.
– If we use a linear hierarchy of keys, then, in order to save space, we have to choose node disjoint paths that together contain as many arcs as possible. Clearly, in general the remaining arcs (outside these paths) require public key information to be encoded in a document.

So we see that from the point of view of size expansion of a document, the first option is the most advantageous. However, we should be aware of the following problems that make the final implementation decision quite uneasy:

– In the first case, specific algebraic constructions must be applied. Security of encoding depends solely on security of a given algebraic scheme. Moreover, the schemes are based on asymmetric techniques, which are heavy from computational point of view. For this reason they might be hard to implement on certain consumer devices, while on the other hand the encoding has to be as portable as possible. However, the most important issue is lack of flexibility and a specific dependence on the order of the group applied.
– Tree schemes may be based on a wide range of techniques, among them on standard hash functions and symmetric encryption. For this reason computational requirements are significantly lower. On the other hand, underlying schemes must be secure in a specific model where the keys produced for siblings of a node are derived from the same secret (plus some public parameters) and therefore are related in some cryptanalytic sense.

For these reasons we feel that tree and linear schemes might be interesting for practical applications. This point of view is shared by many authors constructing such schemes.

If we decide to use a tree scheme or a linear scheme for key derivation in a dag, we are faced with the following problems:

Problem 1. Given a dag G, embed some number of trees in G so that

– *the embedded trees are node disjoint,*
– *the number of arcs that do not belong to any of embedded trees is minimal.*

Problem 2. Given a dag G, embed some number of paths in G so that

– *the embedded paths are node disjoint,*
– *the number of arcs that do not belong to any of embedded paths is minimal.*

Note that if we are given an embedding such as mentioned in Problem 1 or Problem 2, then within each embedded tree (respectively, path), we can separately

[1] Recall that ϕ is an embedding of a directed graph G_1 in a directed graph G_2, if ϕ is a bijection from the mapping the vertices of G_1 to vertices of G_2, and if (v, w) is an arc in G_1, then $(\phi(v), \phi(w))$ is an arc in G_2.

derive the keys starting from a single root key. Due to node disjointness this does not lead to any conflict. Consequently, in the access graph all arcs that correspond to embedded arcs do not require offset key information. Only for the remaining arcs we need offset key information.

2 Tree-Based Key Derivation

Let us apply the following algorithm:

Algorithm 1.

Input: a dag G
Output: a subgraph G' of G that consists a set of disjoint trees and containing the maximal possible number of arcs

Algorithm:
Construct a reduced graph G', a subgraph of G, in the following way:
for each node $v \in G$ of indegree greater than 1 pick up an arbitrary arc with endpoint v and remove all other arcs with endpoint v.

Let us observe that in G' each node has indegree at most 1. Therefore G' is a forest, i.e. a graph consisting of some number of disjoint trees. The shape of the forest depends very much on the choices made by Algorithm 1, however we may observe that this does not influence quality of the solution from our point of view:

Proposition 1. *The number of arcs in G' is constant and does not depend on algorithm execution. Moreover, no forest embedded in G may contain more arcs than G'.*

Proof. First observe that the choice made for a node v does not influence the choices at other nodes. Indeed, removing arcs pointing to v does not change the arc pointing to w for $w \neq v$.

If for node v there are m_v incoming arcs, then the total number of arcs removed during algorithm execution equals always $\sum_{v \in G} \max(0, m_v - 1)$, and therefore the number of arcs in G' is always the same.

For the second part of Proposition 1 observe that if F is a forest and a subgraph of G, then for each node v of F, let $\mathrm{in}(v)$ denote an arc of F with the endpoint v (if there such an arc), or undefined (if there is no such an arc). Then we construct graph G' in the following way: for a given v we leave the arc $\mathrm{in}(v)$ (if it is defined), or an arbitrary arc with endpoint v (if $\mathrm{in}(v)$ is undefined). Clearly, in this way we do not remove any arc of F and therefore all arcs of F are contained in G'. So the number of arcs in F is not higher than the number of arcs in G'. □

By Proposition 1 we see that determining a forest, a subgraph of an access graph G, with the maximal number of arcs is straightforward.

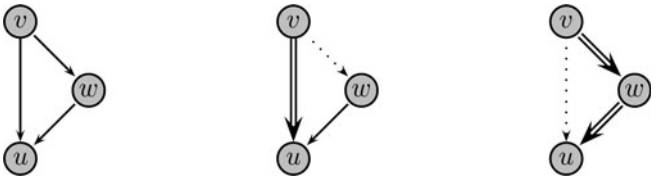


Fig. 1. Let us consider a dag consisting of arcs (v, u), (v, w) and (w, u) - a triangle with source v and sink u. If during a greedy procedure we take the arc (v, u) (or remove (v, w)), then the final solution will contain just one path with a single arc. On the other hand, the optimal solution is the path $(v, w), (w, u)$.

3 Sequential Key Derivation

First let us observe that unlike in the case of constructing a maximal subforest, determining the maximal set of node disjoint paths cannot be done in a greedy way. The simplest example is presented on Figure 1.

Before we present an algorithm constructing the optimal solution for Problem 2, let us make the following observation:

Lemma 1. *Let d be the maximal number of arcs in a subgraph of a dag G so that the subgraph consists of node disjoint paths. Let v be a node of G with indegree 0. Then there is a subgraph S' of G consisting of node disjoint paths having d arcs and containing an arc starting at v pointing to an arbitrary of its immediate ancestors.*

Proof. Let S be a subgraph of a dag G, where S consists of node disjoint paths containing in total d arcs.

Let v be a node of G with indegree 0 such that no arc starting in v belongs to S (otherwise the claim of Lemma 1 is already fulfilled for v). Consider all arcs $(v, u_1), \ldots, (v, u_k)$ in G that originate in v. According to our assumption, none of them belongs to S. If any node u_i does not belong to S, then we could add the arc (v, u_i) to S. The resulting graph would be still composed of node disjoint paths (with a new path (v, u_i) of length 1), but would have one more arc than S, contrary to the assumption that S is optimal. In the same way we show that each of the nodes u_i must have a predecessor in S. Indeed, otherwise adding the arc (v, u_i) would extend the path that originates in u_i.

Now, consider the node u_1. Assume that w is a predecessor of u_1 in S. We construct a new graph S' by removing the arc (w, u_1) from S and adding the arc (v, u_1). As S' has the same number of arcs as S, the graph S' is optimal, too. □

Below we present a recursive algorithm called TOP_DOWN_SELECT (Algorithm 2) that is based on Lemma 1.

Now we argue that TOP_DOWN_SELECT outputs an optimal solution.

Lemma 2. *An output S of TOP_DOWN_SELECT is a graph consisting of node disjoint paths.*

Algorithm 2. TOP_DOWN_SELECT

Input: a dag G

Output: a subgraph S of G that consists of node disjoint paths and containing the maximal possible number of arcs

Algorithm:

1. select an arbitrary node v in G of indegree 0,
2. select an arbitrary arc (v, u) in G,
3. construct a graph G' by removing from G:
   - node v and all its outgoing arcs,
   - all arcs with the endpoint u.
4. run TOP_DOWN_SELECT for graph G', yielding an output S',
5. compose S by adding the arc (v, u) to S',
6. output S.

If G contains at most one arc, then output this arc.

Proof. It suffices to prove that each node in S has indegree at most 1 and outdegree at most 1.

The proof is by induction on the number of arcs of graph G. For a graph with one or no arc the algorithm obviously provides the optimal solution.

So let us consider an inductive step and a graph G. For G, the algorithm TOP_DOWN_SELECT constructs a solution S' for a graph G' with a smaller number of arcs. So by induction hypothesis, S' consists of node disjoint paths.

When the arc (v, u) is added to S', then indegree and outdegree requirements may be violated at most by u and v. First let us consider v. Since its indegree is 0 in G, it will be 0 in S as well (S is a subgraph of G). For outdegree, observe that there is no arc starting at v left in G', so no arc starting at v may occur in S' (recall that S' is a subgraph of G'). Hence outdegree of v is 1.

Now consider node u. Its outdegree in S is the same as in S', hence at most 1. For indegree, let us recall that according to the construction there is no arc pointing to u in G'. Therefore the indegree in S is 1. □

Lemma 3. *An output S of TOP_DOWN_SELECT contains the optimal number of arcs.*

Proof. Assume that the optimal set of node disjoint paths in G contains d arcs. By Lemma 1, there is a solution Z with d arcs such that (u, v) selected at the second step of TOP_DOWN_SELECT belongs to it. Let Z' be the graph obtained from Z by removing the arc (v, u). It is easy to observe that Z' is contained in the graph G'. Indeed, any arc in $G \setminus G'$ would cause violation of the indegree or outdegree rule for u or v.

Z' contains $d-1$ arcs, but we do not know a priori if it is an optimal solution for G'. However, we know that the number of arcs in S' is at least $d-1$, since S' is optimal in G'. So S contains at least $d-1+1=d$ arcs. So S contains the maximal number of arcs d. □

Let us observe that the algorithm TOP_DOWN_SELECT not only derives the optimal solution, but is very efficient.

BOTTOM_UP_SELECT alternative. One can reverse the ordering and start to fix arcs not at source nodes but from sink nodes. Below we present such a dual version of algorithm TOP_DOWN_SELECT:

Algorithm 3. BOTTOM_UP_SELECT

Input: a dag G
Output: a subgraph S of G that consists of node disjoint paths and containing the maximal possible number of arcs

Algorithm:

1. select an arbitrary node v in G of outdegree 0,
2. select an arbitrary arc (u, v) in G,
3. construct a graph G' by removing from G:
   - node v and all its ingoing arcs,
   - all arcs with the startpoint u.
4. run BOTTOM_UP_SELECT for graph G', yielding an output S',
5. compose S by adding the arc (u, v) to S',
6. output S.

If G contains at most one arc, then output this arc.

Lemma 4. *Algorithm BOTTOM_UP_SELECT outputs the optimal solution: a subgraph consisting of node disjoint paths with the maximal number of arcs.*

Proof. Consider a graph G^{-1} obtained from G by reversing each arc. The claim of Lemma 4 follows from observation that BOTTOM_UP_SELECT for graph G runs exactly as TOP_DOWN_SELECT for G^{-1}, and that the number of arcs in the optimal solutions are the same for G and G^{-1}. □

4 Open Problems

In some application areas it might be useful to generate subdocuments of a given document truncated to those segments that are readable by some specific group of users. In this case we consider a subgraph $\bar{G}$ of access right graph G with the following property: if v is a vertex in $\bar{G}$, then all arcs of the form (v, u) from G are also arcs of $\bar{G}$. Therefore, a node v from $\bar{G}$ has the same ancestors in $\bar{G}$ as in G. If D is an encoded document with access graph G, we construct a new document D' such that:

- D' contains only those segments that correspond to the nodes of $\bar{G}$, all remaining segments are removed,
- D' contains only those key information fields that correspond to the arcs of $\bar{G}$.

When truncated versions of a document D are concerned, then optimization of key derivation scheme has to be reconsidered.

Tree-based Scheme. Given a node v in G of indegree higher than 1, Algorithm 1 selects one of arc with endpoint v. For the generalized scheme we need a optimize the choice taking into account statistical data about access rights. Namely, to each node v in G we associate frequency parameter p_v which describes the fraction of users that are assigned the access rights corresponding to v. Then for each arc (u, v) in G we compute the weight which is the sum of p_u and frequency parameters p_w for all predecessors of u in G. Modified Algorithm 1 selects an arc with endpoint v with the highest weight. One can easily see that this is an optimal solution, if every user is getting a truncated version of the document D. Indeed, such a choice leads to elimination of key information for arcs pointing to v for the highest number of users.

Linear Scheme. The argument used for tree-based schemes cannot be extended directly to Algorithm 2. The reason is that selection of an arc based on Lemma 1 does not change the number of arcs in the optimal solution, but may change a lot the shape of all paths in the final solution. Since each decision has global consequences, it is unclear how to make an optimal choice.

Our first impression is that it is more reasonable to optimize Algorithm 3 than 2. A promising heuristic is to choose always an arc leading to a node with outdegree 0 that is used by the highest number of users. However, it remains an open problem how to compute an optimal solution.

References

1. Qiu, R., Tang, Z., Gao, L., Yu, Y.: A novel XML-based document format with printing quality for web publishing. In: Imaging and Printing in a Web 2.0 World; and Multimedia Content Access: Algorithms and Systems IV. Proc. SPIE, vol. 7540. Society of Photographic Instrumentation Engineers (2010)
2. Xu, D., Tang, Z., Yu, Y.: An efficient key management scheme for segment-based document protection. In: 2011 IEEE Consumer Communications and Networking Conference (CCNC), pp. 896–900 (2011)
3. Akl, S.G., Taylor, P.D.: Cryptographic solution to a problem of access control in a hierarchy. ACM Trans. Comput. Syst. 1(3), 239–248 (1983)
4. Levin, L.A.: The tale of one-way functions (2003), http://arxiv.org/abs/cs.CR/0012023 (retrieved on May 20, 2011)
5. Hassen, H.R., Bouabdallah, A., Bettahar, H.: A new and efficient key management scheme for content access control within tree hierarchies. In: AINA Workshops, vol. (1), pp. 551–556. IEEE Computer Society (2007)
6. Wu, J., Wei, R.: An Access Control Scheme for Partially Ordered Set Hierarchy with Provable Security. In: Preneel, B., Tavares, S.E. (eds.) SAC 2005. LNCS, vol. 3897, pp. 221–232. Springer, Heidelberg (2006)
7. Atallah, M.J., Blanton, M., Fazio, N., Frikken, K.B.: Dynamic and efficient key management for access hierarchies. ACM Trans. Inf. Syst. Secur. 12(3) (2009)

Securing the Future — An Information Flow Analysis of a Distributed OO Language*

Martin Pettai[1,2] and Peeter Laud[1,2]

[1] University of Tartu
[2] Cybernetica AS

Abstract. We present an information-flow type system for a distributed object-oriented language with active objects, asynchronous method calls and futures. The variables of the program are classified as high and low. We allow while cycles with high guards to be used but only if they are not followed (directly or through synchronization) by an assignment to a low variable. To ensure the security of synchronization, we use a high and a low lock for each concurrent object group (cog). In some cases, we must allow a high lock held by one task to be overtaken by another, if the former is about to make a low side effect but the latter cannot make any low side effects. This is necessary to prevent synchronization depending on high variables from influencing the order of low side effects in different cogs. We prove a non-interference result for our type system.

1 Introduction

The question of information security arises when the inputs and outputs of a program are partitioned into different security classes. In this case we want the high-security inputs not inappropriately influence the low-security outputs and other behaviour observable at low clearance. The strongest such property is non-interference [9] stating that there is no influence at all; or that variations in the high-security inputs do not change the observations at the low level.

Over the years, static analyses, typically type systems for verifying secure information flow have been proposed for programs written in many kinds of programming languages and paradigms — imperative or functional, sequential or parallel, etc. Each new construct in the language can have a profound effect on the information flows the programs may have. With the spread of distributed computing and multi-core processors, concurrent object-oriented programming is gaining mindshare. The languages supporting this paradigm emphasize the greater independence of objects and various methods of communication between the objects and the concurrently running tasks. The effect these constructs have

* The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 231620, from Estonian Science Foundation through grants No. 7543 and 8124, and from the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 576–587, 2012.

on the information flow has not yet been thoroughly investigated. On the one hand, the communication primitives are prone to introducing information leaks. On the other hand, the explicit independence of objects and their groups can provide clear evidence that certain flows are missing.

In this paper, we investigate a particular concurrent object-oriented language, related to Creol [12] and JCoBoxes [21]. The language, its syntax and semantics is almost the same as the concurrent OO sublanguage of the *core Abstract Behavioural Specification Language (ABS)* [10] and shall henceforth be named as this. At the core of this language lies the notion of a *concurrent object group* (cog). Different cogs run concurrently and independently of each other and communicate only via asynchronous method calls. When placing such a call, a *future* [8] (a placeholder for the eventually available return value) is immediately returned. A future admits certain operations for checking the presence of and reading the return value. Inside a cog, there may also be several running tasks sharing some common state (the fields of the object). In contrast, these tasks are scheduled cooperatively, such that there is always just a single active task per cog.

In this paper we propose a type system for checking the non-interference in programs written in ABS. For eliminating certain information flows, and for simplifying the checks we will fix or adjust certain details of the language in a manner that can be seen as non-essential for its purposes (specifying concurrent systems). Compared to [10], our language has a more fine-grained system of locks for controlling which task is currently running inside a cog. We have also restricted the scheduler for non-preemptive tasks, such that information flow properties are easier to enforce. The specification of scheduling decisions is made harder by the necessity to not introduce deadlocks into the program that were not there before. On the other hand, different cogs are running in a truly parallel fashion, scheduled nondeterministically.

We will introduce the syntax and semantics of ABS in Sect. 2. While describing the syntax, we will already introduce security types and annotations that form the basis for defining non-interference. In Sect. 3 we introduce our type system for secure information flow, and state the properties satisfied by well-typed programs. We review the related work in Sect. 4 and discuss our results in Sect. 5.

2 The Programming Language

2.1 Syntax

Our programming language is a simplified version of ABS. Its (abstract) syntax is given in Fig. 1. The notation $\overline{X}$ denotes a sequence of X-s. Several constructs in the syntax are annotated by security levels. These do not have to be provided by the programmer, as they can be inferred automatically during type checking.

Let us explain the language constructs related to distributed execution. $e_p!m(\overline{e_p})$ denotes the *asynchronous* call of the method m. The call immediately

$$
\begin{array}{rll}
x \mid n \mid o \mid b \mid f & & \text{local variable} \mid \text{task} \mid \text{object} \mid \text{cog} \mid \text{field name} \\
Pr ::= \overline{Cl}\ B & & \text{program} \\
Cl ::= \mathsf{class}\ C\{\overline{T\ f}\ \overline{M}\} & & \text{class definition} \\
M ::= (m : (l, \overline{T}) \overset{l[,i]}{\to} \mathsf{Cmd}^l(T))(\overline{x})\ B & & \text{method definition} \\
B ::= \{\overline{T\ x}\ s; x\} & & \text{method body} \\
v ::= x \mid \mathsf{this} \mid \mathsf{this}.f & & \text{variable} \\
i ::= \ldots \mid -1 \mid 0 \mid 1 \mid \ldots & & \text{integer} \\
e ::= e_p \mid e_s & & \text{expression} \\
e_p ::= v \mid \mathsf{null} \mid i \mid e_p = e_p & & \text{pure expression} \\
e_s ::= e_p !_l m(\overline{e_p}) \mid e_p.\mathsf{get}_l \mid \mathsf{new}\ C \mid \mathsf{new\ cog}\ C & & \text{expression with side effects} \\
s ::= v := e \mid e \mid \mathsf{skip} \mid \mathsf{suspend}_l \mid \mathsf{await}_l\ g & & \text{statement} \\
\quad \mid \mathsf{if}\ (e_p)\ s\ \mathsf{else}\ s \mid \mathsf{while}_l\ (e_p)\ s \mid s; s & & \\
g ::= v? & & \text{guard} \\
l ::= L \mid H & & \text{security level} \\
\ell ::= l \mid i & & \text{security level or integer} \\
T ::= \mathsf{Int}_l \mid C_l \mid \mathsf{Fut}_l^\ell(T) \mid \mathsf{Guard}_l^\ell & & \text{security type}
\end{array}
$$

Fig. 1. Syntax

returns a future. The get-construct is used to read the value of that future, if it is available. If not, then get blocks. The suspend-statement suspends the current thread, it is used for non-preemptive scheduling inside a cog. The statement await g suspends until the guard $g = v?$ becomes true, which happens when the future v obtains a value.

2.2 Operational Semantics

We first define run-time configurations. The program at run time is a set of cogs (concurrent object groups), each of which contains a set of objects. Each object is related to zero or more tasks. The run-time configurations are as follows:

$$P ::= b[n_1, n_2] \mid o[b, C, \sigma] \mid n\,\langle b, o, \sigma, s\rangle \mid P \parallel P$$

Each cog is represented by its identifier b and the state of its locks. Each cog has two locks—the low lock, which is owned by task n_1 (or is free if $n_1 = \bot$), and the high lock, which is owned by task n_2 (or is free if $n_2 = \bot$).

Each object is represented by its identifier o, its cog b, its class C, and its state σ (the values of its fields). Each task is represented by its identifier n, its cog b, its object o, the statement s that is yet to be executed in this task, and its state σ (the values of its local variables).

The run-time syntax will have some additional constructs:

$$e_p ::= \ldots \mid n \mid o \qquad s ::= \ldots \mid \mathsf{grab}_l \mid \mathsf{release}_l \qquad a ::= \mathsf{null} \mid i \mid n \mid o$$

Thus task and object identifiers can be used (these result from evaluating other expressions) and we will use a to denote fully reduced expressions (i.e. constants): Also separate statements are introduced for grabbing and releasing locks (used for executing suspend and when starting and terminating tasks).

The initial configuration for the program $\overline{Cl}\ \{\overline{T\,x}\ s; x_0\}$ will be

$$b_0[n_0, n_0] \parallel n_0\,\langle b_0, \mathsf{null}, \sigma, s; \mathsf{release}_L; x_0\rangle$$

i.e. an initial cog b_0 will be created for the task n_0 executing the main method. This task will have both locks initially and the statement $\mathsf{release}_L$ is added to release the locks before the task terminates. This task is the only task that is not tied to an object (all tasks created later will be tied to some object). The variable x_0 (which must have type Int_L) will contain the return value of the program. Input can be given to the program through the initial values of the variables in σ. These variables must be declared in the body of the main method.

Now we can give the reduction rules (including the necessary reduction contexts) in Fig. 2. Again, some explanations are in order. The commands grab and release manipulate the locks of a cog. Suspending a task is equivalent to releasing a lock and then trying to grab it again. A method body that starts with a grab is currently suspended. It is possible to perform either a low or a high suspend. When a task has performed a high suspend, then only other high-suspended tasks can continue.

An asynchronous call (acall) creates a new task in the cog containing the receiver of the call. The new task is initially suspended. The name of the new task is used as the future.

A while-loop suspends after each iteration. Hence an infinite loop cannot stop the computation in the entire cog and cause information flows through non-termination in such manner. The semantics of the await-command is straightforward, except for the rule (await_3). It is used to avoid certain deadlocks. See Sect. 3.1 for the definition of low and high-low tasks and further discussion. Basically, rule (await_3) allows the task n' to preemptively start running (and suspend the task n_1) if its final value is being waited for. In such manner, the possible non-termination of task n_1 cannot affect the termination behaviour of n (the high-low task n' always terminates).

3 Type System for Non-interference

3.1 Types

The types in the type system are the following:

$$T ::= \mathsf{Int}_l \mid C_l \mid \mathsf{Fut}^{\ell}_{l}(T) \mid \mathsf{Guard}^{\ell}_{l} \mid \mathsf{Exp}^{l}(T) \mid \mathsf{Cmd}^{l} \mid \mathsf{Cmd}^{l}(T) \mid (l, \overline{T}) \overset{l[,i]}{\to} \mathsf{Cmd}^{l}(T)$$

Thus we can have integers, objects of class C, futures, guards, possibly non-terminating expressions, commands, commands (method bodies) returning a value, and methods. Here the subscript represents the security level of the value.

$$R_1[e] ::= x := e \mid \mathsf{this}.f := e$$

$$R_2[e] ::= R_1[e] | \mathsf{if}\ (e)\ s_1\ \mathsf{else}\ s_2 | R_1[e.\mathsf{get}_l] | R_1[e!_l m(\overline{e'})] | R_1[e'!_l m(\overline{e_1}\, e\, \overline{e_2})] | R_2[e = e'] | R_2[e' = e]$$

$$\frac{n' \text{ fresh} \qquad \mathrm{body}(m) = s(\bar{x}); x' \qquad s_{task} = \mathsf{grab}_l; s[\bar{a}/\bar{x}]; \mathsf{release}_l; x'}{o'[b', C, \sigma'] \parallel n\,\langle b, o, \sigma, R_1[o'!_l m(\bar{a})]; s\rangle \rightsquigarrow o'[b', C, \sigma'] \parallel n\,\langle b, o, \sigma, R_1[n']; s\rangle \parallel n'\,\langle b', o', \sigma_{init}, s_{task}\rangle} \ (\mathsf{acall})$$

$$\frac{o' \text{ fresh}}{n\,\langle b, o, \sigma, R_1[\mathsf{new}\ C]; s\rangle \rightsquigarrow n\,\langle b, o, \sigma, R_1[o']; s\rangle \parallel o'[b, C, \sigma_{init}]} \ (\mathsf{new})$$

$$\frac{b' \text{ fresh} \qquad o' \text{ fresh}}{n\,\langle b, o, \sigma, R_1[\mathsf{new\ cog}\ C]; s\rangle \rightsquigarrow n\,\langle b, o, \sigma, R_1[o']; s\rangle \parallel b'[\bot, \bot] \parallel o'[b', C, \sigma_{init}]} \ (\mathsf{newcog})$$

$$\frac{}{n\,\langle b, o, \sigma', R_1[n'.\mathsf{get}_l]; s\rangle \parallel n'\,\langle b', o', \sigma, x\rangle \rightsquigarrow n\,\langle b, o, \sigma', R_1[\sigma(x)]; s\rangle \parallel n'\,\langle b', o', \sigma, x\rangle} \ (\mathsf{get}_1)$$

$$\frac{}{n\,\langle b, o, \sigma', R_1[n'.\mathsf{get}_l]; s\rangle \parallel n'\,\langle b', o', \sigma, s'; x\rangle \rightsquigarrow n\,\langle b, o, \sigma', \mathsf{await}_l(n'?); R_1[n'.\mathsf{get}_l]; s\rangle \parallel n'\,\langle b', o', \sigma, s'; x\rangle} \ (\mathsf{get}_2)$$

$$\frac{}{n\,\langle b, o, \sigma, R_2[x]; s\rangle \rightsquigarrow n\,\langle b, o, \sigma, R_2[\sigma(x)]; s\rangle} \ (\mathsf{var})$$

$$\frac{}{o[b, C, \sigma] \parallel n\,\langle b, o, \sigma', R_2[\mathsf{this}.f]; s\rangle \rightsquigarrow o[b, C, \sigma] \parallel n\,\langle b, o, \sigma', R_2[\sigma(f)]; s\rangle} \ (\mathsf{field})$$

$$\frac{}{n\,\langle b, o, \sigma, a; s\rangle \rightsquigarrow n\,\langle b, o, \sigma, s\rangle} \ (\mathsf{dummyexpr})$$

$$\frac{}{n\,\langle b, o, \sigma, x := a; s\rangle \rightsquigarrow n\,\langle b, o, \sigma[x \mapsto a], s\rangle} \ (\mathsf{assignvar})$$

$$\frac{}{o[b, C, \sigma] \parallel n\,\langle b, o, \sigma', \mathsf{this}.f := a; s\rangle \rightsquigarrow o[b, C, \sigma[f \mapsto a]] \parallel n\,\langle b, o, \sigma', s\rangle} \ (\mathsf{assignfield})$$

$$\frac{}{n\,\langle b, o, \sigma, \mathsf{skip}; s\rangle \rightsquigarrow n\,\langle b, o, \sigma, s\rangle} \ (\mathsf{skip})$$

$$\frac{}{n\,\langle b, o, \sigma, \mathsf{suspend}_l; s\rangle \rightsquigarrow n\,\langle b, o, \sigma, \mathsf{release}_l; \mathsf{grab}_l; s\rangle} \ (\mathsf{suspend})$$

$$\frac{}{b[\bot, \bot] \parallel n\,\langle b, o, \sigma, \mathsf{grab}_L; s\rangle \rightsquigarrow b[n, n] \parallel n\,\langle b, o, \sigma, s\rangle} \ (\mathsf{grab_L})$$

$$\frac{}{b[n', \bot] \parallel n\,\langle b, o, \sigma, \mathsf{grab}_H; s\rangle \rightsquigarrow b[n', n] \parallel n\,\langle b, o, \sigma, s\rangle} \ (\mathsf{grab_H})$$

$$\frac{}{b[n, n] \parallel n\,\langle b, o, \sigma, \mathsf{release}_L; s\rangle \rightsquigarrow b[\bot, \bot] \parallel n\,\langle b, o, \sigma, s\rangle} \ (\mathsf{release_L})$$

$$\frac{}{b[n', n] \parallel n\,\langle b, o, \sigma, \mathsf{release}_H; s\rangle \rightsquigarrow b[n', \bot] \parallel n\,\langle b, o, \sigma, s\rangle} \ (\mathsf{release_H})$$

$$\frac{i \neq 0}{n\,\langle b, o, \sigma, \mathsf{if}\ (i)\ s_1\ \mathsf{else}\ s_2; s\rangle \rightsquigarrow n\,\langle b, o, \sigma, s_1; s\rangle} \ (\mathsf{if_+})$$

$$\frac{}{n\,\langle b, o, \sigma, \mathsf{if}\ (0)\ s_1\ \mathsf{else}\ s_2; s\rangle \rightsquigarrow n\,\langle b, o, \sigma, s_2; s\rangle} \ (\mathsf{if_-})$$

$$\frac{}{n\,\langle b, o, \sigma, \mathsf{while}_l\ (e)\ s_1; s_2\rangle \rightsquigarrow n\,\langle b, o, \sigma, \mathsf{if}\ (e)\ (s_1; \mathsf{suspend}_l; \mathsf{while}_l\ (e)\ s_1)\ \mathsf{else}\ \mathsf{skip}; s_2\rangle} \ (\mathsf{while})$$

$$\frac{}{n\,\langle b, o, \sigma', \mathsf{await}_l(n'?); s\rangle \parallel n'\,\langle b', o', \sigma, x\rangle \rightsquigarrow n\,\langle b, o, \sigma', s\rangle \parallel n'\,\langle b', o', \sigma, x\rangle} \ (\mathsf{await}_1)$$

$$\frac{}{\begin{array}{l} n\,\langle b, o, \sigma', \mathsf{await}_l(n'?); s\rangle \parallel n'\,\langle b', o', \sigma, s'; x\rangle \rightsquigarrow \\ \rightsquigarrow n\,\langle b, o, \sigma', \mathsf{suspend}_l; \mathsf{await}_l(n'?); s\rangle \parallel n'\,\langle b', o', \sigma, s'; x\rangle \end{array}} \ (\mathsf{await}_2)$$

$$\frac{\text{the next step of } s_1 \text{ is low and the task } n' \text{ is high-low}}{\begin{array}{l} n\,\langle b, o, \sigma', \mathsf{await}_H(n'?); s\rangle \parallel n'\,\langle b', o', \sigma, \mathsf{grab}_H; s'; x\rangle \parallel n_1\,\langle b', o_1, \sigma_1, s_1\rangle \parallel b'[n_1, n_1] \rightsquigarrow \\ \rightsquigarrow n\,\langle b, o, \sigma', \mathsf{suspend}_H; \mathsf{await}_H(n'?); s\rangle \parallel n'\,\langle b', o', \sigma, s'; x\rangle \parallel n_1\,\langle b', o_1, \sigma_1, \mathsf{grab}_H; s_1\rangle \parallel b'[n_1, n'] \end{array}} \ (\mathsf{await}_3)$$

Fig. 2. Reduction rules

For integers and objects, this corresponds to the upper bound on the security levels of the inputs that may have affected this value. For futures and guards, this is the upper bound on the (control flow) information that may affect which task this future is referring to. The security level on top of the arrow of the method type corresponds to the minimum level of side effects this method is allowed to perform. If this is high, then the side effects of this method do not affect the low part of the computation. The level l_0 in the method type denotes the security level of the receiver of the method call (i.e., this-argument). The superscripts on the types are the upper bound on information that may affect whether this future, guard, expression, or command eventually returns a value or terminates. If this information is high, then the effects of any computation that follows are high, too.

We also define the security level corresponding to those security types that can be types of variables:

$$level(C_l) = l \qquad level(\mathsf{Int}_l) = l \qquad level(\mathsf{Fut}_l^{\ell'}(T)) = l \qquad level(\mathsf{Guard}_l^{\ell'}) = l$$

Thus $level(T)$ is the maximum context level where assignments to variables of type T are allowed.

The typing rules are given in Figures 3 and 4. The general shape of the typing

$$l \leq l \qquad L \leq H \qquad \frac{l_2 \leq l_1 \qquad \ell_3 \leq \ell_4}{\mathsf{Guard}_{l_1}^{\ell_3} \leq \mathsf{Guard}_{l_2}^{\ell_4}} \qquad \frac{l_2 \leq l_1 \qquad \ell_3 \leq \ell_4 \qquad T_5 \leq T_6}{\mathsf{Fut}_{l_1}^{\ell_3}(T_5) \leq \mathsf{Fut}_{l_2}^{\ell_4}(T_6)}$$

$$\mathsf{Guard}_H^i \leq \mathsf{Guard}_H^L \qquad \frac{l_1 \leq l_2}{C_{l_1} \leq C_{l_2}} \qquad \frac{l_1 \leq l_2}{\mathsf{Int}_{l_1} \leq \mathsf{Int}_{l_2}} \qquad \frac{\gamma, l \vdash e : T}{\gamma, l \vdash e : \mathsf{Exp}^L(T)}$$

$$\frac{\gamma, l \vdash e : T_1 \qquad T_1 \leq T_2}{\gamma, l \vdash e : T_2} \qquad \frac{\gamma, l \vdash s : \mathsf{Cmd}^{l_1} \qquad l_1 \leq l_2}{\gamma, l \vdash s : \mathsf{Cmd}^{l_2}} \qquad \frac{\gamma, l_1 \vdash s : \mathsf{Cmd}^{l} \qquad l_1 \geq l_2}{\gamma, l_2 \vdash s : \mathsf{Cmd}^{l}}$$

Fig. 3. Subtyping rules

rules is $\gamma, l \vdash X : T$, where γ is the typing context giving the types of local variables, fields, and methods, l is the current *security context* upper bounding the information that may have affected whether the execution reaches the current program point, X is a typable quantity and T is its type. For typing methods, there is no security context. Considering the meaning of sub- and superscripts in the types, the rules in Figures 3 and 4 should be rather straightforward. A program Pr is well typed if $\vdash Pr : ok$ is derivable.

We also allow an integer i to be added to the security level of the context. This is used to guarantee termination for methods (corresponding to high-low tasks in Def. 2) where the security level of the context is higher than the security level of termination and thus while cycles are forbidden. Cycles could still occur through cycles in the await graph and to disallow this, each of these methods has a positive integer $i > 0$ and can only await after a task with a smaller integer.

$$\frac{(\forall i)\ Cl_i = \mathsf{class}\ C_i\{T_{i1}\ f_{i1} \ldots T_{ir_i}\ f_{ir_i}\ M_{i1} \ldots M_{ik_i}\} \qquad (\forall i,j)\ M_{ij} = (m_{ij} : T'_{ij})(\overline{x_{ij}})\ B_{ij} \\ \gamma = \{C_i.f_{ij} \mapsto T_{ij} \mid 1 \le i \le n, 1 \le j \le r_i\} \cup \{C_i.m_{ij} \mapsto T'_{ij} \mid 1 \le i \le n, 1 \le j \le k_i\} \\ (\forall i,j)\ \gamma \vdash M_{ij} : T'_{ij} \qquad B = \{\overline{T\ x}\ s\} \qquad \gamma, \overline{x : T}, L \vdash s : \mathsf{Cmd}^L(\mathsf{Int}_L)}{\vdash Cl_1 \ldots Cl_n\ B : ok}\ (\mathsf{Prog})$$

$$\frac{\gamma, \overline{x : T}, \mathsf{this} : C_{l_0}, l \vdash s : \mathsf{Cmd}^{l_1}(T') \qquad l = l_1}{\gamma \vdash (m : (l_0, \overline{T}) \xrightarrow{l} \mathsf{Cmd}^{l_1}(T'))(\overline{x})\ \{\overline{T\ x}\ s\} : (l_0, \overline{T}) \xrightarrow{l} \mathsf{Cmd}^{l_1}(T')}\ (\mathsf{Method}_1)$$

$$\frac{\gamma, \overline{x : T}, \mathsf{this} : C_{l_0}, l, i \vdash s : \mathsf{Cmd}^{l_1}(T') \qquad l > l_1 \qquad i > 0}{\gamma \vdash (m : (l_0, \overline{T}) \xrightarrow{l,i} \mathsf{Cmd}^{l_1}(T'))(\overline{x})\ \{\overline{T\ x}\ s\} : (l_0, \overline{T}) \xrightarrow{l,i} \mathsf{Cmd}^{l_1}(T')}\ (\mathsf{Method}_2)$$

$$\frac{\gamma(x) = T}{\gamma, l \vdash x : T}\ (\mathsf{Var}) \qquad \frac{\gamma(C.f) \le T \qquad T \in \{C'_l, \mathsf{Int}_l, \mathsf{Fut}^{\ell_1}_l(T')\} \qquad \gamma, l_2 \vdash \mathsf{this} : C_l}{\gamma, l_2 \vdash \mathsf{this}.f : T}\ (\mathsf{Field})$$

$$\frac{}{\gamma, l \vdash \mathsf{null} : C_L}\ (\mathsf{Null}) \qquad \frac{}{\gamma, l \vdash i : \mathsf{Int}_L}\ (\mathsf{Int})$$

$$\frac{\gamma, l \vdash e : C_{l_0} \qquad \gamma, l \vdash \overline{e} : \overline{T} \qquad \gamma(C.m) = l_0, \overline{T} \xrightarrow{l} \mathsf{Cmd}^{l_1}(T_2) \qquad l_0 \ge l \qquad \overline{T} \ge l \qquad l_1 = l}{\gamma, l \vdash e!_l m(\overline{e}) : \mathsf{Fut}^{l_1}_l(l \vee l_1 \vee T_2)}\ (\mathsf{ACall}_1)$$

$$\frac{\gamma, l \vdash e : C_{l_0} \qquad \gamma, l \vdash \overline{e} : \overline{T} \qquad \gamma(C.m) = l_0, \overline{T} \xrightarrow{l,i} \mathsf{Cmd}^{l_1}(T_2) \qquad l_0 \ge l \qquad \overline{T} \ge l \qquad l_1 < l}{\gamma, l \vdash e!_l m(\overline{e}) : \mathsf{Fut}^{i}_l(l \vee l_1 \vee T_2)}\ (\mathsf{ACall}_2)$$

$$\frac{\gamma, l \vdash e : \mathsf{Fut}^{l_1}_l(T)}{\gamma, l \vdash e.\mathsf{get}_l : \mathsf{Exp}^{l_1}(T)}\ (\mathsf{Get}_1) \qquad \frac{\gamma, l, i \vdash e : \mathsf{Fut}^{i_1}_l(T) \qquad i_1 < i}{\gamma, l, i \vdash e.\mathsf{get}_l : \mathsf{Exp}^{L}(T)}\ (\mathsf{Get}_2)$$

$$\frac{}{\gamma, L \vdash \mathsf{new}\ C : C_L}\ (\mathsf{New}) \qquad \frac{}{\gamma, L \vdash \mathsf{new\ cog}\ C : C_L}\ (\mathsf{NewCog})$$

$$\frac{\gamma, l \vdash e : \mathsf{Exp}^{l_2}(T)}{\gamma, l \vdash e : \mathsf{Cmd}^{l_2}}\ (\mathsf{DummyExpr}) \qquad \frac{}{\gamma, l \vdash \mathsf{skip} : \mathsf{Cmd}^{L}}\ (\mathsf{Skip})$$

$$\frac{\gamma(x) = T \qquad level(T) = l \qquad \gamma, l \vdash e : \mathsf{Exp}^{l_2}(T)}{\gamma, l \vdash x := e : \mathsf{Cmd}^{l_2}}\ (\mathsf{AssignVar})$$

$$\frac{\gamma(C.f) = T \qquad level(T) = l \qquad \gamma, l \vdash e : \mathsf{Exp}^{l_2}(T) \qquad \gamma, l \vdash \mathsf{this} : C_l}{\gamma, l \vdash \mathsf{this}.f := e : \mathsf{Cmd}^{l_2}}\ (\mathsf{AssignField})$$

$$\frac{\gamma, l \vdash e : \mathsf{Guard}^{l_1}_l}{\gamma, l \vdash \mathsf{await}_l(e) : \mathsf{Cmd}^{l_1}}\ (\mathsf{Await}_1) \qquad \frac{\gamma, l, i \vdash e : \mathsf{Guard}^{i_1}_l \qquad i_1 < i}{\gamma, l, i \vdash \mathsf{await}_l(e) : \mathsf{Cmd}^{L}}\ (\mathsf{Await}_2)$$

$$\frac{}{\gamma, l \vdash \mathsf{grab}_l : \mathsf{Cmd}^L}\ (\mathsf{Grab}) \qquad \frac{}{\gamma, l \vdash \mathsf{release}_l : \mathsf{Cmd}^L}\ (\mathsf{Release}) \qquad \frac{}{\gamma, l \vdash \mathsf{suspend}_l : \mathsf{Cmd}^L}\ (\mathsf{Suspend})$$

$$\frac{\gamma, l \vdash e : \mathsf{Int}_l \qquad \gamma, l \vdash s_1 : \mathsf{Cmd}^{l_1} \qquad \gamma, l \vdash s_2 : \mathsf{Cmd}^{l_1}}{\gamma, l \vdash \mathsf{if}\ (e)\ s_1\ \mathsf{else}\ s_2 : \mathsf{Cmd}^{l_1}}\ (\mathsf{If}) \qquad \frac{\gamma, l \vdash e : \mathsf{Int}_l \qquad \gamma, l \vdash s : \mathsf{Cmd}^{l}}{\gamma, l \vdash \mathsf{while}_l\ (e)\ s : \mathsf{Cmd}^{l}}\ (\mathsf{While})$$

$$\frac{\gamma, l \vdash s_1 : \mathsf{Cmd}^{l_1} \qquad \gamma, l \vee l_1 \vdash s_2 : \mathsf{Cmd}^{l_2}}{\gamma, l \vdash s_1; s_2 : \mathsf{Cmd}^{l_1 \vee l_2}}\ (\mathsf{Seq}_1)$$

$$\frac{\gamma, l \vdash s_1 : \mathsf{Cmd}^{l_1} \qquad \gamma, l \vee l_1 \vdash s_2 : \mathsf{Cmd}^{l_2}(T)}{\gamma, l \vdash s_1; s_2 : \mathsf{Cmd}^{l_1 \vee l_2}(T)}\ (\mathsf{Seq}_2)$$

$$\frac{\gamma, l \vdash x : T \qquad level(T) = l}{\gamma, l \vdash x : \mathsf{Cmd}^L(T)}\ (\mathsf{ReturnVar}) \qquad \frac{\gamma, l' \vdash e : \mathsf{Fut}^{l_1}_l(T)}{\gamma, l' \vdash e? : \mathsf{Guard}^{l_1}_l}\ (\mathsf{Guard})$$

Fig. 4. Reduction rules

This makes the await graph of high-low tasks acyclic. To achieve this, we have some typing rules of the form $\gamma, l, i \vdash X : T$. The integer i can also be assimilated with γ. For example, a rule

$$\frac{\gamma, l, i \vdash s_1 : \mathsf{Cmd}^{l_1} \qquad \gamma, l \vee l_1, i \vdash s_2 : \mathsf{Cmd}^{l_2}}{\gamma, l, i \vdash s_1; s_2 : \mathsf{Cmd}^{l_1 \vee l_2}}$$

is considered a special case of the rule (Seq_1) and thus is not given separately. The integer i is also used (instead of L) in the superscript of the futures and guards of high-low tasks.

By the next definition, we can now distinguish high and low reduction steps, depending on whether the reduced statement is typable in high context or not.

Definition 1. *Let the statement s have the form $s_1; s_2$ where s_1 is not a sequential composition (because of associativity of the sequential composition operator, a statement always has either this form or the form x (a single variable, which cannot be further reduced)). We call the next reduction step of s a* high step *if $\gamma, H \vdash s_1 : \mathsf{Cmd}^l$ and a* low step *otherwise.*

The next definition allows to also distinguish high and low tasks. The previous and the next definition are used in the (await_3) rule in Fig. 2.

Definition 2. *We call a task $n\,\langle b, o, \sigma, s; x\rangle$ a* high task *if $\gamma, H \vdash s : \mathsf{Cmd}^l$ and a* low task *otherwise. The high tasks are further distinguished: if $\gamma, H \vdash s : \mathsf{Cmd}^L$ then it is a* high-low task *and otherwise it is a* high-high task*.*

A high task can only make high steps, but a low task can make both high and low steps. A high-high task can contain only high cycles (cycles with a high guard) because low cycles are not allowed in high context. A high-low task cannot contain any cycles (because at most low guards are allowed but high context requires at least high guards). We have the following lemma.

Lemma 1. *A low task cannot contain high* while *cycles. A high-low task cannot contain any* while *cycles.*

The restriction on the use of high while cycles is modeled after the restriction in [22]. Thus no low steps can follow a high while cycle in the same task. This restriction is checked in the rules (Seq_1) and (Seq_2). Because a low task must eventually release both locks, which is a low step, a low task cannot contain high while cycles at all. We extend the same restriction to await cycles. Thus a low task cannot await after a task that is allowed to make high cycles.

In our language, the scheduler of a cog cannot switch to a different task before the current task releases the high lock (or both locks). This can be done explicitly using suspend, but it is also done implicitly after each iteration of a while or await cycle. In contrast, in [22], by default the scheduler can switch tasks at any time, this can be disallowed by wrapping a sequence of commands in a protect construct. The protect construct is not allowed to contain cycles. This restriction corresponds to our implicit suspend after each iteration of a cycle.

Because our language allows more than one cog, there can be several low tasks running in parallel (at most one in each cog). This can create a situation where a low task n_1 in one cog (b_1) is in high context and awaits for a high task n_2 in another cog (b_2) but the high lock of cog b_2 is held by a low task n_3 in cog b_2. Thus the task n_1 cannot make the next low step before the task n_2 terminates, which cannot happen before the task n_3 releases the high lock but n_3 may make some low steps before it releases the lock. Thus it may depend on the high variables in n_1 whether low steps must be made in n_3 before the next low step in n_1 or not. Thus the low steps in n_3 are essentially in high context. To prevent this indirect information flow, we allow the task n_2 to overtake the high lock from n_3 in this situation. This means that n_3 is not required to make low steps before n_1 does, no matter what the values of high variables in n_1 are. This is handled by the reduction rule (await_3).

3.2 Non-interference

We first define the low-equivalence relation in Fig. 5. Here we assume (w.l.o.g.) that all variables in the program have globally unique names. Thus we can use a single type context γ instead of separate type contexts for each task.

$$\frac{\gamma, l \vdash s : \mathsf{Cmd}^H}{s \sim_\gamma s} \qquad \frac{\gamma, H \vdash s : \mathsf{Cmd}^H \qquad \gamma, H \vdash s' : \mathsf{Cmd}^H}{s \sim_\gamma s'}$$

$$\frac{\gamma, l \vdash s : \mathsf{Cmd}^H(T)}{s \sim_\gamma s} \qquad \frac{\gamma, H \vdash s : \mathsf{Cmd}^H(T) \qquad \gamma, H \vdash s' : \mathsf{Cmd}^H(T)}{s \sim_\gamma s'}$$

$$\frac{\gamma, H \vdash s_1 : \mathsf{Cmd}^H \qquad s_2 \sim_\gamma s_2'}{s_1; s_2 \sim_\gamma s_2'} \qquad \frac{\gamma, H \vdash s_1 : \mathsf{Cmd}^H \qquad s_2 \sim_\gamma s_2'}{s_2 \sim_\gamma s_1; s_2'} \qquad \frac{s_2 \sim_\gamma s_2'}{s_1; s_2 \sim_\gamma s_1; s_2'}$$

$$\sigma \sim_\gamma \sigma' \equiv dom(\sigma) = dom(\sigma') \wedge \forall v \in dom(\sigma).\, level(\gamma(v)) = L \Rightarrow \sigma(v) = \sigma'(v)$$

$$b[n_1, n_2] \sim_\gamma b[n_1, n_2']$$

$$\frac{\sigma \sim_\gamma \sigma'}{o[b, C, \sigma] \sim_\gamma o[b, C, \sigma']} \qquad \frac{\sigma \sim_\gamma \sigma' \qquad s \sim_\gamma s'}{n\,\langle b, o, \sigma, s\rangle \sim_\gamma n\,\langle b, o, \sigma', s'\rangle} \qquad \frac{P_1 \sim_\gamma P_1' \qquad P_2 \sim_\gamma P_2'}{P_1 \parallel P_2 \sim_\gamma P_1' \parallel P_2'}$$

$$\frac{\gamma, H \vdash s : \mathsf{Cmd}^{l_1}(T_2) \qquad P \sim_\gamma P'}{n\,\langle b, o, \sigma, s\rangle \parallel P \sim_\gamma P'} \qquad \frac{\gamma, H \vdash s : \mathsf{Cmd}^{l_1}(T_2) \qquad P \sim_\gamma P'}{P \sim_\gamma n\,\langle b, o, \sigma, s\rangle \parallel P'}$$

Fig. 5. The low-equivalence relation $\sim_\gamma$

From the definition of $\sim_\gamma$ we see that any typable command is equivalent to itself. Two commands are also equivalent if they both only have high side-effects. Commands with only high side-effects are also equivalent to skip-s.

Two local states are equivalent if the values of variables with low types are equal. Two objects are equivalent if the values of fields with low types are equal. The notion of equivalence is then extended to program configurations. We can now define the notion of non-interference we are considering. It is typical for the non-deterministic treatment of information flows, dating back to [24].

Definition 3 (Non-interference). *A program* $\overline{Cl}\{\overline{T\ x}\ s; x_0\}$ *is* non-interferent *if for any three states* σ_0, $\sigma_0^\bullet$ *and* σ_1 *satisfying* $\sigma_0 \sim_{\overline{x:T}} \sigma_1$,

$$b_0[n_0, n_0] \parallel n_0 \langle b_0, \mathsf{null}, \sigma_0, s; \mathsf{release}_L; x_0\rangle \overset{*}{\rightsquigarrow} n_0 \langle b_0, \mathsf{null}, \sigma_0^\bullet, x_0\rangle \parallel \ldots$$

implies that there exists a state $\sigma_1^\bullet$ *with* $\sigma_1^\bullet(x_0) = \sigma_0^\bullet(x_0)$ *and*

$$b_0[n_0, n_0] \parallel n_0 \langle b_0, \mathsf{null}, \sigma_1, s; \mathsf{release}_L; x_0\rangle \overset{*}{\rightsquigarrow} n_0 \langle b_0, \mathsf{null}, \sigma_1^\bullet, x_0\rangle \parallel \ldots \ .$$

Now we can prove the lemmas and the theorem for non-interference, stating that well-typed programs are non-interferent. Due to space constraints, we will just state the theorems here, and refer to [15] for the proofs and the necessary lemmas.

Theorem 1 (Subject reduction). *If* P_1 *and* P_2 *are well typed under* γ *and* $P_1 \sim_\gamma P_2$ *then if* $P_1 \rightsquigarrow P_1'$ *then there exists* P_2' *such that* $P_2 \rightsquigarrow^* P_2'$ *and* $P_1' \sim_\gamma P_2'$.

Theorem 2 (Non-interference). *If* $\vdash Pr : ok$, *where* $Pr = \overline{Cl}\ \{\overline{T\ x}\ s; x_0\}$, *then* Pr *is non-interferent.*

4 Related Work

The treatment of secure information flow in the language- and lattice-based setting is considered to have been pioneered by Denning and Denning. The first well-known type-based analysis for secure information flow in a simple imperative language was proposed by Volpano et al. [25]. Later, their analysis has been extended in many different directions, including treated language constructs, and the versatility of the tools for defining information flow properties. Our analysis, applied to a complex language, draws ideas from the developments in many of those directions. Let us give an overview of those.

While at first, the definitions of secure information flow were given in terms of distinguishable memories, bisimulation relations over program configurations [13,18] soon emerged as a convenient and composable way of defining information flow properties. The use of weak bisimulations, allowing stuttering, appeared in [23].

Object-oriented features, including fields and methods, were first treated in the JFlow (Jif) compiler [14]. However, they did not provide formal non-interference results. Such results for an OO-language were provided in [3]. In that area, a lot of attention has also been devoted to the analysis of low-level OO-languages, e.g. Java bytecode [4,5].

Concurrent languages, with possible race conditions and synchronization primitives, bring their own challenges. Secure information flow in a language with the possibility to spawn new threads was first considered in [24]. Synchronization primitives were considered in [19]. A bisimulation-based definition of secure information flow was provided in [7]. In this area, most of the research

seems to have concentrated on languages with parallel threads operating on a shared state. For the treatment of processes with private states, one may have to refer to the work based on process calculi [2,11,6]. Another interesting area is the building of distributed systems [26] satisfying certain information-flow properties.

In the analysis of thread pools with shared state, the properties of schedulers play a major role in the analysis of information flow properties. Their effect was first considered in [20]. More recently, scheduler strategies for providing the security of information flow have been considered [17,16].

5 Conclusions

We have demonstrated a type-based information flow analysis for a rich modeling language that has been designed to be applicable in designing large systems. As such, the type-based technique is a suitable choice because of its efficiency in checking large artefacts.

Our work demonstrates that the notion of futures, heavily employed by the language, may cause some interesting information flows in the system. These information flows are particularly apparent if the futures are considered as first-class values. In particular, the synchronization points they create can interfere with the scheduling decisions. Our work shows that the details of scheduling in ABS may need some further design efforts.

Our analysis has been applied to a language employing many different features. Our work has been valuable in pointing out how these features interact with each other in terms of possible information flows. We believe that our work will be helpful in making information flow type systems more widely used in the design and programming phases of the software development process.

References

1. 14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), Cape Breton, Nova Scotia, Canada. IEEE Computer Society (2001)
2. Abadi, M.: Secrecy by Typing in Security Protocols. In: Ito, T., Abadi, M. (eds.) TACS 1997. LNCS, vol. 1281, pp. 611–638. Springer, Heidelberg (1997)
3. Banerjee, A., Naumann, D.A.: Secure Information Flow and Pointer Confinement in a Java-like Language. In: CSFW, p. 253. IEEE Computer Society (2002)
4. Barthe, G., Rezk, T.: Non-interference for a JVM-like language. In: Morrisett, J.G., Fähndrich, M. (eds.) TLDI, pp. 103–112. ACM (2005)
5. Barthe, G., Rezk, T., Naumann, D.A.: Deriving an Information Flow Checker and Certifying Compiler for Java. In: IEEE Symposium on Security and Privacy, pp. 230–242. IEEE Computer Society (2006)
6. Bernardeschi, C., De Francesco, N., Lettieri, G.: Concrete and Abstract Semantics to Check Secure Information Flow in Concurrent Programs. Fundamenta Informaticae 60(1-4), 81–98 (2004)
7. Boudol, G., Castellani, I.: Noninterference for concurrent programs and thread systems. Theor. Comput. Sci. 281(1-2), 109–130 (2002)

8. de Boer, F.S., Clarke, D., Johnsen, E.B.: A Complete Guide to the Future. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 316–330. Springer, Heidelberg (2007)
9. Goguen, J.A., Meseguer, J.: Security Policies and Security Models. In: IEEE Symposium on Security and Privacy, pp. 11–20 (1982)
10. Hähnle, R., Johnsen, E.B., Østvold, B.M., Schäfer, J., Steffen, M., Torjusen, A.B.: Report on the Core ABS Language and Methodology: Part A. Highly Adaptable and Trustworthy Software using Formal Models (HATS), Deliverable D1.1A (2010)
11. Honda, K., Vasconcelos, V.T., Yoshida, N.: Secure Information Flow as Typed Process Behaviour. In: Smolka, G. (ed.) ESOP 2000. LNCS, vol. 1782, pp. 180–199. Springer, Heidelberg (2000)
12. Johnsen, E.B., Blanchette, J.C., Kyas, M., Owe, O.: Intra-Object versus Inter-Object: Concurrency and Reasoning in Creol. Electr. Notes Theor. Comput. Sci. 243, 89–103 (2009)
13. Mantel, H., Sabelfeld, A.: A Generic Approach to the Security of Multi-Threaded Programs. In: CSFW [1], p. 126
14. Myers, A.C.: JFlow: Practical Mostly-Static Information Flow Control. In: POPL, pp. 228–241 (1999)
15. Pettai, M., Laud, P.: Securing the Future — an Information Flow Analysis of a Distributed OO Language. Technical Report T-4-14, Cybernetica AS (2011)
16. Russo, A., Hughes, J., Naumann, J.D.A., Sabelfeld, A.: Closing Internal Timing Channels by Transformation. In: Okada, M., Satoh, I. (eds.) ASIAN 2006. LNCS, vol. 4435, pp. 120–135. Springer, Heidelberg (2008)
17. Russo, A., Sabelfeld, A.: Security for Multithreaded Programs Under Cooperative Scheduling. In: Virbitskaite, I., Voronkov, A. (eds.) PSI 2006. LNCS, vol. 4378, pp. 474–480. Springer, Heidelberg (2007)
18. Sabelfeld, A.: Confidentiality for Multithreaded Programs via Bisimulation. In: Broy, M., Zamulin, A.V. (eds.) PSI 2003. LNCS, vol. 2890, pp. 260–274. Springer, Heidelberg (2004)
19. Sabelfeld, A., Mantel, H.: Securing Communication in a Concurrent Language. In: Hermenegildo, M.V., Puebla, G. (eds.) SAS 2002. LNCS, vol. 2477, pp. 376–394. Springer, Heidelberg (2002)
20. Sabelfeld, A., Sands, D.: Probabilistic Noninterference for Multi-Threaded Programs. In: CSFW, pp. 200–214 (2000)
21. Schäfer, J., Poetzsch-Heffter, A.: JCoBox: Generalizing Active Objects to Concurrent Components. In: D'Hondt, T. (ed.) ECOOP 2010. LNCS, vol. 6183, pp. 275–299. Springer, Heidelberg (2010)
22. Smith, G.: A New Type System for Secure Information Flow. In: CSFW [1], pp. 115–125
23. Smith, G.: Probabilistic Noninterference through Weak Probabilistic Bisimulation. In: CSFW, pp. 3–13. IEEE Computer Society (2003)
24. Smith, G., Volpano, D.M.: Secure Information Flow in a Multi-Threaded Imperative Language. In: POPL, pp. 355–364 (1998)
25. Volpano, D.M., Irvine, C.E., Smith, G.: A Sound Type System for Secure Flow Analysis. Journal of Computer Security 4(2/3), 167–188 (1996)
26. Zheng, L., Chong, S., Myers, A.C., Zdancewic, S.: Using Replication and Partitioning to Build Secure Distributed Systems. In: IEEE Symposium on Security and Privacy, pp. 236–250. IEEE Computer Society (2003)

Improving Watermark Resistance against Removal Attacks Using Orthogonal Wavelet Adaptation

Jan Stolarek and Piotr Lipiński

Institute of Information Technology
Technical University of Lodz, Poland
{jan.stolarek,piotr.lipinski}@p.lodz.pl

Abstract. This paper proposes a new approach for enhancing the robustness of wavelet-based image watermarking algorithms. The method adjusts wavelet used in the process of image watermarking in order to maximize resistance against image processing operations. Effectiveness of the approach is demonstrated using blind multiplicative watermarking algorithm, but it can easily be generalized to all wavelet-based watermarking algorithms. Presented results demonstrate that wavelets generated using proposed approach outperform other wavelet bases commonly used in image watermarking in terms of robustness to removal attacks.

1 Introduction

Discrete Wavelet Transform (DWT) is widely applied in the field of digital image watermarking. Despite the popularity and proliferation of DWT-based watermarking algorithms, the following aspects attract surprisingly little attention: the influence of DWT filter coefficients on watermarked image fidelity and the influence of DWT filter coefficients on watermark robustness against attacks. Although the problem of choosing the optimal wavelet has been noticed by some authors [7,12], so far there have been no attempts to optimize the wavelet in order to increase either the watermarked image fidelity or attack resistance of the watermark. Huang and Jiang in [8] notice the influence of wavelet filter coefficients on watermarking fidelity, but they do not optimize it. Instead, they use wavelet filter parameters as a private key to increase security of the watermark. A similar approach is presented in [2] for two-dimensional case.

The algorithm for improving the watermarked image fidelity and watermark separability by wavelet adaptation has already been presented in [11]. In this paper we modify that algorithm in order to increase the watermarking robustness against removal attacks [16] while maintaining constant image fidelity. It will be demonstrated that the presented algorithm adapts the wavelet to a cover image and a watermark signal. Image fidelity assessment using the MSE measure was replaced by the Wang-Bovik Index [17].

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 588–599, 2012.

2 Orthogonal Wavelet Filter Parametrization

Orthogonal wavelet transform is implemented by an orthonormal filter bank, each of the filters having impulse response of even length L. In the case of wavelets, $L/2+1$ degrees of freedom are bound by the theoretical conditions imposed on the filters. The remaining $L/2-1$ degrees of freedom can be manipulated to adapt the properties of the filter. In order to effectively synthesize wavelets a filter parametrization must be defined. In this paper, parametrization based on the lattice filter [14] is used, following the description given in [18]. Basic operations of the lattice filter are orthogonal 2×2 rotations, which ensure both orthogonality of the structure and perfect reconstruction of the signal. For a filter of length L, $L/2$ orthogonal rotations have to be used. Fig.1 shows an example of a filter implementing 4-tap transform ($H(z)$ and $G(z)$ are the low-pass and high-pass filter outputs respectively). To ensure that wavelet implemented by the lattice filter has zero mean, the sum of all rotation angles in the structure must equal $-45°$ [13]:

$$\sum_{k=0}^{L/2} \alpha_k = -45^\circ \ , \tag{1}$$

where α_k are the angles of the orthogonal rotations. In order to ensure that this condition is always fulfilled, the following substitution can be used [13]:

$$\begin{aligned} &\alpha_1 = \vartheta - \varphi_1 \ , \\ &\alpha_i = (-1)^i(\varphi_{i-1} + \varphi_i) \ , \text{ for } i = 2, \ldots, \frac{L}{2} - 1 \ , \\ &\alpha_{\frac{L}{2}} = \varphi_{\frac{L}{2}-1} \ . \end{aligned} \tag{2}$$

The above shows that wavelets parametrized using the lattice filter are defined by a set of $L/2-1$ rotation angles φ_k.

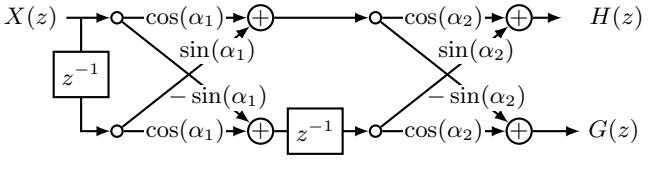


Fig. 1. Lattice filter implementing a 4-tap wavelet transform

3 Image Watermarking in the Wavelet Transform Domain

Figure 2a shows the generic scheme of digital image watermark embedding in the wavelet transform domain. The image to be watermarked is decomposed

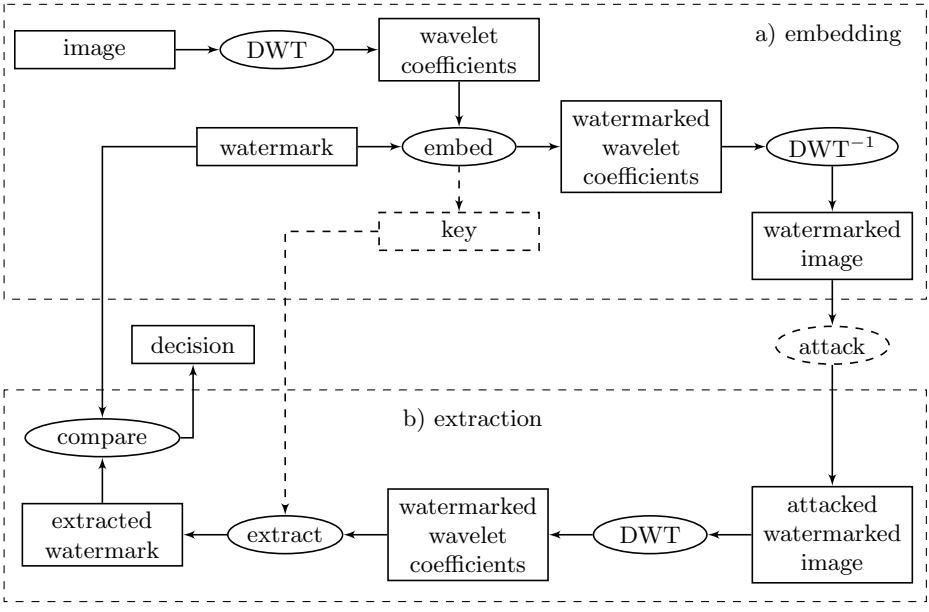


Fig. 2. Generic scheme of watermark embedding and blind extraction in the wavelet transform domain. Rectangles represent data, elipses represent algorithms.

using from 3 to 5 levels of DWT. Watermark sequence is then embedded in the wavelet coefficients. Some embedding algorithms return supplementary data – a *key* – that is essential to successfully extract the watermark. The image is reconstructed using the inverse wavelet transform (denoted as DWT^{-1} in Fig.2a) and it is released to the public where it may be subjected to attacks, i.e. operations – deliberate or not – that make the watermark extraction and identification more difficult. To check for watermark presence the presumably watermarked and possibly attacked image has to be decomposed using the same number of DWT levels as was used for watermark embedding. Watermark is then extracted from the wavelet coefficients (using the optional key, if necessary) and compared with the embedded one. Schemes that do not require knowledge of the original image are called *blind*. It must be noted that this generic scheme may differ in details for particular embedding algorithms.

The improvement method presented in this paper is a general one and can be used to enhance any wavelet-based watermarking algorithm. For demonstration purposes the simplest blind multiplicative watermarking method was used, in which the watermark $\mathbf{w}$ is a sequence of N random numbers from set $\{-1, 1\}$ and the embedding formula is

$$\mathbf{c}^{(w)} = \mathbf{c} + \kappa \cdot |\mathbf{c}| \cdot \mathbf{w} \ , \tag{3}$$

where $\mathbf{c}$ are the N largest wavelet coefficients selected from the highest level detail subbands (see Fig.3), κ is the embedding strength, $|\cdot|$ denotes the absolute

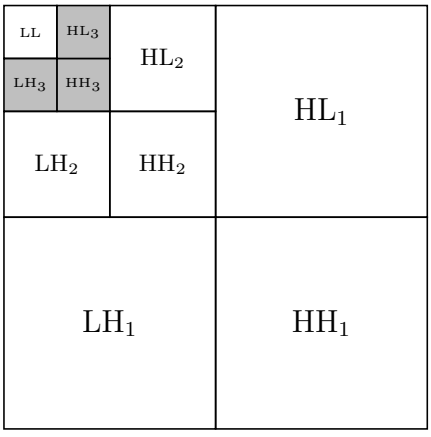


Fig. 3. Wavelet decomposition of an image. Watermark is embedded in the third level detail subbands, corresponding to middle frequencies (in gray).

value, $\mathbf{w}$ is the watermark sequence and $\mathbf{c}^{(w)}$ are the watermarked wavelet coefficients. Operations on the vectors in (3) are performed element-wise. Locations of the watermarked coefficients $\mathbf{c}^{(w)}$ have to be stored, since they are required in the process of watermark extraction – these locations play the role of the key (see Fig.2). To detect the presence of the watermark, normalized correlation between the embedded watermark and the presumably watermarked coefficients is calculated using the formula

$$C = \frac{1}{N-1} \sum_{i=1}^{N} \frac{(c_i^{(w)} - \overline{c^{(w)}})(w_i - \overline{\mathbf{w}})}{\sigma_c \sigma_w} \quad , \tag{4}$$

where $c_i^{(w)}$ are the presumed watermarked coefficients, $\overline{c^{(w)}}$ is the mean value of the watermarked coefficients, w_i are the embedded watermark values, $\overline{\mathbf{w}}$ is the mean value of the embedded watermark, σ_c and σ_w are standard deviations of watermarked coefficients and the watermark respectively.

In the digital image watermarking there are three main requirements, that determine whether the watermarking process is effective:

- Fidelity: watermarking process must not degrade the quality of the watermarked image.
- Separability: the extracted watermark must have significantly greater normalized correlation with the embedded watermark than with random watermarks. This ensures faultless watermark detection and distinction.
- Attack resistance: watermark separability must be maintained despite image manipulations (unless the image is damaged beyond usability).

It has already been demonstrated [11] that the mother wavelet used for image decomposition can be adjusted to the image, the watermark and the watermarking

algorithm in order to improve separability and fidelity. In this paper we modified that system to maximize watermark resistance against image processing operations and maintain constant fidelity of the watermarked image.

4 Watermarked Image Fidelity

We assess the fidelity of the watermarked image using the Wang-Bovik Index (WBI) [17], which takes into account the loss of correlation, luminance distortion and contrast distortion, thus offering a better image quality estimation than the widely used MSE. The original WBI is calculated by using the moving window approach. For each window the index between original image X and watermarked image Y is calculated using the following formula:

$$Q_m = \frac{4\sigma_{xy}\overline{x}\overline{y}}{(\sigma_x^2 + \sigma_y^2)[(\overline{x})^2 + (\overline{y})^2]} \quad , \tag{5}$$

where $\overline{x}$ and $\overline{y}$ are mean values of pixels in a window, σ_x^2 and σ_y^2 are the variances, σ_{xy} is given as

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})(y_i - \overline{y}) \quad , \tag{6}$$

where x_i and y_i denote pixels of non-watermarked and watermarked image window respectively and N denotes the number of pixels in a single window. The moving window approach creates a map of distortions, where each $Q_m \in [-1, 1]$. We found that converting this map to a single value describing quality of an image – which is required if the constant distortion rate is to be maintained – is problematic. We achieved the best result by dropping the moving window approach and calculating WBI once for the whole image instead. To ensure that the distortion of the watermarked image was constant for every image, watermark and wavelet, the embedding strength κ in (3) was adjusted using Matlab's `fmincon` function.

5 Improving Watermark Attack Resistance

In commonly used DWT-based watermarking algorithms wavelet basis used for image decomposition and reconstruction is chosen arbitrarily. As a result the wavelets used in watermark embedding are suboptimal for a given cover image, watermark, embedding algorithm and attacks. In this paper we used evolutionary approach to synthesize wavelet basis that adapts to the cover image, watermark and embedding algorithm and also provides better robustness against attacks than already existing wavelets. Simple Genetic Algorithm (SGA) combined with Evolution Strategies was applied. SGA maintains a population of possible solutions called *individuals*. Each individual represents a wavelet filter parametrized using the lattice structure. Therefore, the filter of length L was represented as

a set of $\frac{L}{2} - 1$ binary coded φ_i angles. Watermark resistance against selected attacks (e.g. JPEG compression, median filtering, noise contamination etc.) was increased by carrying out the attacks independently on the watermarked image. For each attack the watermark extraction was performed and partial fitnesses based on separability of the watermark were assigned to an individual:

$$F_j(k) = \min_{i \in \{1,\ldots,M\}} (C_e^{(k)} - C_r^{(i,k)}) \ , \tag{7}$$

where $k \in \{1, \ldots, K\}$ is index of the attack and $k = 0$ denotes no attack, j is the index of an individual, $C_e^{(k)}$ is the normalized correlation between the extracted watermark and the embedded watermark for k-th attack and $C_r^{(i,k)}$ is the normalized correlation between the extracted watermark and the i-th random watermark and k-th attack. M denotes the number of random watermarks. The smallest difference between the correlations was selected as individual's fitness. Since $C_e^{(k)}, C_r^{(i,k)} \in [-1, 1]$, then $F_j(k) \in [-2, 2]$. Tournament selection was used and therefore normalization of $F_j(k)$ to ensure that it was greater than zero was avoided. Introducing (7) created selection pressure that promoted individuals maximizing the separability despite the image distortion introduced by the attack.

The lowest partial fitness was selected as the total fitness of an individual:

$$F_j = \min_{k \in \{0,\ldots,K\}} \{F_j(k)\} \ . \tag{8}$$

This approach ensured that synthesized wavelets offered high resistance to all of the performed attacks. Wavelets that failed to meet the high robustness requirement against at least one of the attacks were assigned low fitness value, which eventually led to their elimination in the genetic algorithm. The scheme of the fitness evaluation algorithm is given in Fig.4.

6 Experimental Results

For our experiments, we selected 20 images from the USC-SIPI Image Database (including textures and well-known images like Lena, Barbara, Baboon, Airplane, Boat, Lake etc.). A watermark $\mathbf{w}$ consisting of 512 random values in $\{-1, 1\}$ was embedded in the third level detail subbands and the acceptable perceptual image distortion rate was selected to WBI = 0.996. Separability given by (7) was used to characterize the robustness of the watermarking algorithm. 1000 randomly generated watermarks were used for calculating the separability. Removal attacks contained JPEG compression, median filtering, low-pass filtering and scaling (the image was scaled down to 40% of original size and then rescaled to original size).

The proposed algorithm was used to synthesize the optimal wavelet filter for every selected image. Orthogonal wavelets of length 4, 6, 8, 12 and 20 taps were synthesized. Due to lack of space, detailed results are presented only for two example images: one photo (Boat) and one texture (Fig.5b). The robustness of adaptive wavelets (denoted as $\mathbf{A}$) was compared with other wavelets known

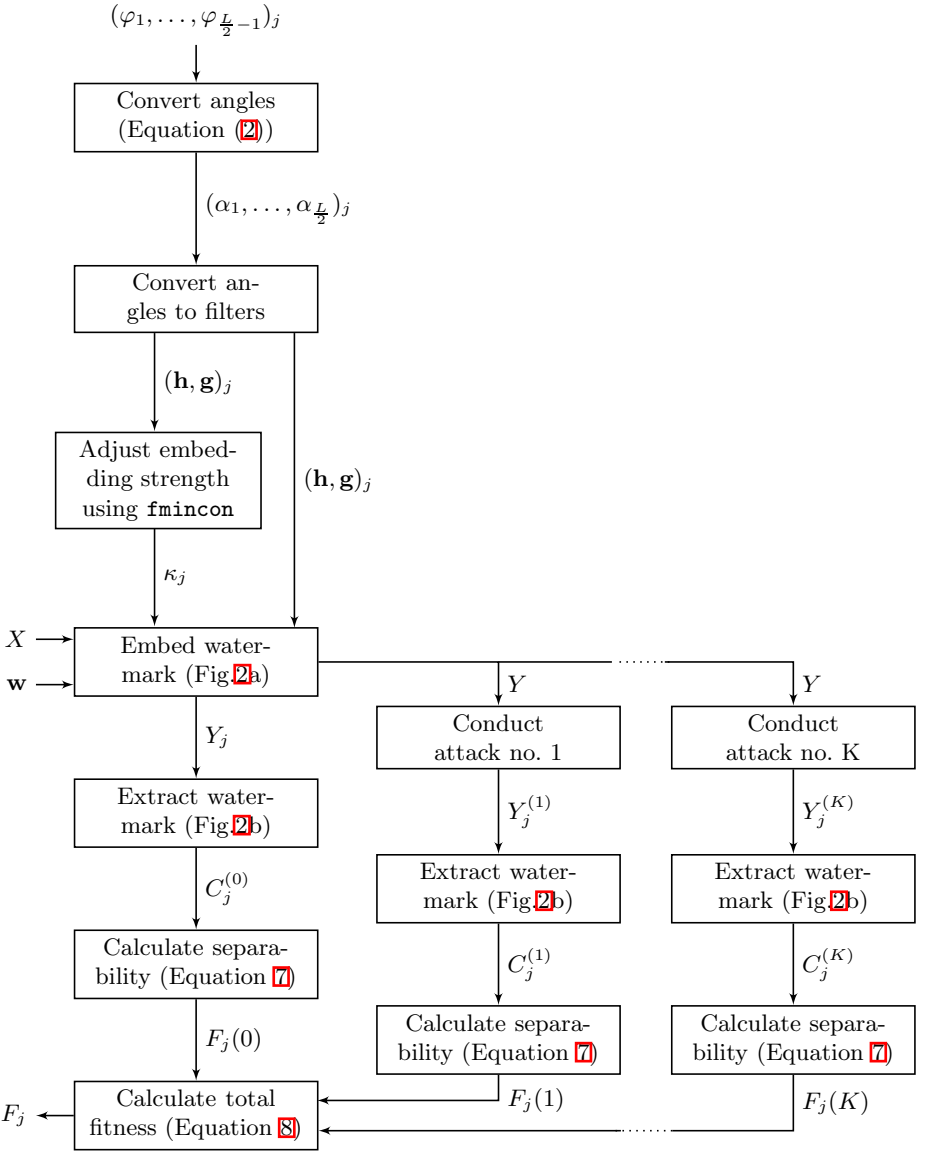


Fig. 4. Fitness evaluation scheme. X represents the original image, Y represents the image watermarked with $\mathbf{w}$, $(\mathbf{h},\mathbf{g})$ represents the low-pass and high-pass filters respectively, j is an index of an indvidual in a population.

Table 1. Comparison of separability values for adaptive wavelets and commonly used wavelets for Boat test image. Adaptive wavelet denoted with bold font, biorthogonal wavelets denoted with oblique font.

no attack		JPEG compression		median filter		low-pass filter		scaling	
A6	0.327	**A6**	0.302	**A6**	0.311	**A6**	0.315	**A6**	0.311
A12	0.311	**A8**	0.279	**A8**	0.275	**A20**	0.272	**A8**	0.268
A8	0.306	**A4**	0.268	**A20**	0.270	**A8**	0.268	**A4**	0.263
A4	0.291	**A20**	0.265	**A4**	0.262	**A4**	0.258	**A20**	0.258
A20	0.281	**A12**	0.233	**A12**	0.251	**A12**	0.249	**A12**	0.246
C6	0.273	C12	0.230	C6	0.242	S12	0.242	C6	0.239
D4	0.265	*V3*	0.226	S12	0.241	C6	0.241	D4	0.226
S12	0.259	S12	0.224	C12	0.237	C12	0.238	C12	0.214
Ha	0.251	C6	0.224	*V2*	0.231	*V6*	0.235	*V2*	0.205
C12	0.250	*V5*	0.217	*V6*	0.226	*V2*	0.230	*V3*	0.204
V2	0.244	*V6*	0.217	D4	0.220	*V3*	0.218	D8	0.202
V6	0.242	Ha	0.210	*V3*	0.220	D4	0.217	*V6*	0.197
V5	0.233	*V2*	0.188	D8	0.214	D8	0.215	*V5*	0.197
V3	0.231	D8	0.180	*V5*	0.213	*CDF*	0.210	S8	0.193
D8	0.231	*CDF*	0.176	*CDF*	0.210	*V5*	0.201	S12	0.190
CDF	0.214	C18	0.175	Ha	0.198	*An*	0.185	C18	0.179
An	0.209	D4	0.172	*An*	0.189	C18	0.184	*Od*	0.178
LG	0.203	S10	0.165	S8	0.187	S8	0.184	*An*	0.172
S8	0.201	S8	0.164	C18	0.185	Ha	0.178	*LG*	0.158
C18	0.199	*V4*	0.155	*Od*	0.174	*Od*	0.175	*CDF*	0.157
Od	0.189	*An*	0.153	*LG*	0.164	*LG*	0.172	Ha	0.153
S10	0.183	*LG*	0.150	S10	0.163	S10	0.168	D6	0.150
V4	0.177	*Od*	0.146	*V4*	0.155	*V4*	0.160	S10	0.141
D6	0.173	D6	0.138	D6	0.148	D6	0.144	*V4*	0.119

from the literature: orthogonal Haar (Ha), Daubechies (D4, D6, D8), Symlet (S4, S6, S8) and Coiflet (C6, C12, C18) wavelets [5] and the biorthogonal CDF 9/7 (*CDF*) [3], LeGall 5/3 (*LG*) [9], Antonini 9/7 (*An*) [1], Odegard (*Od*) [6], Villasenor 13/11 (*V2*), 6/10 (*V3*), 5/3 (*V4*), 2/6 (*V5*), 9/3 (*V6*) [15] wavelets. Each of these wavelets was used to embed the watermark in an image and the separability – with and without the attacks – was calculated. Tables 1 and 2 present separability values for all of the above mentioned wavelets. The first column contains separability for watermarked, unattacked image; the second column: separability for JPEG attack; the third column: separability for median attack; the fourth column: separability for low-pass filter; the fifth column: separability for scaling attack.

The results show that in each case the adaptive wavelets outperform wavelets proposed in the literature. Furthermore, no non-adaptive wavelet could be considered the best, as they perform differently depending on the conducted attack even though the same image and watermark are used. This confirms the observations made by Dietze and Jassim [7] that no single wavelet can be regarded as

Table 2. Comparison of separability values for adaptive wavelets and commonly used wavelets for texture test image. Adaptive wavelet denoted with bold font, biorthogonal wavelets denoted with oblique font.

no attack		JPEG compression		median filter		low-pass filter		scaling	
A4	0.717	**A8**	0.623	**A4**	0.679	**A4**	0.653	**A6**	0.517
A20	0.699	**A4**	0.575	**A6**	0.642	**A12**	0.619	**A12**	0.492
A8	0.685	**A6**	0.568	**A20**	0.635	**A6**	0.614	**A4**	0.492
A6	0.663	**A12**	0.547	**A8**	0.634	**A20**	0.607	**A8**	0.490
A12	0.649	**A20**	0.546	**A12**	0.620	**A8**	0.605	**A20**	0.439
C6	0.603	C6	0.539	C6	0.577	C6	0.552	C6	0.418
V4	0.532	*V4*	0.408	*V4*	0.508	*V4*	0.510	*V4*	0.362
LG	0.516	*An*	0.405	*LG*	0.493	*LG*	0.496	*CDF*	0.359
An	0.490	*LG*	0.398	*An*	0.457	*An*	0.470	*LG*	0.346
CDF	0.413	C12	0.327	*CDF*	0.395	*CDF*	0.404	*Od*	0.345
C12	0.384	*CDF*	0.317	C12	0.358	C12	0.363	*An*	0.344
Od	0.341	*Od*	0.287	*Od*	0.323	*Od*	0.340	C12	0.321
V6	0.328	*V6*	0.239	*V6*	0.306	*V6*	0.324	*V6*	0.263
D6	0.267	D6	0.226	D6	0.241	D6	0.252	D6	0.235
V2	0.248	C18	0.213	*V2*	0.231	*V2*	0.245	*V2*	0.228
C18	0.239	*V2*	0.195	C18	0.219	C18	0.217	C18	0.199
D4	0.229	S8	0.147	D4	0.195	D4	0.192	D4	0.195
S8	0.173	D4	0.143	S8	0.153	S8	0.167	S8	0.157
S10	0.122	S10	0.100	S10	0.108	S10	0.110	S12	0.147
S12	0.122	S12	0.096	S12	0.103	S12	0.105	S10	0.103
Ha	0.093	Ha	0.075	Ha	0.069	D8	0.068	D8	0.081
D8	0.080	*V5*	0.073	*V5*	0.067	*V5*	0.065	Ha	0.053
V5	0.074	*V3*	0.055	D8	0.064	*V3*	0.056	*V3*	0.049
V3	0.056	D8	0.051	*V3*	0.045	Ha	0.050	*V5*	0.039

optimal in terms of robustness. The proposed algorithm overcomes this limitation by adapting to the image, the watermark and the watermarking algorithm. This can be demonstrated by embedding the watermark in the Boat image using adaptive wavelets synthesized for Lena image. The result of such an experiment is shown in Table 3. It can be clearly noticed that the performance of adaptive wavelets has degraded significantly. For each image the synthesized wavelet is different. Fig. 6a and Fig. 6b show example scaling functions synthesized for two of the test images. Function in Fig. 6b resembles Coiflet 6 wavelet and indeed the Coiflet 6 wavelet performs better than other non-adaptive wavelets for the example texture image. Tables 1 and 2 allow also to conclude that the algorithm works for different classes of images (natural and textures).

7 Scope of the Proposed Method

It is important to clearly define the scope of the proposed method. As the experimental results have shown, the introduced algorithm increases watermark

Table 3. Comparison of separability values for adaptive wavelets synthesized for Lena image and used to embed the watermark in Boat image. Adaptive wavelet denoted with bold font, biorthogonal wavelets denoted with oblique font.

no attack		JPEG compression		median filter		low-pass filter		scaling	
C6	0.273	C12	0.230	C6	0.242	S12	0.242	C6	0.238
D4	0.265	*V3*	0.226	S12	0.241	C6	0.241	S12	0.230
S12	0.259	S12	0.224	C12	0.237	C12	0.238	C12	0.230
Ha	0.251	C6	0.224	*V2*	0.231	*V6*	0.235	*V2*	0.221
C12	0.250	*V5*	0.217	*V6*	0.226	*V2*	0.230	D4	0.217
A4	0.245	*V6*	0.217	D4	0.220	*V3*	0.218	*V6*	0.217
V2	0.244	Ha	0.210	*V3*	0.220	D4	0.217	*V3*	0.216
V6	0.242	**A20**	0.194	**A20**	0.219	D8	0.215	D8	0.204
A20	0.242	*V2*	0.188	D8	0.214	**A20**	0.212	*V5*	0.201
V5	0.233	D8	0.180	*V5*	0.213	*CDF*	0.210	**A20**	0.197
V3	0.231	**A12**	0.178	*CDF*	0.210	*V5*	0.201	*CDF*	0.192
D8	0.231	*CDF*	0.176	**A4**	0.199	**A4**	0.198	**A12**	0.189
A8	0.214	C18	0.175	Ha	0.198	**A12**	0.190	C18	0.186
CDF	0.214	**A4**	0.175	**A12**	0.190	*An*	0.185	**A4**	0.182
An	0.209	D4	0.172	*An*	0.189	C18	0.184	S8	0.181
A12	0.208	S10	0.165	S8	0.187	S8	0.184	*An*	0.180
LG	0.203	S8	0.164	C18	0.185	**A8**	0.180	*Od*	0.173
S8	0.201	**A8**	0.158	**A8**	0.178	Ha	0.178	*LG*	0.171
C18	0.199	*V4*	0.155	*Od*	0.174	*Od*	0.175	Ha	0.166
Od	0.189	*An*	0.153	*LG*	0.164	*LG*	0.172	**A8**	0.165
A6	0.183	*LG*	0.150	S10	0.163	S10	0.168	S10	0.153
S10	0.183	*Od*	0.146	**A6**	0.155	*V4*	0.160	*V4*	0.149
V4	0.177	D6	0.138	*V4*	0.155	D6	0.144	D6	0.146
D6	0.173	**A6**	0.122	D6	0.148	**A6**	0.143	**A6**	0.133

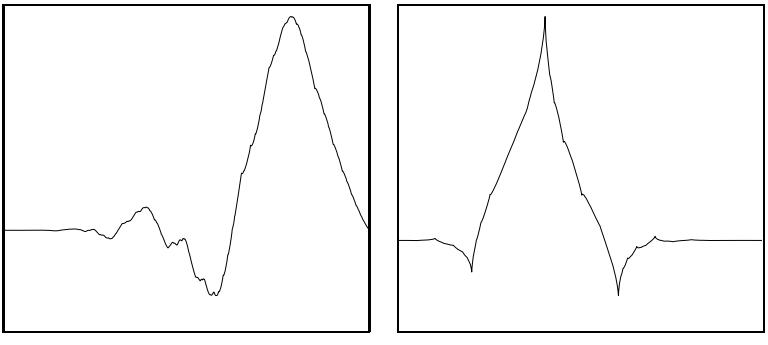
resistance to undeliberate[1] removal attacks[2], e.g. image compression or filtering. Nevertheless, wavelet bases adaptation is not a universal solution to all the security problems that arise in the field of image watermarking. The method will not increase robustness against desynchronization attacks, e.g. geometric attacks like cropping, scaling or rotation. The watermarking algorithm must ensure that the synchronization is regained after such an attack [10]. Please note that in our experiments, we carried out scaling attack by scaling the image down to 40% of its original size and then scaling it back to its original size, thus inverting the desynchronization. The proposed method does not also increase robustness against protocol, coping and sensitivity attacks and it does not increase security of invertible or quasi-invertible [4] watermarking algorithms.

[1] Not exploiting knowledge of the watermark embedding scheme, watermark sequence etc.

[2] According to classification by Voloshynovskiy et al. [16].

(a) Boat (b) texture

Fig. 5. Example test images



(a) Scaling function synthesized for Boat test image (b) Scaling function synthesized for texture test image

Fig. 6. Adaptive scaling functions

8 Conclusions

In this paper, the problem of improving digital watermarking effectiveness by wavelet synthesis is addressed. Genetic-based method of adapting the wavelets to the image, the watermark, the embedding algorithm and selected types of removal attacks is presented. The results of experiments that are demonstrated here, as well as the experiments which were carried out on the other test images, prove that wavelets generated using the proposed method outperform the commonly used wavelet bases in terms of robustness against attacks.

References

1. Antonini, M., Barlaud, M., Mathieu, P., Daubechies, I.: Image coding using wavelet transform. IEEE Transactions on Image Processing 1, 205–220 (1992)
2. Cheng, G., Yang, J.: A watermarking scheme based on two-dimensional wavelet filter parametrization. In: Fifth International Conference on Information Assurance and Security, IAS 2009, pp. 301–304 (2009)

3. Cohen, A., Daubechies, I., Feauveau, J.-C.: Biorthogonal bases of compactly supported wavelets. Communications on Pure and Applied Mathematics 45(5), 485–560 (1992)
4. Craver, S., Memon, N., Yeo, B.-L., Yeung, M.M.: Resolving rightful ownerships with invisible watermarking techniques: Limitations, attacks, and implications. IEEE Journal on Selected Areas in Communications 16(4), 573–586 (1998)
5. Daubechies, I.: Ten Lectures on Wavelets. SIAM (1992)
6. Davis, G.: Wavelet image compression construction kit (1997), http://www.geoffdavis.net/dartmouth/wavelet/wavelet.html
7. Dietze, M., Jassim, S.: Filters ranking for DWT-domain robust digital watermarking. EURASIP Journal on Applied Signal Processing 14, 2093–2101 (2004)
8. Huang, Z.Q., Jiang, Z.: Watermarking still images using parametrized wavelet systems. In: Image and Vision Computing. Institute of Information Sciences and Technology, Massey University (2003)
9. Le Gall, D., Tabatabai, A.: Sub-band coding of digital images using symmetric short kernel filters and arithmetic coding techniques. In: International Conference on Acoustics, Speech, and Signal Processing, ICASSP 1988 (1988)
10. Licks, V., Jordan, R.: Geometric attacks on image watermarking systems. IEEE Multimedia 12(3), 68–78 (2005)
11. Lipiński, P., Stolarek, J.: Digital watermarking enhancement using wavelet filter parametrization. In: Dobnikar, A., Lotrič, U., Šter, B. (eds.) Adaptive and Natural Computing Algorithms (10th ICANNGA, 2011), vol. 1, pp. 330–339 (2011)
12. Miyazaki, A.: A study on the best wavelet filter bank problem in the wavelet-based image watermarking. In: 18th European Conference on Circuit Theory and Design, ECCTD 2007, pp. 184–187 (2007)
13. Rieder, P., Götze, J., Nossek, J.S., Burrus, C.S.: Parameterization of orthogonal wavelet transforms and their implementation. IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 45(2), 217–226 (1998)
14. Vaidyanathan, P.P., Hoang, P.-Q.: Lattice structures for optimal design and robust implementation of two-channel perfect-reconstruction QMF banks. IEEE Transactions on Acoustics, Speech and Signal Processing 36(1), 81–94 (1988)
15. Villasenor, J.D., Belzer, B., Liao, J.: Wavelet filter evaluation for image compression. IEEE Transactions on Image Processing 4(8), 1053–1060 (1995)
16. Voloshynovskiy, S., Pereira, S., Pun, T., Eggers, J.J., Su, J.K.: Attacks on digital watermarks: Classification, estimation based attacks and benchmarks. IEEE Communications Magazine 39(8), 118–126 (2001)
17. Wang, Z., Bovik, A.C.: A universal image quality index. IEEE Signal Processing Letters 9(3), 81–84 (2002)
18. Yatsymirskyy, M.: Lattice structures for synthesis and implementation of wavelet transforms. Journal of Applied Computer Science 17(1), 133–141 (2009)

MAK€ – A System for Modelling, Optimising, and Analyzing Production in Small and Medium Enterprises

Roman Barták[1], Con Sheahan[2], and Ann Sheahan[3]

[1] Charles University in Prague, Faculty of Mathematics and Physics
Malostranské náměstí 2/25, 118 00 Praha 1, Czech Republic
bartak@ktiml.mff.cuni.cz
[2] University of Limerick, Limerick, Ireland
Con.Sheahan@ul.ie
[3] ManOPT Systems Ltd., National Technology Park, Limerick, Ireland
Ann.Sheahan@manopt.com

Abstract. The paper presents a performance prediction and optimisation tool MAK€ that allows users to model enterprises in a visually rich and intuitive way. The tool automatically generates a scheduling model describing all choices that users can do when optimising production. This model then goes to the Optimiser Module that generates schedules optimising on-time-in-full performance criterion while meeting the constraints of the firm and the customer demand. Finally, the Performance Manager Module shows the decision maker what is the best possible outcome for the firm given the inputs from the Enterprise Modeller. The Optimiser Module, which is the main topic of this paper, is implemented using constraint-based solving techniques with specific search heuristics for this type of problems. It demonstrates practical applicability of constraint-based scheduling – one of the killer application areas of constraint programming, a technology originated from AI research.

Keywords: production scheduling, problem modelling, optimisation.

1 Introduction

Though there exists a vast amount of research in the area of scheduling there is still a large gap between practical problems and research results especially in the area of production optimisation for small and medium enterprises. This gap is partly due to missing modelling and visualization tools that would allow easy transformation of real-life problems to optimisation models and the results back to customers [8] and partly due to large distance of academic algorithms from existing problems. The MAK€ tool developed by ManOPT Systems (now traded as Entellexi, www.entellexi.com) addresses the above gaps by providing a streamlined feature rich environment where the user can do all of the following in a simple, efficient and user-friendly way:

- specify how a particular product is manufactured (i.e. define a bill of material accompanied by a workflow describing manufacturing of a single product);

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 600–611, 2012.

- enter a work order from a customer (customers request certain quantities of products that the factory can manufacture, together with a desired deadline);
- generate a schedule for the order (the schedule describes which elementary operations are executed together with exact timing and used resources to efficiently fulfil the work orders);
- display the generated schedule in the form of a Gantt chart including production characteristics of the schedule such as resource utilization.

The MAK€ tool is a result of EU funded projects EMPOSME and ValuePOLE that gave a unique opportunity for co-operation between academia and industry as usually researchers and final customers are too far from each other to communicate directly the needs on one side and the possibilities on the other side. In the project we demonstrate the recent research advancements in the areas of formal problem modelling and solving, namely using Temporal Networks with Alternatives [5] to describe workflows, in real-life industrial setting. The goal was to make optimisation technology accessible to practitioners without any knowledge of optimisation techniques. The paper shows how AI-originated technology called Constraint Programming (CP) is used in the MAK€ scheduling engine also called optimiser. We will first specify the optimisation problem solved by the MAK€ tool. Then we will explain why CP was selected as the core optimisation technology and give some particular examples how CP techniques were used. Finally, we will present the added value of using this technology from the industrial perspective. In the paper we are giving a broader picture of the area to show that there is a long path from the customer to the optimiser where many important decisions must be done before the formal model is presented to the optimiser to generate a schedule.

2 The Problem

Frequently, when people are applying Constraint Programming techniques to real-life problems, they focus on solving a particular problem for which they formulate (manually) a constraint model and then fine tune the model to achieve acceptable performance for a given sort of data [13,16]. This is not the case of the MAK€ project which is intended to provide a tool for production optimisation in small and medium enterprises in general. It means that there is no single problem to be solved but rather a collection of problems with a common core – production of certain pieces of items that are manufactured, assembled, packed, and delivered to a customer. It means that the system collects data about particular enterprise in the form common for enterprise systems; it generates an optimisation model fully automatically; it solves the optimisation problem, it shows the result to the user, and finally it supports analysis of the result. Obviously, there are many optimisation problems appearing in enterprises starting with optimising layout of the factory, minimizing waste production and energy consumption, optimising resource utilization and others. In the MAK€ system we focus on optimising production schedules.

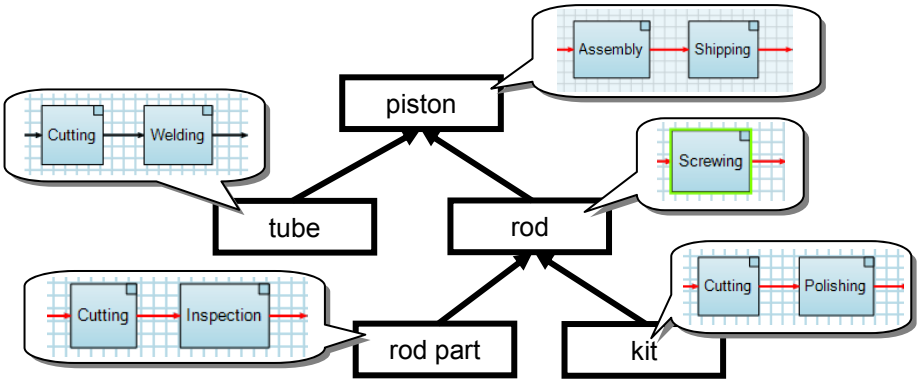


Fig. 1. Example of bill of materials accompanied by workflows for individual items

There exists a huge number of various scheduling problems with many ad-hoc algorithms to solve particular classes of problems [10]. In the MAK€ system, the scheduling problem is not described explicitly by the user, but it is rather derived automatically from data about enterprise provided by the user [8]. These data consist of description of bill of materials, workflows, resources, and custom orders. *Bill of materials* (BOM) can be seen as a hierarchical structure (tree) with the final product on top (in the root) and items from which this product is composed below. BOM determines how the final product is assembled from its component parts. For example to produce a piston we need a tube and a rod, where the rod consists of a rod part and some kit (Figure 1). BOM describes not only the structure of the product but also the quantities of required components (in our example, one unit of each component is required). BOM is accompanied by the description of manufacturing routes which describe *workflows* for each basic part. Both bill of materials and workflows define temporal constraints between individual manufacturing operations. In particular, the BOM structure specifies that processing of components must be finished before assembling them together. The workflow structure defines explicit temporal relations between the operations including details such as the minimal time distance between two operations. For each operation there are one or more *machines, tools, operators*, etc. which are required to carry it out. These resources have assigned availability calendars and performance characteristics that can be used to pre-compute durations of operations (together with information about quantity of products from BOM). Currently, we assume unary resources only so each resource can process at most one operation at any time. Bill of materials, workflows, and resources describe the enterprise, but we need custom orders to dictate what needs to be manufactured. Each *order* specifies the ordered item, its quantity, and the delivery date. The major objective that we focus on is *on-time-in-full* delivery. It means that the goal of optimiser is producing a schedule where the ordered products are ready to ship at requested dates. Notice that the input data are not in the format of a typical scheduling problem so when formalising the scheduling problem to be solved, we need to take in account all possible scenarios that can be obtained from input data.

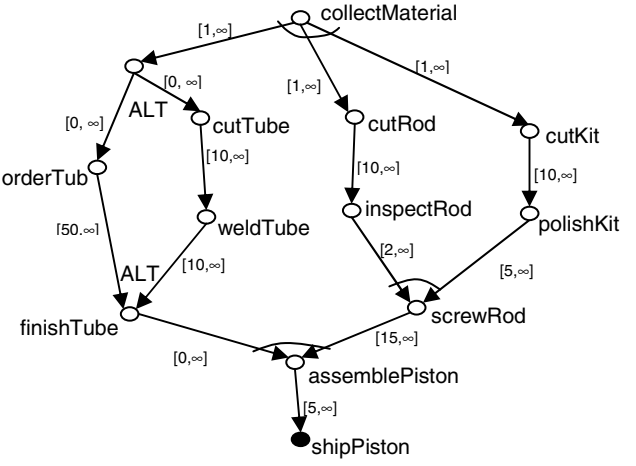


Fig. 2. A manufacturing process with alternatives (the alternative branching is marked by ALT, the other branchings are parallel; arcs are annotated by simple temporal constraints expressing minimal and maximal distance in time)

The closest formalism for the underlying scheduling problem is probably the Extended Resource Constrained Project Scheduling Problem [15]. We describe the scheduling problem as a set of non-interruptible operations, where each operation has a fixed duration and has assigned a set of unary resources to which the operation must be allocated if the operation is decided to be part of the schedule. The operations are connected in a temporal network with alternatives [5] that describes the workflows.

Temporal network with alternatives (TNA) is basically a directed acyclic graph where nodes correspond to operations and arcs are annotated by simple temporal constraints specifying the minimal and maximal time distance between the start times of operations (see Figure 2). The main difference from traditional temporal networks is specifying the *branching constraints*. If there are more arcs going from the node then either parallel or alternative branching is specified in the TNA. The type of branching describes the flow of the process. *Parallel branching* means that either all operations (the operation and all its direct successors) are present in the solution or no operation is present (see *collectMaterial* in Figure 2). This constraint is also assumed in the case of having exactly one successor. *Alternative branching* means that either the operation together with exactly one of its direct successors (predecessors) in the TNA is present in the solution or no operation is present (see *finishTube* and its predecessors in Figure 2). This is the way to describe splitting of the process into several alternatives. Note also, that the same mechanism can be used to model alternative resources (the process splits into alternative operations and each of the operations is allocated to one of the alternative resources). Similarly, we describe alternative or parallel joining of processes if there are more arcs going into a node. We may also assume *auxiliary operations* with zero duration and no resource requirements in the TNA to model auxiliary time points in TNA (see the top left node in Figure 2). Using alternatives in TNA is the main difference from traditional

scheduling problems as the scheduling task now includes selecting the operations [9] together with allocating them to time while respecting the temporal, branching, and resource constraints. To state it differently: the task is to select a subset of operations that satisfies the branching constraints (this was called a P/A graph assignment problem in [5]) and to decide the start time of each selected operation in such a way that the temporal constraints from TNA are satisfied and the selected operations allocated to the same resource do not overlap in time. This is a form of integrated planning (selection of operations) and scheduling (allocation of operations) problem.

Note that selecting a consistent subset of nodes from a general TNA where some nodes are preselected (a P/A graph assignment problem) is an NP-complete problem [5]. Fortunately, real-life workflows frequently have a specific structure [1], which we call a *nested TNA*, where the P/A graph assignment problem is tractable [6]. Nested structure means that the network is obtained by a decomposition process starting with single arc and decomposing any arc into a set of arcs and nodes called a nest as Figure 3 shows (the TNA in Figure 2 is also nested). The bad news is that adding simple temporal constraints [14] to nested TNA makes the problem NP-complete again [7]. What makes the problem hard is using the "deadline" constraints, in terms of TNA it means using maximal distance between operations. In practice, users usually specify only the minimal distance constraints though there exist industries, such as metallurgy, where maximal distance between operations is important.

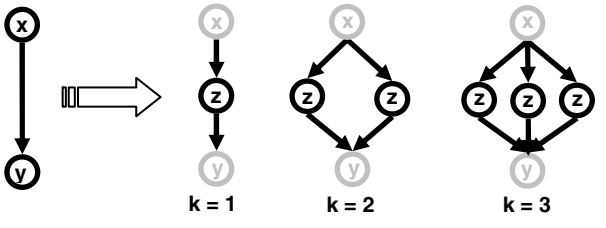


Fig. 3. Arc decompositions in nested graphs

So far we focused on the feasibility problem, that is, formalizing the constraints to be satisfied by the schedule. As we mentioned at the beginning we are in fact solving an optimisation problem with an on-time-in-full objective. This objective is reflected in the formalization of the problem as follows. Some operations, typically those describing the delivery operation, have assigned a due date with earliness and lateness costs. Assume that E is the earliness cost, L is the lateness cost, D is due date and T is the scheduled time when the operation finishes. Then the cost of the operation is $E.max(D-T,0) + L.max(T-D,0)$. Now the task is to find a feasible schedule minimizing the sum of costs of all the selected operations.

In summary there are two main issues to be solved when integrating the optimiser into a MAK€ tool. First, it is necessary to generate the scheduling model from existing data. It means converting the bill of materials, workflow descriptions, resources, and demands to scheduling concepts such as operations, temporal and resource constraints, and objectives and converting the schedule back to the enterprise

model. We solved this problem by defining translation rules that automatically transform the notions and related data between the enterprise system and the scheduling engine [8]. Though the translation may seem easy (which is really not the case if real data are assumed), it is important to highlight that users do not specify the scheduling problem and the problem is generated automatically. The second issue is solving the scheduling problem itself and this is where CP was used.

3 Why CP?

The first question one needs to ask is why the users should use new optimisation software. According to our experience the reason is never because the users want to try a new technology. This claim holds across all business areas including the most innovative ones such as space exploration [18]. It is almost impossible to go to a company with an offer of a new and theoretically better system and get the contract. There must be internal necessity inside the company that pushes the company to implement a new technology. Typically, this necessity goes from the outside of the company, from the competitors. Currently, there is an increased competition in the area of mass production from Far East. Big multi-national companies solve this problem by moving their production facilities to areas with low labour cost which helps them to decrease the cost of their products but sometimes with the trade-off of lower quality. This strategy is not applicable to small and medium enterprises (SMEs) that are typically family-owned and closely connected to their area of origin. Such companies can compete by providing high quality products that are built-to-order or even engineered-to-order so the products fit exactly the specific needs of their customers. However, this brings huge variety of produced items and increases complexity of work organization which implies that traditional scheduling methods (in these companies) based on Excel sheets and manual production of work plans are less applicable. There is also increased demand for decreasing production cost by optimising production processes and utilisation of resources. Existing enterprise resource planning tools are less applicable for SMEs as they are too expensive, too complex, and too hard to customize for SMEs so there is a need for new tools that fit well the requirements of SMEs. The MAK€ tool has already been available for several years and it is used in a couple of manufacturing companies. It used a heuristic scheduling algorithm that was capable to automatically generate schedules, but it was less flexible and too connected to existing scheduling practice (using preferred process routes) rather than providing alternative and justified solutions based on optimisation. This is where Constraint Programming enters the scene.

Scheduling is definitely not a new application area to Constraint Programming [2] and in some sense, constraint-based scheduling is the prominent and the most influential application area for CP with many successful applications [3,13,16]. The reason could be the flexibility of constraint models that allows adding side constraints appearing in real-life problems without big problem. Another reason is a support for specific scheduling constraints in existing constraint satisfaction packages [16,17]. Many specific global constraints for example for describing resources have been

proposed in recent years [2,17,20], which simplifies significantly problem modelling and brings advanced scheduling techniques to fingertips of regular CP users. The MAK€ tool brings this technology further to practitioners who are even not familiar with the optimisation techniques. The reasons to use Constraint Programming in the MAK€ tool can be summarized as:

- *flexibility*, it is easy to modify the formal model by side constraints,
- *efficiency*, it is possible to integrate specific solving techniques as additional inference (constraint propagation),
- *customization*, it is easy to use search heuristics derived from the problem specification,
- *familiarity*, it is a technology that the developer (the authors of this paper) is familiar with.

4 How CP?

As specified in the previous section, we decided for a pure CP approach to solve the problem. Though hybrid techniques are very popular and can solve some problems faster, they are also harder to implement and maintain as they require expertise from more areas. Basically, we believe that a pure solving approach is fine when the produced solutions are satisfactory which seems to be the case of the MAK€ tool. Moreover, CP technology provides enough flexibility to integrate specific inference techniques and search heuristics which is crucial for the types of problems that we are solving. In this section we will describe the core ideas of the CP model with some specific inference techniques and search strategy.

The constraint model and search strategy were implemented in SICStus Prolog 4. We used the unary resource constraint `disjoint1`, arithmetic and logical constraints from `clpfd` library of SICStus Prolog and we implemented special inference procedures for temporal constraints using the global constraints interface in SICStus Prolog.

4.1 The Core Model

For the MAK€ tool we decided for a "light" constraint model with only two types of variables (start time and validity) and a few types of constraints (integrated branching and temporal constraints, unary resource constraints, and some auxiliary constraints). This is a big difference from our previous scheduling approach used in Visopt ShopFloor system [3] which used a very complex dynamic constraint model. The main reason for the light model was fast development and easy maintenance. This gives us opportunity to focus on core features and implement them as efficiently as possible and to explore more alternatives how to implement the core constraints and search strategies. On the other hand, this approach requires some real-life features to be compiled to this light model. This is done during the translation of the enterprise model to the scheduling model. For example, the availability calendar is modelled by

auxiliary operations that are fixed in time and occupy the resource when the resource is not available. In the rest of this section, we will describe the light constraint model.

The traditional scheduling features are modelled in a standard way. For each operation we have a start time and duration variables. The domain of start time variable is defined by the time window for given operation. The domain of duration variable consists of two values: zero and the constant processing time of the operation specified in the problem formulation. These variables participate in the *resource constraint* describing the unary resource to which the operation is allocated (as we mentioned we use `disjoint1` from SICStus Prolog). Using zero duration is a method to describe optional operations in traditional resource constraints that do not support optional operations [2,20]. If duration is zero then the operation is ignored in the unary resource constraint and vice versa the resource constraint may also set duration to zero if there is not enough capacity to process the operation.

To model selection of operations we use a Boolean validity variable for each operation. This variable is assigned to 1 (*true*) if the operation is selected (then also the duration variable is non-zero for real operations) and to 0 (*false*) if the operation is not selected (then the duration variable is zero). The duration variable can model selection of operations, but we use the validity variable for simplicity reasons and also to cover auxiliary (milestone) operations, which have zero duration by definition but could be selected to the schedule. For real operations the relation between duration and validity variables is described using the constraint:

$$Val_A = 1 \Leftrightarrow Dur_A > 0$$

The validity variable together with the start time variable participates in the constraints describing the temporal network with alternatives. It is possible to use a straightforward model of branching constraints in the following form. If operation B follows directly operation A in some parallel branching then the constraint is:

$$Val_A = Val_B.$$

If *Branch* is a set of direct successors (predecessors) of operation A in alternative branching then the branching constraint can be described using arithmetic formula:

$$Val_A = \Sigma_{B \in Branch}\, Val_B.$$

Note that the above model for branching constraints does not achieve global consistency when alternative branching is present even if the TNA is nested. Assume constraints A = 1, A = B + C, B + C = D over the Boolean variables that describe a simple nest with two alternative nodes B and C. Obviously D = 1, but standard arc consistency techniques used in most constraint solvers cannot infer this information. Adding a redundant constraint A = D improves inference there, actually this constraint can substitute constraint B + C = D. In [6] we showed how such a constraint model for nested graphs can be automatically generated. We are however not using these redundant constraints in the current version of the MAK€ tool because there is no assumption about the workflows to be nested and detecting the nested structure is an expensive process for large general graphs.

Recall that the temporal relation between the start times of operations A and B is described by a pair $[a_{A,B}, b_{A,B}]$ representing minimal and maximal distance. This relation can be naturally represented using the following constraint (we assume that 0 is the schedule start point):

$$Val_A * Val_B * (Start_A + a_{A,B}) \leq Start_B \ \wedge Val_A * Val_B * (Start_B - b_{A,B}) \leq Start_A.$$

The above straightforward model is appropriate if there is only a small number of alternative branchings. For such problems, values of most validity variables are known and the constraints propagate well. However, if the number of alternative branches increases then the straightforward model contains many (hidden) disjunctions that do not propagate well in current constraint solvers. In particular, notice that the lower bound of start time variables never increases until both validity variables are set to 1. In such a situation a more *pro-active approach* to domain filtering is better. The pro-active model always filters out infeasible time points from the start time variable (even if the operation is not yet known to be in the solution) and when the domain of the start time variable is to become empty then the filtering algorithm sets the corresponding validity variable to 0 rather than emptying the domain of the start time variable and generating a failure (empty domain in constraint satisfaction means a failure which causes backtracking in the search procedure). The pro-active filtering algorithm is described in [7]. In the constraint solver, the pro-active filtering is realized as inference for the global constraint over the validity and temporal variables. The possibility of integrating such special inference procedures to the underlying constraint solver without influencing the rest of the model is one of important advantages of CP technology.

4.2 Search Strategy

There are two types of decision variables in the constraint model, the start time variables and the validity variables. The duration variables are auxiliary and their value is fully determined by the value of validity variables. Hence the search strategy focuses on selection of operations (assigning the validity variables) and time allocation (assigning the start time variables). Note that selecting the operations also means deciding the alternative resources, if they are defined. This is one of the consequences of the light constraint model – we can focus on the selection of alternative process routes and still cover features such as resource allocation.

Rather than using generic labelling procedures available in constraint solvers, we decided to implement our own search strategy driven by the objective function. The search procedure consists of two stages. In the first stage we select the operations to form the schedule and we decide their order if they share a resource. In the second stage we decide the particular start times. The first stage can be seen as a generalization of *Precedence Constraint Posting* (PCP) scheduling strategy [11] while the second stage is a known *Set a Start Time* (SST) strategy [12]. The ideas how to extend the PCP to support optional operations is described in [4]. In the MAK€ tool we are using a similar technique but the technical details are of commercial value so they cannot be revealed. Briefly speaking, we are doing left-to-right scheduling where we are ordering the operations from left (earlier times) to right (later times). We select the operation based on its minimal start time, its slack [19] and validity status (these

values depend on previous decisions that are propagated to domains of variables), and its recommended start time derived from the due date. For the selected operation, the validity variable is set to 1 and new precedence constraints are posted between this operation and all not-yet scheduled operations that share a resource with the currently selected operation (the selected operation is ordered before all not-yet scheduled operations using the same resource). If a failure is detected (via inference or later during search), the selected operation is known not be the first one so its minimal start time is increased and the operation is remembered to be after some other valid operation that will be selected later during search. The last option, if postponing the operation also fails, is setting the validity status of the operation to zero which means that the operation is not part of the solution. After the first stage successfully finishes, we obtain a set of valid (selected to the schedule) operations connected via a simple temporal network (STN) [14]. The operations are not yet allocated to precise times; this allocation is done in the second stage. We first sort the operations based on their earliness and lateness costs and in this order we try to allocate them to their due dates, if possible. More precisely, assume that we take an operation with the largest lateness cost parameter and due date D. Then we post a constraint that the operation finishes before or at D which implies that the lateness cost for this operation is zero. The alternative branch during search uses a constraint stating that the operation finishes after D. The same mechanism is used for the earliness cost. The operations are selected based on the largest lateness or earliness cost parameters and depending on which type of cost is the largest one we use one of the above two branching schemes. After allocating the operations with costs we instantiate the start times of all other operations to their minimal possible values. Thanks to strong inference of temporal constraints (recall that solving STN is tractable [14]), the second stage will never fail and will always produce a solution. Actually, for our data we obtained optimal solutions for the second stage without backtracking which justifies that this technique is indeed appropriate for this type of problems.

The above search procedure looks like a greedy search but it allows exploration of alternatives in case of failure. The failure may occur only if there are some hard deadlines (or maximal distance between some operations) otherwise it is always possible to postpone the operations with the penalty of increased cost of the solution. The search procedure can also be encapsulated in a standard branch and bound procedure where first a feasible solution is found and then we look for a better solution. However, experiments will real data showed that the first found solution has acceptable quality. To give the idea of size of problems solved by this approach, we can generate a schedule of good quality (according to customers) with more than 4000 operations and more than 30 resources in less than two minutes on standard laptops.

5 Added Value of CP

There have been many attempts to enhance the economic performance of firms with scheduling tools. Most have failed due to their limited ability to represent the typical SME where there are a large diversity of workflows and resource alternatives being applied to a very large diversity of products with tight delivery time lines. In the case of the MAK€ tool a key contribution of the CP approach was the ability to effectively represent this complex problem as one unified model. Once this initial representation

was defined the CP approach supported an iterative development of the MAK€ tool where the validation of the proposed scheduling solutions was completed on actual production problems from the SME end user partners. The structure of these actual problems tends to be quite different from the synthetic benchmark problems sets common in the literature. The end users in particular made an important contribution to the refinement of the objective function so that the CP search was focused on finding solutions that were of greatest practical significance to the SME partner firms involved rather than the more traditional makespan type objective functions. The MAK€ application has had a total of 15 years of use in five different production facilities with no change in the model being applied to each facility. This general model of enterprises exploits the expressive properties of CP and has dramatically reduced the level of customer model development required for each new implementation. This has dramatically reduced the cost of the MAK€ tool development and deployment. This has enabled the allocation of resources to creating a GUI that more closely meets the decision support needs of practitioners. The end users are presented with the consequences of the proposed scheduling solutions on a suite of Key Performance Indicators (KPIs) for the enterprise. These are the focus of the practitioners when the MAK€ tool is being used in production mode. In production mode a critical requirement of the application end users was that the scheduling solutions were generated quickly and could not be improved with manual interventions. The end user population had minimal interest in understanding the technology used to generate the schedule solutions. The ability of the decision makers to comprehensively evaluate the proposed scheduling solutions via a schedule analyser proved to be critical in establishing the credibility of the MAK€ tool with the end users. This method of presenting the proposed scheduling solution proved to be more important to end users than arguments based on optimality proofs being achieved for large populations of representative problem sets.

The payback for each facility has been very rapid since the workflow preparation effort is modest and required only once for each product whereas the MAK€ tool can be run 50-60 times per day in production mode. The typical pay back for the MAK€ tool is less than six months since the KPI focus of the application is aligned with the scheduling solutions being proposed. This is far superior to the current state of the art ERP (Enterprise Resource Planning) and manufacturing execution systems.

Acknowledgments. The research is supported by the Czech Science Foundation under the contract P202/10/1188 and by EU Funding Scheme Research for the benefit of SMEs: FP7-SME-2007-1 under the project ValuePOLE (contract 222218).

References

1. Bae, J., Bae, H., Kang, S.-H., Kim, Z.: Automatic Control of Workflow Processes Using ECA Rules. IEEE Transactions on Knowledge and Data Engineering 16(8), 1010–1023 (2004)
2. Baptiste, P., Le Pape, C., Nuijten, W.: Constraint-based Scheduling: Applying Constraints to Scheduling Problems. Kluwer Academic Publishers, Dordrecht (2001)

3. Barták, R.: Visopt ShopFloor: On the Edge of Planning and Scheduling. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 587–602. Springer, Heidelberg (2002)
4. Barták, R.: Search Strategies for Scheduling Problems with Optional Activities. In: Proceedings of CSCLP 2008 Annual ERCIM Workshop on Constraint Solving and Constraint Logic Programming, Rome (2008)
5. Barták, R., Čepek, O.: Temporal Networks with Alternatives: Complexity and Model. In: Proceedings of the Twentieth International Florida AI Research Society Conference (FLAIRS), pp. 641–646. AAAI Press (2007)
6. Barták, R., Čepek, O.: Nested Precedence Networks with Alternatives: Recognition, Tractability, and Models. In: Dochev, D., Pistore, M., Traverso, P. (eds.) AIMSA 2008. LNCS (LNAI), vol. 5253, pp. 235–246. Springer, Heidelberg (2008)
7. Barták, R., Čepek, O., Hejna, M.: Temporal Reasoning in Nested Temporal Networks with Alternatives. In: Fages, F., Rossi, F., Soliman, S. (eds.) CSCLP 2007. LNCS (LNAI), vol. 5129, pp. 17–31. Springer, Heidelberg (2008)
8. Barták, R., Little, J., Manzano, O., Sheahan, C.: From Enterprise Models to Scheduling Models: Bridging the Gap. Journal of Intelligent Manufacturing 21(1), 121–132 (2010)
9. Beck, J.C., Fox, M.S.: Constraint-directed techniques for scheduling alternative activities. Artificial Intelligence (121), 211–250 (2000)
10. Brucker, P.: Scheduling algoritms. Springer, Heidelberg (2001)
11. Cesta, A., Oddi, A., Smith, S.F.: Iterative Flattening: A Scalable Method for Solving Multi-Capacity Scheduling Problems. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), pp. 742–747. AAAI Press (2000)
12. Godard, D., Laborie, P., Nuijten, W.: Randomized Large Neighborhood Search for Cumulative Scheduling. In: Proceedings of the 15th International Conference of Automated Planning and Scheduling (ICAPS), pp. 81–89. AAAI Press (2005)
13. Delgado, A., Jensen, R.M., Schulte, C.: Generating Optimal Stowage Plans for Container Vessel Bays. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 6–20. Springer, Heidelberg (2009)
14. Dechter, R., Meiri, I., Pearl, J.: Temporal Constraint Networks. Artificial Intelligence (49), 61–95 (1991)
15. Kuster, J., Jannach, D., Friedrich, G.: Handling Alternative Activities in Resource-Constrained Project Scheduling Problems. In: Proceedings of Twentieth International Joint Conference on Artificial Intelligence (IJCAI 2007), pp. 1960–1965 (2007)
16. Laborie, P.: IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems. In: van Hoeve, W.-J., Hooker, J.N. (eds.) CPAIOR 2009. LNCS, vol. 5547, pp. 148–162. Springer, Heidelberg (2009)
17. Laborie, P., Rogerie, J.: Reasoning with Conditional Time-intervals. In: Proceedings of the Twenty-First International Florida AI Research Society Conference (FLAIRS), pp. 555–560. AAAI Press (2008)
18. Rabenau, E., Donati, A., Denis, M., Policella, N., Schulster, J., Cesta, A., Cortellessa, G., Fratini, S., Oddi, A.: The RAXEM Tool on Mars Express - Uplink Planning Optimisation and Scheduling Using AI Constraint Resolution. In: Proceedings of the 10th International Conference on Space Operations, SpaceOps 2008, Heidelberg, Germany (2008)
19. Smith, S.F., Cheng, C.-C.: Slack-Based Heuristics for Constraint Satisfaction Scheduling. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), pp. 139–144. AAAI Press (1993)
20. Vilím, P., Barták, R., Čepek, O.: Extension of O(n log n) filtering algorithms for the unary resource constraint to optional activities. Constraints 10(4), 403–425 (2005)

Knowledge Compilation with Empowerment

Lucas Bordeaux[1] and Joao Marques-Silva[1,2,3]

[1] Microsoft Research, Cambridge, UK
[2] University College Dublin, Ireland
[3] IST/INESC-ID, Lisbon, Portugal

Abstract. When we encode constraints as Boolean formulas, a natural question is whether the encoding ensures a "propagation completeness" property: is the basic unit propagation mechanism able to deduce all the literals that are logically valid? We consider the problem of automatically finding encodings with this property. Our goal is to compile a naïve definition of a constraint into a good, propagation-complete encoding. Well-known Knowledge Compilation techniques from AI can be used for this purpose, but the constraints for which they can produce a polynomial size encoding are few. We show that the notion of *empowerment* recently introduced in the SAT literature allows producing encodings that are shorter than with previous techniques, sometimes exponentially.

1 Introduction

Modeling problems with constraints is as much an art as it is a science. Very often the same relations between variables can be encoded in a wide range of possible ways. Although semantically equivalent these various encodings may present dramatic differences in terms of complexity: some encodings are "well-posed" in some sense, which guarantees a tractable reasoning on the constraints, while some others are formulated in a way that hinders the deduction mechanisms of constraint solvers.

The Question we investigate is how to automatize the search for such "well-posed" encodings: given a naïve definition of a constraint in propositional logic, how do we produce an equivalent, but "well-posed", encoding of the constraint? Both of the words *naïve* and *well-posed* require clarification. The specific notion of "well-posed" we focus on is defined precisely later in the paper by the property of *propagation completeness*, which states that the simple unit propagation rule that is at the core of SAT solvers effectively deduces all the literals that are logically entailed. The notion of "naïve" encoding is unavoidably more subjective: by this we essentially mean a concise encoding that logically captures the semantics of the constraint, but ignores any of the redundancies or "modeling tricks" that experts will usually add for performance reasons.

Producing propagation-complete encodings is an important question on which significant prior work can be found. In particular the Knowledge Compilation literature proposes a rich set of techniques for the preprocessing of logical formulas into a "compiled" form that allows certain types of reasoning, including literal and clausal entailment, to be done in polynomial time [16,8]. In parallel, recent Constraint Programming (CP) literature proposes propagation-complete SAT encodings of specific constraints [2,6,15,11,4,9,4]. Bacchus [2] makes the connection between the two areas and was the

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 612–624, 2012.

first to explicitly suggest the use of Knowledge Compilation techniques to automatically obtain propagation-complete encodings of complex constraints.

Our Contribution in this paper is to relate clausal Knowledge compilation to the notion of *empowerment*, recently introduced by [1,14]. We show that generating empowering redundant clauses is a key rewriting technique that can be used to generate "useful" redundant constraints that ultimately produce a propagation-complete formula. This observation is simple and, in retrospect, natural, but we prove that restricting the clause generation to empowering clauses can in some cases reduce exponentially the size of compiled formulas, compared to previous CNF knowledge compilation techniques.

Overview of the Paper. We start by reviewing the notation and required material in Section 2. Section 3 presents the notion of empowerment and studies the question of how to find empowering clauses. Section 4 introduces and studies the technique of compilation using empowerment, which is compared to prior CNF compilation techniques in Section 5. Finally, Section 6 outlines a number of research directions.

2 Preliminaries

This section overviews the needed preliminary material and notation.

Resolution. The paper will mostly consider logical formulas in CNF (Conjunctive Normal Form). Recall that these formulas are conjunctions of clauses, each of which is a disjunction of literals (variable or its negation). We write $\varphi \models c$ to indicate that clause c is a logical consequence of φ, i.e. all models of φ also satisfy c. We say that c is an *implicate* of φ. We denote by $\bot$ the empty clause; by $vars(\varphi)$ the set of variables that have an occurrence in a formula φ; by $|c|$ the length (number of literals) of a clause c.

Propositional *resolution* is the well-known deduction rule that deduces from two clauses of the form $A \vee x$ and $\neg x \vee B$ the consequence $A \vee B$. Additionally to this rule we implicitly use the rewriting rules of *simplification* ($x \vee x$ and $x \wedge x$ rewrite to x) and *exchange* ($x \vee y$, resp. $x \wedge y$, rewrite to $y \vee x$, resp. $y \wedge x$), which mean that conjunctions and disjunctions are effectively treated as sets. *Unit resolution*, aka unit propagation, is the special case where A or B is empty. We use the symbol $\vdash$ for deduction using resolution, with subscripts corresponding to restricted cases of resolution: in particular given a CNF formula φ we write $\varphi \vdash_1 l$ if the clause c can be deduced from φ using unit propagation, as used in SAT solvers.

Propagation-Completeness is defined as follows:

Definition 1 (Propagation-Completeness). *A formula φ is propagation-complete if for any set of literals $\{l_1, \cdots, l_k\}$ any literal d that is logically entailed can be obtained by unit propagation, i.e.,*

$$\textit{if} \quad \varphi \wedge l_1 \wedge \cdots \wedge l_k \models d \quad \textit{then} \quad \varphi \wedge l_1 \wedge \cdots \wedge l_k \vdash_1 d$$

In particular, if $\varphi \wedge l_1 \wedge \cdots \wedge l_k$ is inconsistent, and φ is propagation-complete, unit propagation deduces $\bot$. Propagation completeness has been implicitly considered in recent CP papers [10,2,6,15,11,4] because of its connection to *Domain Consistency* (aka *Generalized Arc-Consistency*): when we encode in SAT a constraint over some

Finite-Domain variables, if the encoding of the variables is a natural encoding with one Boolean variable per value in the variable's domain (as opposed to "logarithmic encodings" where b Booleans encode 2^b possible values), and if the encoding of the constraint is propagation-complete, then unit propagation on the SAT encoding effectively finds the same implications as Domain Consistency.

Knowledge Compilation. We can now define more formally the problem we are considering throughout the paper:

Problem 1. Given a CNF formula φ, produce a "compiled formula" that is equivalent to φ and propagation-complete.

This question has been extensively studied in the *Knowledge Compilation* literature; more precisely what is studied is usually a variant called *clausal entailment* (see, e.g. [8]): given a clause $c \equiv (l_1 \vee \cdots \vee l_k)$, do we have $\varphi \models c$? It is easy to see that if a formula is propagation-complete then clausal entailment on the formula can be done in polynomial time as we can simply check whether $\varphi \wedge \neg l_1 \wedge \cdots \wedge \neg l_k \vdash_1 \bot$. Conversely if a polynomial-time algorithm (not necessarily based on unit propagation) exists for clausal entailment then we can check efficiently whether $\varphi \wedge l_1 \wedge \cdots \wedge l_k \models d$ for any literal d by checking whether $(\neg l_1 \vee \cdots \vee \neg l_k \vee d)$ is entailed. Clausal entailment and propagation-completeness are therefore essentially equivalent, but tractable clausal entailment can resort to any type of polynomial-time algorithm, while propagation-completeness specifically focusses on unit propagation.

In general, if the formula φ to be compiled is arbitrary, the compilation process hits fundamental complexity limits: clausal entailment, like other intractable problems, is "non-compilable" in general, unless *NP* $\subseteq$ *P/poly* [7]; hence reaching propagation-completeness requires in the worst case an exponentially large encoding. There are nevertheless many specific constraints whose satisfiability and unit implication problems are polynomial-time solvable, and for many of them (but *not all*, as shown in [5]!), concise propagation-complete CNF encodings have been proposed. [6,15,11,4,9,4]. Problem 1 asks how we automatically find such encodings.

3 Empowering Implicates

In this section we review the notion of *empowering clause* and relate it to knowledge compilation. We next address the question: how do we compute empowering clauses for a formula?

3.1 Empowerment

It is well-known that adding logically redundant constraints (in SAT: implicates) to a problem can *in some cases* improve propagation but that not all redundant clauses are useful. Consider for instance $\varphi = \{(\neg x \vee y), (\neg y \vee z)\}$. The clause $(\neg x \vee z)$ is an implicate, however this clause does not add anything that really benefits propagation: from x we can deduce z and from $\neg z$ we can deduce $\neg x$, with or without this clause. The property that this clause is missing is *empowerment* [14]. This notion has been

introduced in a different setting, but we argue that it is the *exact* characterization of "useful" clause. Implicates such as $(\neg x \vee z)$ that are useless are said to be *absorbed* by the CNF φ.

Definition 2 (Empowerment [14]; Absorption [1]). *Let φ be a CNF formula, and $c = (l_1 \vee \cdots \vee l_k)$ be an implicate of φ. The clause c is empowering*[1] *w.r.t. φ if one of its literals l_i, called* empowered literal*, is such that:*

$$\varphi \wedge \bigwedge_{j \in 1..k,\ j \neq i} \neg l_j \quad \not\models_1 \quad l_i.$$

c called absorbed *by φ if it has no empowered literal.*

Our first observation is that empowerment is intimately related to compilation:

Definition 3 (Closure under Empowerment; Completion). *A formula ψ is said to be* closed under empowerment *if it absorbs any implicate. Given a formula φ, let ψ be a set of implicates of φ; if the formula $\varphi \cup \psi$ is closed under empowerment, then we say that it is a* completion *of φ.*

Proposition 1. *A formula is closed under empowerment iff it is propagation-complete.*

Given a CNF formula φ of size s (size here means the sum of clause lengths), and a candidate clause $c = (l_1 \vee \cdots \vee l_k)$, there are two contexts in which it may be useful to check whether c is an empowering implicate. If we do not know whether c is an implicate, then checking whether it is is coNP-complete. In some other cases, c may be *known to be an implicate*, for instance if it is obtained by application of steps of propositional resolution. Checking whether it is empowering is in this case easier: we simply have to check whether any of the l_is $(i \in 1..k)$ is obtained by propagation when we assert the conjunction $\bigwedge_{j \in 1..k, j \neq i} \neg l_j$. This can be done in time $\mathcal{O}(k \cdot s)$ since it is sufficient to run (and undo) k propagations.

3.2 Finding Empowering Implicates Using QBF

Empowerment was introduced in the context of SAT solvers, and it was noted that learnt clauses in DPLL solvers are in fact empowering [14,1]. However the literature has not, to our knowledge, proposed any *complete* method for finding empowering clauses. Such a method should fail to return an empowering implicate only when the formula is proved to be propagation-complete. We propose such a method based on an encoding to Quantified Boolean Formulae (QBF). The generated Quantified Boolean Formula asks whether there exists a clause that is a valid implicate and that is empowering. This QBF encoding is described below.

A set of variables X is assumed, with $|X| = n$, $X = \{x_1, \ldots, x_n\}$. Literals are represented as x_i^p, where $x_i \in X$, $p \in \{0, 1\}$, $x_i \equiv x_i^1$ and $\neg x_i \equiv x_i^0$. Essentially, p

[1] Reference [14] uses the term 1-empowering, but we drop the 1- prefix for simplicitiy. Also we use in fact the original formulation from a AAAI conference paper that precedes [14]: for technical reasons [14] adds the extra condition $\varphi \wedge \bigwedge_{j \neq i} \neg l_j \not\models_1 \bot$, which we do not need.

denotes the truth value of x_i that satisfies the literal associated with x_i^p. Furthermore, let $T(x_i^p)$ denote the clauses satisfied by x_i when assigned value $p \in \{0,1\}$.

Consider a CNF formula φ and a clause c. Define $\varphi^a = \varphi \cup \{\neg l_c : l_c \in c\}$ and $\varphi^c = \varphi^a \setminus \{\neg l\}$, for a given literal $l \in c$. A clause c is 1-empowering for φ if there exists $l \in c$ such that: (i) $\varphi \models c$; (ii) $\varphi^c \not\models_1 \bot$; and (iii) $\varphi^c \not\models_1 l$.

The identification of a 1-empowering clause c consists of the following main steps: (i) select the literals of clause c, among all possible sets of literals; and (ii) validate the conditions of Definition 2 for clause c. The configuration of clause c is done through a set of auxiliary variables $S = \{s_i^p \,|\, x_i \in X \land p \in \{0,1\}\}$. Clause c is defined as follows: $c = \{l_1^0, l_1^1, l_2^0, l_2^1, \ldots, l_n^0, l_n^1\}$, where $l_i^p \leftrightarrow x_i^p \land s_i^p$. Moreover, $(\neg s_i^0 \lor \neg s_i^1)$ holds for $i = 1, \ldots, n$. A complete assignment to the variables in set S decides which literals actualy integrate clause c.

Given the definition of set S, the model outlined above can be refined as follows:

$$\exists S.(\varphi \models c) \land (\varphi^c \not\models_1 \bot) \land (\varphi^c \not\models_1 l) \tag{1}$$

which captures the conditions of Definition 2. In the remainder of this section, the complete QBF is derived by specifying the following predicates (each associated with one of the conditions of 1-empowering clause):

$$\exists S.\text{Unsat}(\varphi^a) \land \text{ProperUPConfig}(\varphi^c) \land \bigvee_{l \in c} \text{NonUPImplied}(\varphi^c, l) \tag{2}$$

where $\text{Unsat}(\varphi^a)$ holds if φ^a is unsatisfiable, $\text{ProperUPConfig}(\varphi^c)$ holds if unit propagation (UP) on φ^c is non-inconsistent, and $\text{NonUPImplied}(\varphi^c, l)$ holds if l is not implied by unit propagation. Predicate $\text{Unsat}(\varphi^a)$ is represented by $\forall X.\neg\varphi^a(X)$. The other two predicates require modeling the proper outcomes of unit propagation (UP) as a propositional formula.

The selection of the set of variables to be declared assigned by unit propagation is achieved through a few sets of auxiliary variables. The first set U, is defined as follows: $U = \{u_i^p \,|\, x_i \in X \land p \in \{0,1\}\}$, where u_i^p is true iff $x_i \in X$ takes value $p \in \{0,1\}$ by unit propagation. Clearly, $(\neg u_i^0 \lor \neg u_i^1)$ holds for all $1 \leq i \leq n$. Also, if $u_i^0 = u_i^1 = 0$ then x_i is unassigned by unit propagation. Moreover, $u_{i,d}^p$ denotes whether x_i takes value p in no more than d unit propagation steps. The consistent assignments to $u_{i,d}^p$ are defined recursively as follows: (i) $u_{i,0}^p = 1$ iff (x_i^p) is a unit clause; and (ii) for $d > 0$, $u_{i,d}^p = 1$ if x_i^p was already set to 1 at earlier propagation stages than d, or if there exists a clause $c_t \in \varphi^c$ with all literals but x_i^p assigned value 0 in no more than $d-1$ unit propagation steps. Formally,

$$\begin{array}{ll} u_{i,0}^p = 1 & \text{if } (x_i^p) \in \varphi^c \\ u_{i,0}^p = 0 & \text{if } (x_i^p) \notin \varphi^c \\ u_{i,d}^p = u_{i,d-1}^p \lor \bigvee\limits_{c_t \in T(x_i^p)} \bigwedge\limits_{\substack{x_j^q \in c_t \\ x_j^q \neq x_i^p}} u_{j,d-1}^{1-q} & \text{for } d > 0 \end{array} \tag{3}$$

For simplicity of notation, in the rest of the section we simply denote by u_i^p the variables $u_{i,n}^p$, that represent the state of each x_i at the end of propagation, i.e. when $d = n$ (last "propagation level").

Let $\varphi^{u,d}(u_i^p)$ denote the set of clauses from (3). Then, the predicate ProperUP(u_i^p) is given by the conjunction of $\varphi^{u,d}(u_i^p)$, $(\neg u_i^0 \vee \neg u_i^1)$, $(u_{i,d-1}^p \rightarrow u_{i,d}^p)$, with $1 \leq d \leq n$, and $(u_i^p \leftrightarrow u_{i,n}^p)$.

For a given clause c_t, the predicate SAT(c_t) is given by $(\vee_{x_i^p \in c_t} u_i^p)$. After consistent unit propagation, all clauses *must* either be satisfied or non-unit. Define v_i to be true iff x_i is unassigned, i.e. $v_i \leftrightarrow (\neg u_i^0 \wedge \neg u_i^1)$. Then the NonUnit$(c_t)$ predicate is defined by $(\sum_{x_i^p \in c_t} v_i \geq 2)$.

The predicate ProperUPConfig(φ^c, U) can be defined as follows:

$$\bigwedge_{\substack{1 \leq i \leq n \\ p \in \{0,1\}}} \text{ProperUP}(u_i^p) \wedge \bigwedge_{c_t \in \varphi^c} (\text{SAT}(c_t) \vee \text{NonUnit}(c_t)) \tag{4}$$

The existence of a proper UP configuration is then given by: $\exists U$.ProperUPConfig (φ^c, U). Finally, assuming literal l corresponds to x_k^p, predicate NonUPImplied(φ^c, x_k^p, U) is defined as follows: ProperUPConfig$(\varphi^c, U) \rightarrow \neg u_k^p$, i.e. for any non-inconsistent unit propagation x_k^p remains unassigned. Thus, the condition that x_k is not implied, is given by $\forall U$.NonUPImplied(φ^c, l, U).

Proposition 2. *Given a formula φ, the question:* is there a clause c that is an empowering implicate of φ? *is polynomial-time reducible to a QBF formula with quantifier alternation $\exists\forall$.*

Proof. See QBF derivation above.

We also note that the QBF encoding can easily be tuned to express the existence of empowering clauses *of bounded length*: this amounts to constraining the length of the desired clause in the encoding.

4 Compilation by Iterative Empowerment

The most natural approach to knowledge compilation using empowering clauses is to iteratively add empowering clauses to the formula until it ultimately becomes propagation-complete. We call this approach *compilation by iterative empowerment*. Here we study this approach and focus in particular on a disciplined approach where empowering clauses are introduced by increasing length.

4.1 Compilation Sequences

Knowledge compilation by empowering implicate generation is highly dependent on the order in which empowering clauses are generated. The notion of *compilation sequence* captures this ordering:

Definition 4. *Let φ be a CNF formula. A* compilation sequence *is a sequence of clauses $[\varphi; c_1; \cdots; c_k]$ such that:*

- *Each clause c_i for $1 \leq i \leq k$ is an implicate of φ that is empowering wr.t. the partially compiled formula $\varphi \wedge c_1 \wedge \cdots \wedge c_{i-1}$;*

– $\varphi \wedge c_1 \wedge \cdots \wedge c_k$ *is ultimately propagation-complete.*

Example 1. Consider the formula: $\varphi \equiv \{(a \vee b), (\neg a \vee x), (\neg a \vee y), (\neg b \vee x), (\neg b \vee y), (\neg x \vee y), (\neg y \vee x)\}$. There are two possible compilation sequences for this formula: $[\varphi; (x)]$ and $[\varphi; (y)]$. In other words we may start by adding the empowering implicate (x), in which case (y) becomes deducible by unit propagation and therefore is not an empowering clause; or we may start by adding (y) in which case (x) is not empowering.

4.2 Clause Deprecation

Being empowering w.r.t. $\varphi \wedge c_1 \wedge \cdots \wedge c_{i-1}$ does not, in general, guarantee that c_i will remain empowering w.r.t. to the fully compiled formula. One issue is that as we generate empowering clauses sequentially, some clauses created at an earlier stage may become non-empowering later as more clauses are added. We call this *clause deprecation*:

Definition 5 (Clause Deprecation). *In a compilation sequence* $[\varphi; c_1; \cdots; c_k]$ *we say that a clause* c_i *is* deprecated *by the addition of clause* c_j $(j > i)$ *if* c_i *is empowering w.r.t* $\varphi \wedge \bigwedge_{h \in 1..j-1, h \neq i} c_h$ *and non-empowering w.r.t.* $\varphi \wedge \bigwedge_{h \in 1..j, h \neq i} c_h$.

Example 2. Consider the formula: $\varphi \equiv \{(a \vee b), (\neg a \vee x), (\neg a \vee y), (\neg b \vee x), (\neg b \vee y), (\neg x \vee y)\}$. The two possible compilation sequences are: $[\varphi; (y); (x)]$ and $[\varphi; (x)]$. In the first sequence, (y) is empowering w.r.t. φ but is not empowering w.r.t. $\varphi \wedge (x)$, i.e. becomes deprecated by the addition of clause (x).

In extreme examples, some poorly selected sequences can ultimately contain exponentially many deprecated clauses, as shown in the following example.

Example 3. Consider the following formulas parameterized by a size m:

$$\bigwedge_{p} \left(y \vee \bigvee_{h} x_{ph} \right) \quad \wedge \quad \bigwedge_{h} \bigwedge_{p} \bigwedge_{p' > p} (y \vee \neg x_{ph} \vee \neg x_{p'h})$$

where p ranges over $1..m$ and h ranges over $1..m-1$. (These formulas are a variant of the well-known Pigeon-Hole Principle (PHP) formulas encoding $y \vee PHP_m$.) It is clear that for this formula we can generate exponentially many clauses, and in particular many empowering ones, whereas the unit clause (y) is indeed the only meaningful implicate. Specifically, a possible compilation sequence starts by φ, then adds all clauses of the form $\left(y \vee \bigvee_{p \in 1..m-1} \neg x_{p,\delta(p)}\right)$, for all bijections δ from $1 \cdots m-1$ onto itself (i.e. permutations); then adds (y). All clauses are empowering at the time where they are added. Yet adding the final clause (y) deprecates all the previous clauses.

Example 3 shows that generating a short (here, unit) empowering clause can sometimes prevent the creation of many more empowering clauses. This suggests a strategy of *length-increasing* iterative empowerment, detailed next.

function *length_increasing_empower*(φ):
  let $\psi := \emptyset$
  for L **from** 1 **to** *width*(φ)
    while there exists a clause c of length L that is empowering w.r.t. $\varphi \cup \psi$
      $\psi := \psi \cup \{c\}$
    % minimization code optionally goes here
  return $\varphi \cup \psi$

function *minimize*(φ):
  let $\psi := \varphi$
  foreach clause c **in** ψ *% arbitrary order*
    if c is absorbed by $\psi \setminus \{c\}$
      $\psi := \psi \setminus \{c\}$
  return ψ

Fig. 1. Algorithms: *length_increasing_empower* takes a CNF φ and returns a completion of it; *minimize* takes a propagation-complete CNF φ and returns a subset of it that is equivalent, propagation-complete, and minimal

4.3 Length-Increasing Iterative Empowerment

We now consider what happens if we generate clauses *by increasing length*: we first saturate the formula under empowering clauses of length 1; only then do we consider length 2; and so forth. This is shown as Algorithm *length_increasing_empower* in Fig. 1. The algorithm is similar to the simple width-increasing algorithm resolution proposed in e.g. [3], but (1) only generates clauses that are empowering, and (2) exhaustively checks that no implicate of length L exists before incrementing L. The latter test can be done by a width-bounded QBF encoding as suggested in the previous section. This algorithm is non-deterministic in that there are many possible choices in the selection of the empowering clause at any stage. The sequences of implicates it generates can be characterized as follows:

Definition 6 (Length-Increasing Compilation Sequence). *A compilation sequence* $[\varphi; c_1; \cdots; c_k]$ *is* length-increasing *if for every* $i \in 1..k$ *we have that all implicates of length strictly less than* $|c_i|$ *are absorbed by* $\varphi \wedge c_1 \wedge \cdots \wedge c_{i-1}$.

A key property of compilation by length-increasing iterative empowerment is that it limits the effects of deprecation, in the following sense:

Proposition 3. *In a length-increasing sequence, a clause of length* b *never deprecates a clause of length* $a < b$.

4.4 Minimality

It may be desirable in some cases to compute propagation-complete formulas that are *minimal*, where removing any clause would cause the formula not to be propagation-complete anymore.

Definition 7 (Minimal propagation-complete formula). *A propagation-complete CNF formula* $\{c_1 \cdots c_m\}$ *is* minimal *if no* c_i *is absorbed by the set of remaining clauses, i.e.* $\{c_j \ : \ j \neq i\}$.

Example 2 shows that length-increasing iterative empowerment does not necessarily lead to formulas that are minimal: deprecation can happen *between generated implicates of the same length*. Minimizing a propagation-complete formula can be done by simply checking one by one, in some arbitrary order, whether any clause is absorbed by the rest of the formula, as shown in Algorithm *minimize* of Fig. 1. If a clause c is verified to be empowering during the execution of the algorithm, it is clear that it will remain empowering at the end of the execution, where more clauses have been removed; therefore considering each clause once is enough. Minimizing a formula of length s (counted in sum of clause lengths) takes time $\mathcal{O}(s^2)$ since we need to do/undo one propagation for each literal of every clause.

To construct a minimal propagation-complete formula using the length-increasing compilation approach, it is possible to interleave the generation of empowering clauses of every length with minimization steps. In Algorithm *length_increasing_empower* of Fig. 1 the minimization code can be added in the commented area. Because of Proposition 3, it is sufficient, once the generation of empowering implicates of a certain length L has completed, to verify the empowerment of clauses of length L, i.e. the **foreach** loop of Algorithm *minimize* can be restricted to clauses whose length is L.

5 Iterative Empowerment versus Prime Implicate Saturation

We now study compilation by iterative empowerment and compare it against prior implicate-based approaches to knowledge compilation. We focus for most of our results on the length-increasing compilation scheme, but do not assume minimality. Our main point is that it provides a strictly more "succinct" compilation language than prior compilation methods by prime implicates, where succinctness is defined as in [8].

5.1 Previous Compilation Schemes

A well-known way to obtain a propagation-complete formula is to saturate it by prime implicates, as was proposed by, e.g. [16,13] and suggested for the encoding of constraints by [2]. An implicate of a formula φ is a clause c that is a valid consequence, i.e. $\varphi \models c$. An implicate is *prime* if it is not subsumed by another implicate, i.e., there is no implicate c' that contains a strict subset of the literals of c. We denote by *prime*(φ) the set of prime implicates of a formula φ. (This set is uniquely defined.)

It is known that prime implicate generation can generate clauses that are useless for propagation-completeness; for instance from the formula $\varphi = (\neg x \vee y), (\neg y \vee z)$ the absorbed clause $(\neg x \vee z)$ is a prime implicate. Heuristics have been proposed to restrict the number of absorbed clauses, in particular based on the notion of *merge* resolution. When resolving two clauses $A \vee x$ and $\neg x \vee B$, the resolution step is called *non-merge* if *vars*$(A) \cap$ *vars*$(B) = \emptyset$, and *merge* otherwise. Several optimizations to prime implicate generation have been proposed in [16]; the central idea is to avoid adding to the formula some implicates that are generated by non-merge resolution.

We compare iterative empowerment to this approach. Later approaches to prime implicate generation have been proposed, for instance [12], but these approaches depart from explicit CNF generation, while it is our aim to produce CNF encodings amenable to SAT solving. ([12] uses, specifically, a compact clause representation with BDDs).

5.2 Iterative Empowerment versus Prime Implicates

We first note that length-increasing compilation sequences can never generate more clauses than prime implicate generation, as they only contain prime implicates. We then show that compilation by saturation under empowering clauses can in some cases be exponentially more compact than approaches based on prime implicate generation.

Proposition 4. *All implicates generated in a length-increasing compilation sequence are prime.*

Proof. Consider a sequence $[\varphi; c_1; \cdots; c_k]$, and clauses c_i and c_j. We assume that c_i subsumes c_j and show a contradition. Since the literals of c_i are a strict subset of those of c_j, we have $|c_i| < |c_j|$ and $i < j$, i.e. clause c_j is generated after c_i in the sequence. But c_j is not empowering w.r.t. the formula that already includes c_i: whenever c_j becomes unit, c_i either also becomes unit, or becomes inconsistent; in both cases no useful new literal can be inferred from c_j.

A class of formulas that exhibit an exponential separation between prime implicate saturation and iterative empowerment is the so-called EVEN formulas, introduced next. (Comments on the right-hand side of the formula explain the meaning of each block of clauses in this formula.) These are CNF encodings of the formula $x_1 \oplus \cdots \oplus x_n = 0$, true when the number of 1s is even.

Definition 8 (EVEN Formulas). *We denote by* EVEN_n *the following formula over the sets of variables* $X = \{x_1 \cdots x_n\}$ *and* $X = \{y_1 \cdots y_n\}$*:*

$$
\begin{array}{ll}
(\neg x_1 \vee y_1) \wedge (\neg y_1 \vee x_1) & \text{"}y_1 = x_1\text{"} \\
\wedge \bigwedge_{i \in 2..n} \left(\begin{array}{l} (\neg y_i \vee y_{i-1} \vee x_i) \wedge \\ (\neg y_i \vee \neg y_{i-1} \vee \neg x_i) \wedge \\ (y_i \vee \neg y_{i-1} \vee x_i) \wedge \\ (y_i \vee y_{i-1} \vee \neg x_i) \end{array} \right) & \text{"}y_i = y_{i-1} \oplus x_i\text{"} \\
\wedge (y_n) & \text{"output gate } y_n \text{ is true"}
\end{array}
$$

Proposition 5. *The* EVEN_n *formulas are closed under empowerment yet have exponentially many prime implicates.*

Proof. We first note that the constraint hyper-graph of these formulas is Berge-acyclic. It is well-known that for acyclic constraint networks achieving arc consistency for each constraint of the hyper-graph is enough to achieve arc-consistency for the whole network. It follows that the formula is propagation-complete; In other words any implicate is absorbed. Now there exist an exponential number of prime implicates: (1) any clause over the variables $\{x_1 \cdots x_n\}$ that has an odd number of negative literals is an

implicate. (2) these implicates are prime: if we remove any of their literals we obtain an invalid clause. (3) there are 2^{n-1} such clauses.

5.3 Iterative Empowerment versus Merge Resolution

We next consider the optimizations to prime implicate generation based on merge resolution [16]. The goal of these methods was to find a subset of the prime implicates that remains propagation-complete. However these previous approaches did not formulate the notion of empowerment, and the question is to compare them with iterative empowerment. We show that in some cases, those compilation schemes, that discard clauses that are produced by *non-merge* resolution steps, can still generate exponentially many non-empowering clauses. This is exhibited by the following class of formulas.

Definition 9 (MERGE-EVEN formulas). *We denote by* MERGE-EVEN$_n$ *the variant of* EVEN *in which every clause receives an additional positive literal* f*, such that* $f \notin X \cup Y$*; i.e.* MERGE-EVEN$_n$ *is the set of clauses* $\{(f \vee c) \ : \ c \in \text{EVEN}_n\}$.

The set of solutions of MERGE-EVEN$_n$ is exactly the union of: (1) all assignments where f is true (with any value assigned elsewhere); (2) all assignments where f is false and EVEN$_n$ holds.

Proposition 6. *The set of prime implicates of* MERGE-EVEN$_n$ *is:* $\{(f \vee A) \ : \ A \in \textit{prime}(\text{EVEN}_n)\}$

Proposition 7. *The formula* EVEN$_n$ *is closed under empowerment, but has exponentially many prime implicates; furthermore all of these implicates are produced by merge-resolution.*

Proof. Assume the existence of a non-empowering prime implicate of MERGE-EVEN. It is a clause of the form $(f \vee A \vee l)$, not subsumed by any clause of MERGE-EVEN, and where literal l is not obtained from MERGE-EVEN by unit propagation when f is false and all literals in A are false. When f is set to false the formula simplifies to EVEN, therefore it is also the case that l is not obtained from EVEN when all literals in A are false. This implies that $(A \vee l)$ is an empowering clause of EVEN not subsumed by any clause in it, which contradicts Proposition 5.

We now show that all the resolvents applied in the compilation are Merge. All clauses of EVEN include a positive occurrence of f. Any resolution on two such clauses is a merge resolution operation and produces in a clause that also includes a positive occurrence of f. Therefore all clauses in any resolution proof will include only clauses with positive occurrence of f, and make us of merge resolution exclusively.

Note also that in formula MERGE-EVEN the merge literal is always f. This means that the optimization of algorithm *FPI*$_1$ of [16] do not apply and that this algorithm also generates exponentially many absorbed implicates.

6 Conclusion and Perspectives

Clausal (CNF) formalisms played an important role in early Knowledge Compilation work, but more recent work has favoured non-clausal formalisms that are in many respects more expressive [8]. In our view clausal Knowledge Compilation remains important because of its connections to propagation-complete encodings, and to our initial Question (formalized as Problem 1). Our main findings in this paper are the following:

- The notion of *empowerment* of [1,14] sheds a new light on clausal Knowledge Compilation: several heuristics had previously been proposed to limit the "useless" redundant constraints generated by classical prime implicate methods; but they did not, to our knowledge, explicitly define "useless" — it is now clear that empowerment is *the desired property* of implicates used in compilation.
- In particular we showed that length-increasing empowerment is a knowledge compilation scheme that has many appealing features in terms of limited deprecation and easier minimization, connection to treewidth, and comparison with other prime implicate generation schemes.

Acknowledgement. This work is partially supported by SFI PI grant BEACON (09/ IN.1/ I2618), by FCT through grants ATTEST (CMU-PT/ELE/0009/2009) and POLARIS (PTDC/EIA-CCO/123051/2010), and by INESC-ID multiannual funding from the PIDDAC program funds.

References

1. Atserias, A., Fichte, J.K., Thurley, M.: Clause-learning algorithms with many restarts and bounded-width resolution. J. of Artif. Intel. Research (JAIR) 40, 353–373 (2011); preliminary version in SAT 2009
2. Bacchus, F.: GAC Via Unit Propagation. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 133–147. Springer, Heidelberg (2007)
3. Ben-Sasson, E., Wigderson, A.: Short proofs are narrow - resolution made simple. J. of the ACM 48(2), 149–169 (2001)
4. Bessiere, C., Katsirelos, G., Narodytska, N., Quimper, C.G., Walsh, T.: Decompositions of all different, global cardinality and related constraints. In: Int. Joint. Conf. on Artif. Intel. (IJCAI), pp. 419–424 (2009)
5. Bessiere, C., Katsirelos, G., Narodytska, N., Walsh, T.: Circuit complexity and decompositions of global constraints. In: Int. Joint. Conf. on Artif. Intel. (IJCAI), pp. 412–418 (2009)
6. Brand, S., Narodytska, N., Quimper, C.-G., Stuckey, P.J., Walsh, T.: Encodings of the SEQUENCE Constraint. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 210–224. Springer, Heidelberg (2007)
7. Cadoli, M., Donini, F., Liberatore, P., Schaerf, M.: Preprocessing of intractable problems. Information and Computation 176, 89–120 (2002)
8. Darwiche, D., Marquis, P.: A knowledge compilation map. J. of Artif. Intel. Research (JAIR) 17, 229–264 (2002)
9. Feydy, T., Stuckey, P.J.: Lazy Clause Generation Reengineered. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 352–366. Springer, Heidelberg (2009)
10. Gent, I.P.: Arc consistency in SAT. In: Euro. Conf. on Artif. Intel. (ECAI), pp. 121–125 (2002)

11. Huang, J.: Universal Booleanization of Constraint Models. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 144–158. Springer, Heidelberg (2008)
12. Marquis, P., Sadaoui, S.: A new algorithm for computing theory prime implicates compilations. In: Conf. on Artif. Intel. (AAAI), pp. 504–509 (1996)
13. Marquis, P.: Knowledge compilation using theory prime implicates. In: IJCAI, pp. 837–845 (1995)
14. Pipatsrisawat, K., Darwiche, A.: On the power of clause-learning SAT solvers as resolution engines. Artif. Intel. 175(2), 512–525 (2011); preliminary works in AAAI 2008 (notion of empowerment) and CP (2009)
15. Quimper, C.G., Walsh, T.: Decompositions of grammar constraints. In: Conf. on Artif. Intel. (AAAI), pp. 1567–1570 (2008)
16. del Val, A.: Tractable databases: How to make propositional unit resolution complete through compilation. In: Knowledge Representation and Reasoning (KR), pp. 551–561 (1994)

Cost-Sensitive Classification with Unconstrained Influence Diagrams*

Jiří Iša, Zuzana Reitermanová, and Ondřej Sýkora

Department of Theoretical Computer Science
Faculty of Mathematics and Physics
Charles University in Prague
Malostranské náměstí 25, 118 00 Prague, Czech Republic
{isa,reitermanova,sykora}@ktiml.mff.cuni.cz

Abstract. In this paper, we deal with an enhanced problem of cost-sensitive classification, where not only the cost of misclassification needs to be minimized, but also the total cost of tests and their requirements. To solve this problem, we propose a novel method CS-UID based on the theory of Unconstrained Influence Diagrams (UIDs). We empirically evaluate and compare CS-UID with an existing algorithm for test-cost sensitive classification (TCSNB) on multiple real-world public referential datasets. We show that CS-UID outperforms TCSNB.

1 Introduction

The problem of cost-sensitive classification plays a great role in practical applications. Though most learning algorithms assume all classification errors have the same cost, this is seldom true in real world. For example, in the area of medical or technical diagnosis, different tests, classifiers or misclassifications may have different costs or benefits. The goal of the standard cost-sensitive learning is to minimize the total cost of the classification process, when different misclassifications incur different costs. This problem can be solved by deriving cost-sensitive variants of the existing classifiers. However, such conversion must be done for each type of classifier separately, and it is a time consuming and sometimes impossible process. The other option is using a meta-algorithm for extending an existing classifier without changing it. Examples of such approach are undersampling, oversampling, or employing a specialized algorithm such as MetaCost [3]. Contrary to these techniques, we are facing the *generalized test-cost sensitive classification problem*. We optimize not only the total cost of the misclassification, but simultaneously also the total cost of the performed tests and their requirements.

Consider the following motivational example (*medical example*): A patient comes to a doctor and the doctor needs to find out, if the patient has diabetes.

* This work was supported by the grant "Res Informatica" of the Grant Agency of the Czech Republic under Grant-No. 201/09/H057 and by SVV project No. 263 314.

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 625–636, 2012.

Claiming a patient with diabetes healthy is a big mistake and incurs a heavy misclassification cost. Subjecting a healthy patient to more and more tests can become costly too: when the doctor needs to know the patient's glucose level (an attribute of the patient), she does not pay only for the test itself (e.g. for sending a sample to a lab for analysis), but also for its requirements – a blood sample must be taken first. However, once the blood is taken, other tests can be performed on it without the need to take the blood again.

In Section 2, we define the new task of generalized test-cost sensitive classification. Section 3 summarizes the related work. In Section 4, we propose CS-UID, a novel method for the generalized test-cost sensitive classification based on the Unconstrained Influence Diagram theory [10]. Section 5 discusses the experiments we performed and their results. We empirically evaluate CS-UID on multiple real-world publicly available referential datasets and compare it with TCSNB, an existing and effective method for test-cost sensitive classification [2]. We show that CS-UID outperforms TCSNB.

2 Problem Statement

This section defines the generalized test-cost sensitive classification problem, which is an extension of the standard cost-sensitive classification problem (CSC) [4]. The task is to classify samples (i.e. patients in our medical example) into a finite set of final classes $\mathcal{S} = \{s_1, s_2, \ldots, s_k\}$. The cost of reporting class s_i when the real class is s_j is defined as c_{ij}. Usually, $c_{ii} = 0$ for each $s_i \in \mathcal{S}$. The expected cost of misclassification when reporting class s_j for a sample x is defined as

$$c(s_j|x) = \sum_{i=1}^{k} c_{ij} P(s_i|x), \tag{1}$$

where $P(s_i|x)$ is the posterior conditional probability of s_i, given x. For a sample x, the goal of the standard CSC is to report a class $s_* \in \mathcal{S}$, such that

$$c(s_*|x) = \min_{s \in \mathcal{S}} c(s|x). \tag{2}$$

The expected cost of misclassification over the distribution of samples D_x is then

$$c_{MC} = \mathrm{E}\left[\min_{s \in \mathcal{S}} c(s|x)\right]_{D_x}. \tag{3}$$

In our classification scenario, there is a finite set of tests $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$. Each test $T \in \mathcal{T}$ classifies samples into a finite set of classes $\mathcal{S}_T$. These sets may be in general different for each test and completely different from the set of final classes $\mathcal{S}$. For our medical example, one of the tests may be the test of hemoglobin. This test may classify samples (i.e. patients) into three classes *high*, *medium* and *low*, according to the level of hemoglobin. The final classes are *yes* and *no*, corresponding to the patient having or not having diabetes.

Each test $T \in \mathcal{T}$ is characterized by the probability $P_T(s_T|s)$ of reporting class $s_T \in \mathcal{S}_T$ when the final class is $s \in \mathcal{S}$. Each of the tests may have zero or more requirements from the finite set $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$. For a test $T \in \mathcal{T}$, $f(T) \subseteq \mathcal{R}$ denotes the (possibly empty) set of requirements of the test T.

Each test and each requirement has a cost associated with it. More formally, for the set of requirements $\mathcal{R}$ and the set of tests $\mathcal{T}$, cost functions $c_{\mathcal{R}} : \mathcal{R} \to \mathbb{R}$ and $c_{\mathcal{T}} : \mathcal{T} \to \mathbb{R}$ are defined. When the user decides to perform a test, she must pay the cost of the test itself plus the costs of all of its requirements. If a requirement is used by more tests, the user has to pay for it only once. Return to our medical example: for the tests of hemoglobin, glucose and cholesterol, the common requirement would be to take the blood sample from the patient. While the doctor may pay a price for each of these tests, she only needs to take the blood once.

The goal of the generalized test-cost sensitive classification is to optimize the expected total cost of misclassification, requirements and tests. In our scenario, the solution is a strategy π that for a given sample x successively advises, which test (and its requirements) to perform next and which tests and their requirements to omit, depending on the tests already performed and on their results. The decision to perform or to skip a test influences the expected total cost of tests $c_T(x, \pi)$ and the expected total cost of requirements $c_R(x, \pi)$. In the end, after the user performs all the chosen tests and knows their results, π advises the final class $s_\pi \in \mathcal{S}$ with the expected misclassification cost $c(s_\pi|x)$. The goal is to find a strategy π that for a sample x minimizes the expected value of

$$c(s_\pi|x) + c_T(x, \pi) + c_R(x, \pi). \tag{4}$$

The expected overall cost of the optimal strategy over the distribution of samples D_x is then

$$\mathrm{E}\left[\min_\pi (c(s_\pi|x) + c_T(x, \pi) + c_R(x, \pi))\right]_{D_x}. \tag{5}$$

3 Related Work

While most classification algorithms treat all misclassification errors the same and they do not consider test or requirement costs, cost-sensitive learning has a long history of its own. The basic principles were described by Elkan [4], who also gives theoretical foundations for the field.

Currently, there are two major trends in cost-sensitive learning, based on the costs considered: *a)* misclassification cost only, *b)* test cost (feature cost, attribute cost). Several methods optimize the (non-uniform) misclassification costs alone [3,18,19]. Other works [15,17] treat the attribute costs alone. Few approaches [2,7,13,21] consider both misclassification cost and test cost. Contrary to our more general definition of tests and their costs, the previous test-cost sensitive methods assume that samples are tuples of attributes and the test costs are then incurred by obtaining the attribute values. These works introduce various test-strategies as processes of obtaining unknown attribute values at a cost when classifying samples [16].

Some test-cost sensitive methods adopt an optimal search test-strategy to solve the test-cost sensitive learning problem, however with high computational costs – using Markov Decision Process (MDP) [21], or PAC-learning framework [7]. Other methods adopt effective local search algorithms in their test-strategies. They are mostly based on the model of Decision trees [13,12,16,20] or are approximative versions of the exact methods [21]; an exception is Test-cost Sensitive Naive Bayes (TCSNB) [2] that extends the Naive Bayes classifier.

Our method is, similarly to TCSNB [2], based on the Bayesian theory, but it adopts an optimal search strategy. Contrary to the MDP-based optimal test strategy [21], CS-UID offers a more effective, yet optimal, search [9]. Moreover, contrary to existing test-cost sensitive methods, CS-UID solves a more general and enhanced problem (as defined in Section 2): *a)* The test costs in our case are not necessarily the costs of obtaining attribute values – our concept of test costs is much more general (see our medical example in Section 1); *b)* we consider also requirement costs.

4 Proposed Method

In this section, the CS-UID algorithm for generalized cost-sensitive classification is described. To solve a given generalized test-cost sensitive classification problem, we first encode the problem as an UID. This UID can be solved using existing solvers, such as [8] or [14]. Then, the solution of the UID is used to build a classifier for the classification problem.

4.1 Unconstrained Influence Diagrams

Unconstrained Influence Diagrams [10] are a graphical computational model developed for decision making under uncertainty. They combine the Bayesian probability inference with calculating the optimal decisions. They extend the Influence Diagrams [11] to deal with problems where the ordering of the decisions is unspecified. Instead, searching for the optimal ordering is an integral part of the solution.

An UID is a directed acyclic graph (DAG) over decision, chance, and utility variables (see Figure 1 for an example). There are two types of chance variables: observables and non-observables. Each variable has a finite set of its possible values. An edge into a decision variable represents informational precedence; an edge into a chance variable represents causal influence; an edge into a utility variable represents functional dependence. The utility variables have no children. For each chance variable, there is a conditional probability table, while for each utility (or decision) variable, there is a utility table. The goal of the UID model is to find the best linear ordering of actions (corresponding to the decision variables) and the best options of the decisions with respect to the given observations and probabilistic assumptions. This linear ordering must extend the partial temporal ordering given by the UID.

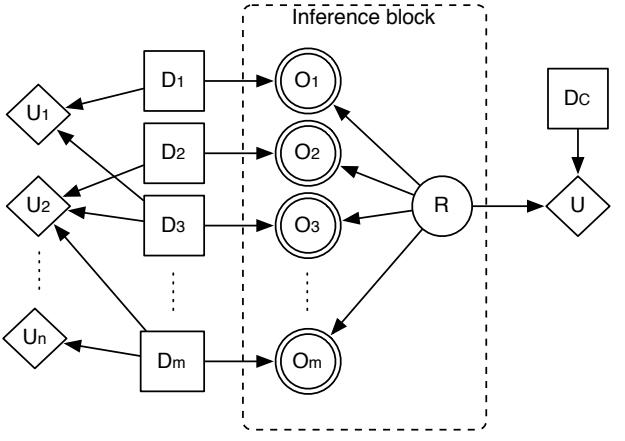


Fig. 1. The schema of the UID for cost-sensitive classification with independent test outcomes, using the notation proposed by Jensen and Vomlelová [10]. Node R represents the real final class of the sample, which is not directly observable, D_C is the decision which final class to assign the sample to, U represents utility coming from the misclassification costs. The value of U functionally depends on the real final class and on the final class selected by the classifier. Nodes $D_1, \ldots, D_m$ are decisions whether to perform or to skip the corresponding tests $T_1, \ldots, T_m$. Variables $O_1, \ldots, O_m$ represent results of the tests $T_1, \ldots, T_m$. The test results depend conditionally on the decision whether to perform the corresponding test, and on the real class R. Utilities $U_1, \ldots, U_n$ represent the costs of requirements and depend only on the decisions corresponding to tests that have these requirements.

4.2 Classification UID

Let the generalized test-cost sensitive classification problem be specified as described in Section 2. For clarity, we will first describe the method for the case, where the probabilities $P(s_T|s)$ are conditionally independent. The case where the outcomes of the tests are not independent given the final class will be discussed later in the section.

The problem can be encoded in an UID. Structure of such UID is shown on Figure 1, and, using the notation $X[x_1, \ldots, x_k]$ for a variable X with values $x_1, \ldots x_k$, it contains exactly the following elements:

- A non-observable chance variable $R[s_1, \ldots, s_k]$ that encodes the real final class. We define the probability $P(R = s_i) = P(s_i)$ for each class $s_i \in \mathcal{S}$.
- A decision variable $D_C[s_1, \ldots, s_k]$ used to select the final class. It represents the final class reported by the model. The utility $U(D_C)$ associated with D_C is zero.
- A utility node U that functionally depends on the values of R and D_C. The node U encodes the misclassification costs: $U(R = s_r, D_C = s_d) = -c_{dr}$,

where c_{dr} is the cost of reporting the final class s_d when the real final class is s_r.

– For each test $T \in \mathcal{T}$, there is a decision node $D_T[Perform, Skip]$, and an observable chance node $O_T[\mathcal{S}_T \cup \{Unknown\}]$. Variable D_T represents the decision to perform or to skip the test T, and its cost $c_{\mathcal{T}}(T)$. The corresponding utility table $U(D_T)$ contains the values $U(D_T = Skip) = 0$, $U(D_T = Perform) = -c_{\mathcal{T}}(T)$.
Variable O_T encodes the information whether the test T was performed and eventually the class reported by T. The value of O_T depends conditionally on D_T and R. The probability table associated with O_T is $P(O_T|R, D_T)$, its values are defined by the following equations for all $s \in \mathcal{S}, s_T \in \mathcal{S}_T$:

$$P(Unknown|s, Skip) = 1, \tag{6}$$

$$P(Unknown|s, Perform) = 0, \tag{7}$$

$$P(s_T|s, Perform) = P(s_T|s), \tag{8}$$

$$P(s_T|s, Skip) = 0, \tag{9}$$

where $P(s_T|s)$ is the conditional probability of T reporting s_T given the real final class is s.

– For each requirement $r \in \mathcal{R}$, there is a utility node U_r, which encodes r and its cost $c_{\mathcal{R}}(r)$. The value of U_r depends functionally on exactly all decision nodes D_T such that $r \in f(T)$; it is equal to 0 if the value of all such decisions is *Skip*; otherwise, the value of U_r is $-c_{\mathcal{R}}(r)$.

4.3 Classification of Samples

A solution of an UID u is a step-strategy π that tells what decision should be made next, and which value to assign to the decision variable. A state σ of UID u is a set of assignments of values to variables in u. Let π be a step-strategy for the UID u. For a state σ, $\pi(\sigma)$ returns a pair (D, v), where D is a decision variable in u that does not have a value assigned in σ, and v is a value that should be assigned to D.

After adding $D = v$ to σ to form a new state σ', observable chance variables that only depend on D and on variables that were already assigned a value in σ will become available. For simplicity, we assume that the values of these variables are immediately observed and added to σ'.

Given a classification UID u, there are two types of decision nodes. First, there are m decision nodes D_T corresponding to the m tests $T_1, T_2, \ldots, T_m$. Then, there is the decision node D_C for determining the final class of the sample. In such setting, a step-strategy π in a state σ either says that a test T, that was not considered until now, should be performed or skipped by returning $(D_T, Perform)$ or $(D_T, Skip)$, or determines the final class s of the sample by returning (D_C, s). Algorithm 1 shows the pseudo-code of the classification algorithm.

Algorithm 1. Classification with CS-UID

Input: UID u that encodes the classification problem, a sample x
Output: The final class s for the sample x

Solve u, yield a strategy π
$\sigma \leftarrow \emptyset$
loop
  $(D, v) \leftarrow \pi(\sigma)$
  $\sigma \leftarrow \sigma \cup \{(D, v)\}$
  if D is a decision variable D_T for test T **then**
    if $v = Perform$ **then**
      Perform test T on x, observe its outcome o
      $\sigma \leftarrow \sigma \cup \{(O_T, o)\}$
    else
      $\sigma \leftarrow \sigma \cup \{(O_T, Unknown)\}$
    end if
  else {D is the final decision D_C, v is the final class}
    Return v as the final class of x
  end if
end loop

4.4 Full-Powered Bayesian Inference

The schema on Figure 1 shows the simpler case of CS-UID, where the outcomes of the tests are conditionally independent given the final class. This condition is rarely fully satisfied in real-world applications, which can influence the quality of the achieved results – they may become non-optimal.

In the general case, the "Inference block" on Figure 1 can be replaced by a full-featured Bayesian network designed for the particular classification problem. In such case, an exact UID solver finds an optiomal solution of the problem.

4.5 Remarks

To use the CS-UID method, an UID solver is needed. Currently, there are two types of UID solvers: exact methods [10] and approximative or anytime methods [1,6]. The exact methods are able to find an optimal strategy, but with high computational costs [8,9]. The approximative or anytime methods are generally faster, but the optimality of the solution is not granted.

For the experiments conducted in this paper, we have used the solution method proposed by Jensen and Vomlelová [10] as implemented by Iša, et al. [8]. This method is an exact solution method that calculates the optimal strategy. It follows from the UID solving algorithm, that the classification decision (the variable D_C) will always be made after all tests are considered. Moreover, the final class is for the simpler CS-UID determined the same way as when applying the Naive Bayes classifier to the results of the tests that were performed.

In the case of zero requirement costs, this property makes the simpler CS-UID remarkably similar to TCSNB [2], but these methods do not advise the same

Table 1. Datasets used for the experiments

Dataset	Classes	Attributes	Samples
balance-scale	3	4	625
car	4	6	1,728
hayes-roth	3	4	160
monks-1	2	6	432
monks-2	2	6	432
monks-3	2	6	432
nursery	5	8	12,960

tests to be performed: TCSNB employs a greedy search strategy, while CS-UID uses a complex search strategy that considers possible outcomes of all tests. For illustration, consider a classification problem with four equiprobable final classes s_1, s_2, s_3, s_4, and tests T_1, T_2, $c_{\mathcal{T}}(T_1) = c_{\mathcal{T}}(T_2) = 10, c_{ij} = 36$ for $i \neq j$. Each test classifies samples into two classes *t*, *f*. The test T_1 returns *t*, if the real class is one of s_1 or s_2, otherwise it returns *f*; T_2 returns *t*, if the real class is one of s_1 or s_3, otherwise it returns *f*. By using both tests, the real class can be determined correctly with cost 20 for any sample. This is the optimal strategy and it is the solution CS-UID comes up with. When only one of the tests T_1, T_2 is used, the probability of misclassification is $\frac{1}{2}$ and expected overall cost is $10 + 36 * \frac{1}{2} = 28$; when none of the tests is used, the probability of misclassification is $\frac{3}{4}$ and expected overall cost is $36 * \frac{3}{4} = 27$. TCSNB will decide to perform no test.

5 Experiments

The goal of our experiments is to evaluate the performance of the CS-UID method, assess its behavior on real-world datasets and compare it to the performance of TCSNB [2]. Comparison between CS-UID and the MDP approach [21] was not performed, because:

1. both models are equivalent and thus provide solutions that have the same expected cost,
2. when m is the number of tests, the exact AO-tree used in [21] requires $O(m!)$ leaves, while an exact UID solver only requires $O(m2^m)$ nodes [9].

Seven publicly available datasets from the UCI Machine Learning Repository [5] were employed: These datasets are widely used as benchmarks for the uniform-cost classification. However, they lack information needed for the cost-sensitive classification. They provide neither the misclassification costs, nor the costs of tests and requirements. To deal with the issue, we extended the approach used by Domingos [3] and generated the costs and the requirements stochastically. To obtain reproducible results, we preferred generated costs on the public datasets over proprietary data.

Domingos [3] distinguishes two types of misclassification costs: *a) uniform* – the misclassification costs do not depend on the class probabilities, *b) proportional* – misclassifying a rare class (e.g. a serious disease as "healthy") is penalized strongly. In the *uniform* case, the misclassification costs are generated randomly from the uniform distribution in the $[0; 1,000]$ interval. The costs of correct classification c_{ii} are kept at 0.0. In the *proportional* case, the misclassification costs of labeling a sample as s_j instead of s_i (for $i \neq j$) come from the uniform distribution in the $\left[0; 1,000 * \frac{P(s_j)}{P(s_i)}\right]$ interval. $P(s_k)$ denotes prior probability of class $s_k \in \mathcal{S}$. [1]

For the experiments, we take the discrete attributes in the datasets for test outputs – each attribute serves as an outcome of one test. Because there exist no requirements for these tests, we created a set of ten (arbitrary number) artificial requirements. Every test needs each of the requirements with a fixed (arbitrary) probability of 0.1. This way we obtained requirements shared among the tests. The costs of tests and requirements are generated from the $[0; 10]$ interval uniformly.

We distinguish three levels of cost-sensitive classification: *a) zero* – zero costs of tests and requirements, non-zero misclassification costs, *b) zero-requirements* – zero costs of requirements, non-zero costs of tests and non-zero misclassification costs, *c) non-zero* – all costs non-zero. The two types of misclassification costs together with three misclassification levels give us six different setups.

We performed the experiments using the non-optimal simplified schema of CS-UID, as described in Section 4.2. This models real-world situations, in which no domain experties is available. Moreover, it is similar to typical applications of the Naive Bayesian classifier – the solution they obtain is not optimal, but often works well in practice.

For the experiments, we had to extend the TCSNB algorithm to take costs of requirements into account – the algorithm described by Chai, et al. [2] considers only test costs. We extended the algorithm in such a way, that when considering a test, the cost of that test includes costs of all its requirements that were not paid for yet. In case of zero requirement costs, this algorithm degrades exactly to the algorithm described by Chai, et al. [2].

The reported results come from a 10-fold cross-validation. To restrict the influence of randomly generated costs, the costs were generated with twenty different seeds for every dataset and the results are averaged over these twenty runs. The varying costs cause a large variance in the results over the ten folds and twenty cost variants. To verify the significance of the comparisons, a paired Wilcoxon test is used – the total classification costs for each individual sample are compared one by one. The significantly better results (with 0.95 confidence) are printed in bold in the tables with the results.

[1] The *nursery* dataset contains one class, which appears only at two samples out of the total 12961. Its extremely low prior causes extremely large penalties for the misclassification of these two samples. Consecutively, the correct decision would be to classify all samples to this rare class to avoid the penalties. We removed the two samples to obtain a richer environment.

Table 2. Average cost in the *zero::uniform* and *zero::proportional* setups. Significantly better results are printed in bold.

Setup	*zero::uniform*		*zero::proportional*	
Dataset	**CS-UID**	**TCSNB**	**CS-UID**	**TCSNB**
balance-scale	**67.23**	75.68	**94.98**	101.71
car	**56.89**	63.23	**16.13**	16.45
hayes-roth	**83.26**	84.49	**84.15**	92.94
monks-1	105.36	**105.12**	104.97	104.78
monks-2	151.36	**143.48**	141.33	142.49
monks-3	15.43	15.46	15.26	15.34
nursery	**43.61**	47.27	**18.02**	18.56

Table 3. Average cost in the *zero-requirements::uniform* and *zero-requirements:: proportional* setups. Significantly better results are printed in bold.

Setup	*zero-req.::uniform*		*zero-req.::prop.*	
Dataset	**CS-UID**	**TCSNB**	**CS-UID**	**TCSNB**
balance-scale	**85.31**	98.72	**96.04**	97.51
car	**71.00**	96.02	**27.62**	29.34
hayes-roth	97.06	100.08	**96.37**	111.07
monks-1	107.95	107.95	107.92	107.92
monks-2	145.52	**141.43**	145.60	**143.44**
monks-3	**24.25**	27.55	**24.06**	29.45
nursery	**58.51**	65.02	**25.90**	26.09

Tables 2, 3 and 4 show the experimental results of CS-UID and TCSNB on all six setups.

From the Tables 2 and 4 we see that CS-UID mostly outperforms TCSNB not only in the *non-zero* setup it was designed for, but also on the other setups. There are few exemptions of the rule – all regarding the *hayes-roth* dataset and the *monks-x* variants. The *monks-x* datasets are artificially generated using rules such as "$a_1 = a_2$ *or* $a_5 = 1$" and such as they are not suited for naive Bayes classification. Taking mostly costs into account, CS-UID and TCSNB perform equally weakly. The *hayes-roth* dataset does not fulfill the assumption of the test results (i.e. attributes in our experiments) being conditionally independent given the final class. The exact CS-UID solver we used for the experiments takes all the future decisions and their impact into account. When the obtained information is not independent, the real information gain of performing multiple tests (and paying for them) does not meet the expectation. Thus CS-UID is willing to pay for multiple tests, while it does not receive the expected information in return. Meanwhile, the greedy TCSNB does not even consider such a cumulated impact of performing multiple tests. This prevents it from paying for the (unavailable) extra information.

Table 4. Average cost in the *non-zero::uniform* and *non-zero::proportional* setups. Significantly better results are printed in bold.

Setup	*non-zero::uniform*		*non-zero::proportional*	
Dataset	**CS-UID**	**TCSNB**	**CS-UID**	**TCSNB**
balance-scale	**106.12**	113.70	**81.99**	92.87
car	**87.31**	98.10	**38.18**	43.24
hayes-roth	116.15	**111.65**	**111.75**	114.60
monks-1	**114.78**	115.76	114.85	114.90
monks-2	**144.71**	145.70	140.88	**140.83**
monks-3	**36.29**	41.14	**37.44**	38.86
nursery	**73.98**	77.67	**34.42**	37.65

The real-world datasets other than *hayes-roth* reveal, that CS-UID may be used even in some situations when the tests are not conditionally independent. This matches a common usage of the Naive Bayes classifier, whose assumptions are also rarely satisfied, yet it often works very well.

6 Conclusions and Future Work

In this paper, we defined the generalized test-cost sensitive classification problem, and we proposed CS-UID – a new method to solve it. The method uses Unconstrained Influence Diagrams as a base for deriving an optimal test strategy. We have described the method in detail for the case where all tests are conditionally independent. We evaluated performance of the new method on multiple real-world data sets and compared it with TCSNB [2], an existing test-cost sensitive classification algorithm. Our experiments revealed that CS-UID gives good classification results even when the independence condition does not hold, and it outperforms TCSNB on most data sets.

Due to the time and memory complexity of the used UID solution algorithm with respect to the number of tests [9], the experiments had to be performed on datasets with a limited amount of attributes ("tests"). If a larger problem needs to be dealt with, either a more advanced hardware and more time needs to be given to the solver, or an approximate or even any-time solver needs to be used.

References

1. Ahlmann-Ohlsen, K.S., Jensen, F.V., Nielsen, T.D., Pedersen, O., Vomlelová, M.: A Comparison of two Approaches for Solving Unconstrained Influence Diagrams. Int. J. of Approximate Reasoning 50(1), 153–173 (2009)
2. Chai, X., Deng, L., Yang, Q., Ling, C.X.: Test-Cost Sensitive Naive Bayes Classification. In: IEEE Int. Conf. on Data Mining, pp. 51–58 (2004)

3. Domingos, P.: MetaCost: A General Method for Making Classifiers Cost-Sensitive. In: Proc. 5th Int. Conf. on Knowledge Discovery and Data Mining, pp. 155–164 (1999)
4. Elkan, C.: The Foundations of Cost-Sensitive Learning. In: Proc. 17th Int. Joint Conf. on Artificial Intelligence, pp. 973–978 (2001)
5. Frank, A., Asuncion, A.: UCI machine learning repository (2010), http://archive.ics.uci.edu/ml
6. Fried, V.: Approximate solution of Unconstrained influence diagrams. Master's thesis, Fac. of Math. and Phys., Charles Univ., Prague, Czech Republic (2006)
7. Greiner, R., Grove, A.J., Roth, D.: Learning Cost-sensitive Active Classifiers. Artificial Intelligence 139(2), 137–174 (2002)
8. Iša, J., Lisý, V., Reitermanová, Z., Sýkora, O.: Unconstrained Influence Diagram Solver: Guido. In: Proc. 19th IEEE Int. Conf. on Tools with Artificial Intelligence, vol. 1, pp. 24–27. IEEE Computer Society, Washington, DC, USA (2007)
9. Iša, J., Reitermanová, Z., Sýkora, O.: On the Complexity of General Solution DAGs. In: Proc. 8th IEEE Int. Conf. on Machine Learning and Applications, pp. 673–678 (2009)
10. Jensen, F., Vomlelová, M.: Unconstrained Influence Diagrams. In: Proc. 18th Annu. Conf. on Uncertainty in Artificial Intelligence, pp. 234–241. Morgan Kaufmann, Edmonton (2002)
11. Jensen, F.V., Graven-Nielsen, T.: Bayesian Networks and Decision Graphs. Information Science and Statistics. Springer, New York (2007)
12. Ling, C.X., Sheng, V.S., Yang, Q.: Test Strategies for Cost-Sensitive Decision Trees. IEEE Trans. on Knowledge Data Engineering 18(8), 1055–1067 (2006)
13. Ling, C.X., Yang, Q., Wang, J., Zhang, S.: Decision Trees with Minimal Costs. In: Proc. 21st Int. Conf. on Machine Learning, pp. 4–8. Morgan Kaufmann (2004)
14. Luque, M., Nielsen, T.D., Jensen, F.V.: An Anytime Algorithm for Evaluating Unconstrained Influence Diagrams. In: Proc. 4th European Workshop on Probabilistic Graphical Models, Hirtshals, Denmark, pp. 177–184 (2008)
15. Núñez, M.: The Use of Background Knowledge in Decision Tree Induction. Machine Learning 6, 231–250 (1991)
16. Sheng, V.S., Ling, C.X., Ni, A., Zhang, S.: Cost-Sensitive Test Strategies. In: Proc. 21st Nat. Conf. on Artificial Intelligence, pp. 482–487. AAAI Press (2006)
17. Tan, M.: Cost-Sensitive Learning of Classification Knowledge and Its Applications in Robotics. Machine Learning 13, 7–33 (1993)
18. Ting, K.M.: Inducing Cost-Sensitive Trees via Instance Weighting. In: Proc. 2nd European Symposium Principles of Data Mining and Knowledge Discovery, pp. 139–147 (1998)
19. Ting, K.M.: An Instance-Weighting Method to Induce Cost-Sensitive Trees. IEEE Trans. on Knowledge Data Engineering 14(3), 659–665 (2002)
20. Turney, P.D.: Cost-Sensitive Classification: Empirical Evaluation of a Hybrid Genetic Decision Tree Induction Algorithm. Journal of Artificial Intelligence Research, 369–409 (1995)
21. Zubek, V.B., Dietterich, T.G.: Integrating learning from examples into the search for diagnostic policies. Journal of Artificial Intelligence Research 24, 263–303 (2005)

Modeling and Predicting Students Problem Solving Times*

Petr Jarušek and Radek Pelánek

Faculty of Informatics, Masaryk University Brno

Abstract. Artificial intelligence and data mining techniques offer a chance to make education tailored to every student. One of possible contributions of automated techniques is a selection of suitable problems for individual students based on previously collected data. To achieve this goal, we propose a model of problem solving times, which predicts how much time will a particular student need to solve a given problem. Our model is an analogy of the models used in the item response theory, but instead of probability of a correct answer, we model problem solving time. We also introduce a web-based problem solving tutor, which uses the model to make adaptive predictions and recommends problems of suitable difficulty. The system already collected extensive data on human problem solving. Using this dataset we evaluate the model and discuss an insight gained by an analysis of model parameters.

1 Introduction

Problem solving is an important component of education. To make problem solving activities attractive, it is important to confront students with problems of suitable difficulty – neither too easy, nor too difficult (see the flow concept [3,4]). Since students vary in their skills, it is crucial to make problem recommendations individually adaptive.

Our main aim in this paper is to predict a difficulty of problems, more specifically to predict a time it will take a student to solve a problem. We aim to do the prediction based on previous data about problem solving activity of this and other students. To this end we model a relation between a problem solving ability and a time to solve a problem. As a concrete application of the proposed model we develop a problem solving tutor – an online application for enhanced learning.

To make our setting clear, we describe one of the problems that we use in our tutoring application (and also in evaluation in this paper). The goal of the problem "Graphs and functions" (see Fig. 1) is to identify a formula for describing a function, which is specified by its graph. As a tool for solving students may use interactive graph drawing facility which plots their attempts to the graph[1]. By solving these puzzles students train their ability to visualize math functions and deduct formulas from visualizations.

* This work is supported by GAČR grant No. P202/10/0334.

[1] The reader can try the problem at `tutor.fi.muni.cz`

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 637–648, 2012.

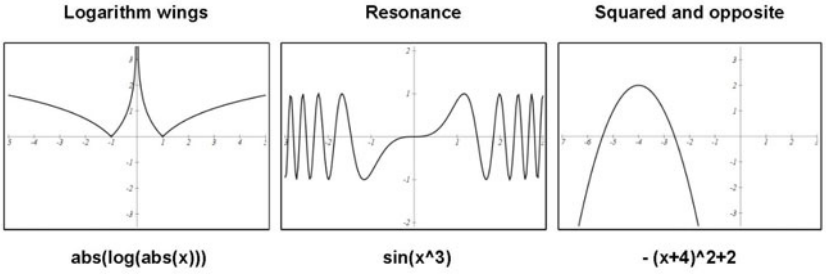


Fig. 1. Three instances of the problem "Graphs and functions" and their solutions. Note that a title is sometimes used to give students a hint.

Our work is related to four research areas, but has significant differences from each of them. *Item response theory* (IRT) [2,5] is used particularly for computerized adaptive testing (i.e., for measuring student latent ability). IRT considers tests where each question (item) has several possible answers. IRT models give relation between student ability and a probability of a correct answer. Our model is directly inspired by IRT, but there is an important difference. The IRT focuses on tests with correct and incorrect answers, whereas we study problem solving and measure a time to solve a problem (as illustrated above on the "Graphs and functions" problem). The most relevant aspect of IRT are models of response times [15,16] (which are discussed in more detail in Section 2.3). Unlike IRT which is primarily applied for adaptive testing, we are interested in intelligent tutoring.

Intelligent tutoring systems [1] are computer programs used to make learning process more adaptive and student oriented. They provide background information, problems to solve, hints, and learning progress feedback. Well known example of an intelligent tutoring system is a system for teaching algebra [9,8]. Tutoring systems usually have static structure which is determined by an expert in a particular domain. Our system is dynamic and recommends problems based on collected problem solving data. Most research on tutoring systems focuses on the "inner loop" (how to give hints about a problem), we focus solely on the "outer loop" (how to dynamically order problems) [17].

Recommendation systems [7] are used mainly in e-commerce. These systems recommend to users products that may be interesting for them (e.g., books on Amazon, films on Netflix). One of the approaches to recommendation – collaborative filtering – is based on the use of data on user behaviour. With this approach a recommender system at the same time collects data and uses these data to make predictions and recommendation. We build our system in the same way, although we are not interested in recommending products, but problems of suitable difficulty. This approach is in contrast with the mainly linear approach (collect data, calibrate models, use models) used in IRT and in education in general.

Research in *human problem solving* [14] so far focused mainly on analysis and computational modeling of human problem solving for a particular problem, e.g.,

Tower of Hanoi [10], Chinese ring puzzle [11], Fifteen puzzle [13], Sokoban [6], Sudoku [12]. This research provides insight into problem difficulty, but the insight is usually closely tied to a particular problem.

We combine principles from the above mentioned areas in the following way. We build *tutor* for practising *problem solving* which *recommends* users problems of suitable difficulty, based on the previously collected data and using a novel model, which is a variation on models used in the *item response theory*.

Our specific contributions are the following. We propose a general setting for modeling problem solving times and three specific models. For these models we discuss methods for parameter estimation and provide evaluation on large problem solving data. We also present direct application of the model – a problem solving tutor (`tutor.fi.muni.cz`).

2 Modeling Problem Solving Times

In this section we describe our approach to modeling problem solving times. It is analogical to models used in the item response theory, but instead of modeling probability of a correct answer, we model a time to solve a problem.

2.1 Summary of Item Response Theory

We start by summarising main principles of the item response theory. We focus only on aspects relevant to our model and we provide simplified description of the theory. The item response theory is a tool for designing, analyzing, and scoring tests, questionnaires, and similar instruments that measure abilities [2]. One of its main applications is computerized adaptive testing – selection of questions in test is done dynamically based on answers of a student.

Main assumption of IRT is that a given test measures one latent ability θ. IRT models response to one item (test question) as a relation between this ability θ and probability P that the item is correctly answered (basic models consider only dichotomous questions). This relation is expressed by an item response function. The basic model in IRT is 3 parameter logistic model (see also Fig. 2):

$$P_{a,b,c,\theta} = c + (1-c)\frac{e^{a(\theta-b)}}{1+e^{a(\theta-b)}}$$

This model has three parameters: b is the basic difficulty of the item, a is the discrimination factor (slope of the curve, how well the item discriminates based on ability), and c is the pseudo-guessing parameter (lower limit of the curve, probability that even a student with very low ability will guess the correct answer). Other two often used models are derived from this model by fixing values of some parameters: a two parameter logistic model ($c = 0$) and a one parameter logistic model ($c = 0, a = 1$).

To apply these models, it is necessary to estimate values of their parameters. Since we do not know neither the item parameters (a, b, c), nor student's abilities (θ), we need to estimate both of these at the same time. This is usually done

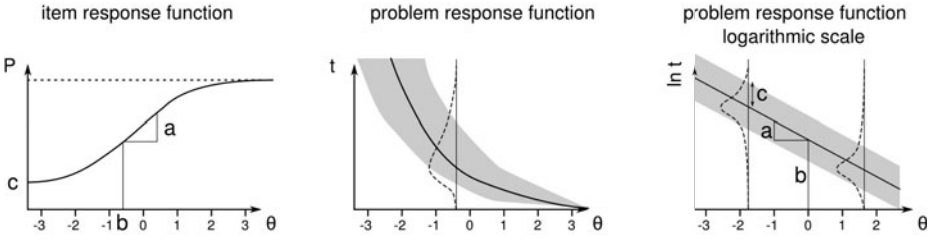


Fig. 2. Intuitive illustration of item response function, general problem response function, and a specific problem response function under our assumptions. Dashed lines illustrate distributions at certain points; solid lines denotes the median of a time distribution, grey areas depict the area into which most attempts should fall.

by joint maximum likelihood estimation, which proceeds by repeating two steps: estimating abilities from item parameters and estimating item parameters from abilities. These steps are repeated until parameter values converge.

An important feature of IRT models is group invariance – item parameters do not depend on a subset of students which answered the item, i.e., even if some item is answered only by above-average students, the estimated item parameters should be similar as if the item was answered by a representative subset of students.

2.2 Problem Solving Times

There are many extensions of the basic IRT models, e.g., models for items with polytomous answers or models which take into account response times. But none of these models is directly applicable to the problem solving setting. We propose a model, which relates problem solving ability and a time to solve a problem. At the moment we study only students time to solve a problem and not a quality of solutions (i.e., the current theory is applicable only to well-structured problems with easily verifiable solutions like the "Graphs and Functions" problem in Fig. 1).

The basic principles are analogical to the above mentioned principles of IRT. Similarly to IRT, we assume that a problem solving performance depends on one latent problem solving ability θ. We are interested in "problem response function" $f(\theta)$, which for a given ability θ gives an estimate of a time to solve a problem. More specifically, the function gives a probabilistic density of times. Fig. 2 gives a comparison of basic setting of IRT and our model.

2.3 Specific Assumptions and Model

To obtain a specific model we make the following two assumptions. First, the distribution of solving times $f(\theta)$ for students with a fixed ability θ is a log-normal distribution. Second, the mean and variance of the distribution $f(\theta)$ are exponentially dependent on θ (this dependency can be, of course, changed

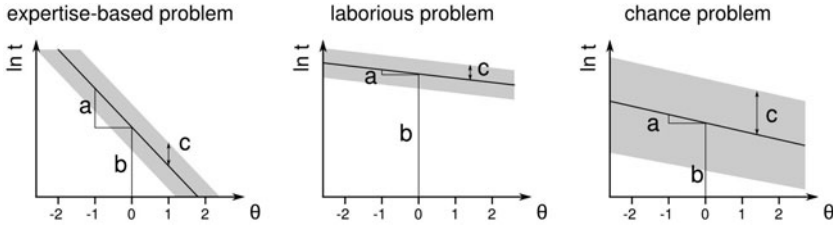


Fig. 3. Examples of different problem types and their modeling using the 3 parameter model. See Fig. 6 for specific examples.

by rescaling θ; we implicitly assume that problem solving ability is normally distributed in the population).

These assumptions are grounded on our data about human problem solving from our previous experiments [6,12] and on experience in modeling response times in IRT [16]. Moreover, the assumptions lead to a tractable model with nice properties – by using logarithm of time we obtain linear relationship and normal distributions.

Based on these assumptions, we can now specify a concrete model. Our basic model is a 3 parameter model in which the intuitive meaning of the parameters is the following (we intentionally use notation analogical to IRT): discrimination factor a, basic difficulty of the problem b, randomness factor c.

The problem response function, i.e., the probability that a student with ability θ will solve a problem at a logarithm of time t, is given by a normal distribution with a mean $b + a\theta$ and a variance c^2. Thus we have:

$$f_{a,b,c,\theta}(\ln t) = \mathcal{N}(a\theta + b, c)(\ln t) = \frac{1}{\sqrt{2\pi}c} e^{-\frac{(\ln t - (a\theta+b))^2}{2c^2}}$$

This model and intuition behind its parameters are illustrated in Fig. 2. Discrimination factor a describes the slope of the function, i.e., it specifies how the problem distinguishes between students with different ability. Basic difficulty describes expected solving time for student with average ability. Randomness factor describes variance in solving times for particular ability. The model is relatively simple, yet it can capture different aspect of problem difficulty and their combinations (see Fig. 3).

The presented model is not yet identified as it suffers from the "indeterminacy of the scale" issue in the same way as the basic IRT model, e.g., we can multiply θ and divide a by k without any effect on the model. Thus we further require the following normalization – for a set of problem parameters b_i, a_i, c_i and student parameters θ_j, we require that the mean of θ_j is 0, the mean of a_i is -1.

Similarly to IRT, we can also use simplified models. A two parameter model is obtained by using a constant randomness factor k, a one parameter model is obtained by using constant randomness and discrimination factors:

$$f_{a_i,b_i,\theta}(\ln t) = \mathcal{N}(b_i + a_i\theta, k)(\ln t) \qquad f_{b_i,\theta}(\ln t) = \mathcal{N}(b_i - \theta, k)(\ln t)$$

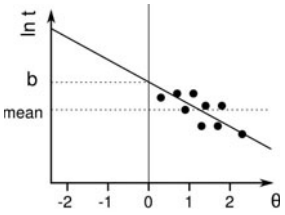


Fig. 4. If a problem is solved by above-average persons, the mean time underestimates the difficulty of a problem, whereas our model can capture it correctly

Our model is similar to van der Linden's model for response times in IRT [15]. There are two main differences. First, he uses the model in a context of testing, where the timing information is just supplementary to information about correctness of an answer, whereas in our case timing is the main focus. Second, his model has just two parameters (basic difficulty and randomness).

2.4 Group Invariance

An important feature of the approach is that the models are group invariant (analogically to IRT), i.e., parameters of a problem do not depend on a subgroup of students which solve the problem.

Let us explain this important feature by comparing our 1 parameter model to the baseline metric of a problem difficulty – the mean time to solve the problem (Fig. 4). In both cases the problem difficulty is captured by one number – by difficulty parameter b in our model or by the mean m. If we have a set of problems, then it typically happens that harder problems are solved only by students with above-average ability. In this case the mean time underestimates the real difficulty of the problem, whereas our mode is not biased by the selection of students.

3 Parameter Estimation

Since we do not know neither parameters of problems, nor parameters of students, we need to estimate them. To compute these estimates we use data of the following type: problem i was solved by a student j in time t_{ij}. From these data we need to estimate both problem parameters a_i, b_i, c_i and student parameters θ_j.

One way to do this is to apply a generic data-fitting method like non-linear least squares directly on the model and to use existing software implementations to compute estimates. Here we discuss an alternative iterative approach which is analogical to the joint maximum likelihood calculation in IRT. Advantage of the iterative approach is that it computes estimates for each student (problem) separately from others, so it is possible to update estimates locally without recomputing the whole set of parameters – this is a useful feature for the

application of the approach in our problem solving tutor, which needs to make prediction in realtime. Moreover, the iterative approach gives better insight into the computation.

3.1 Estimating Ability

Suppose that a student solved n problems, where i-th problem has parameters a_i, b_i, c_i and was solved in time t_i. Based on these data we want to estimate the ability θ of the student. We do this by finding a maximal likelihood θ. The likelihood of the observed times $t_1, \ldots, t_n$ given our 3 parameter model is:

$$L = \prod_{i=1}^{n} f_{a_i,b_i,c_i,\theta}(\ln t_i) = k \prod_{i=1}^{n} \frac{1}{c_i} e^{-\frac{(\ln t_i - (b_i + a_i\theta))^2}{2c_i^2}}$$

We need to find the value of θ such that L is maximized. As is usual, we proceed by finding maximum of $\ln L$ (which is the same as maximum of L):

$$\ln L = k + \sum_{i=1}^{n} \ln \frac{1}{c_i} + \frac{1}{2c_i^2}(a_i^2\theta^2 + 2a_i\theta(\ln t_i - b_i) + (\ln t_i - b_i)^2)$$

Since this is a quadratic function in θ, we can find maximum by finding the value of θ for which the derivation is zero:

$$\frac{\ln L}{\partial\theta} = \sum_{i=1}^{n} \theta\frac{a_i^2}{c_i^2} + \frac{a_i}{c_i^2}(\ln t_i - b_i) = 0 \qquad \theta = \frac{\sum_{i=1}^{n} \frac{a_i^2}{c_i^2}\frac{\ln t_i - b_i}{a_i}}{\sum_{i=1}^{n} \frac{a_i^2}{c_i^2}}$$

The resulting expression for θ has a clear intuitive interpretation. The expression $(\ln t_i - b_i)/a_i$ is a local estimate of ability for i-th problem – it is the value of θ for which the expected logarithm of time is $\ln t_i$. The overall estimate of θ is obtained as a weighted average of these local estimates, where the weight is given by the expression a_i^2/c_i^2, i.e., the more discriminating and less random a problem is, the more weight it gets (which is exactly what one would intuitively expect). For the one parameter model model this expression simplifies to $\theta = (\sum_{i=1}^{n} b_i - \ln t_i)/n$.

3.2 Estimating Problem Parameters

Suppose that a problem was solved by n students, where j-th student has ability θ_j and solved the problem in time t_j. Now we want to estimate problem parameters a, b, c. Maximal likelihood estimates can be found by a regression analysis. For the two and three parameter models we can use standard linear regression (least square method), because for our model (linear dependence with normally distributed errors) the least square method gives maximal likelihood estimation. Parameter c is then estimated from error residuals.

For the one parameter model we are looking for linear regression line with a fixed slope $a = -1$, thus we need to minimize the following sum of squares: $\sum_{j=1}^{n}(\ln t_j - (b - \theta_j))^2$. This is a quadratic function with a minimum at $b = (\sum_{j=1}^{n} \ln t_j + \theta_j)/n$.

3.3 Joint Estimation

So far we assumed that either abilities are known exactly and we estimate problem parameters, or that problem parameters are known exactly and we estimate student ability. In reality, of course, we do not known exactly neither student abilities nor problem parameters. We compute their estimates by an iterative bootstraping process:

1. initialization: for each problem i, set problem parameters as follows: $a_i = -1$, $b_i =$ mean time, $c_i = k$,
2. repeat until a selected convergence criterion is satisfied:
   (a) for each user j update the estimates of θ_j based on the current problem parameters,
   (b) for each problem i update the estimates of a_i, b_i, c_i based on the current ability estimates.

Although each of the steps computes maximum likelihood estimates (with respect to fixed input parameters), overall it is only approximation of the joint maximum likelihood. One of the reasons is that the input parameters in each step of iteration are only estimates and they differ in their confidence, e.g., for students which solved more problems we have better estimates of their ability. However, this aspects is not included in the described computation. This issue can be (pragmatically) addressed by using weighted least squares for estimating parameters a_i and b_i with weight for each student dependent on the number of solved problems.

4 Application and Evaluation

We apply the model in development of a web portal for tutored problem solving: a "Problem solving tutor". In this section we describe the system and provide evaluation of the model using the collected data.

4.1 Problem Solving Tutor

Problem solving tutor is a free web-based tutoring systems for practicing problem solving skills; the system is available at `tutor.fi.muni.cz`. The tutor contains large set of problems of different types. At the moment the system contains a math problem, two programming exercises, regular expressions, and 10 logic puzzles. For each problem type there are between 30 and 80 instances of different difficulty. The system is in active development and we are continuously adding new problems and collecting more data. Although the system was made public only recently, it is already used by several high schools and has more than 1 200 registered users who have spent more then 2000 hours solving more than 60 000 problems.

At the moment we focus solely on the "outer loop" of the tutor [17], i.e., recommending problem instances of the right difficulty. The inner loop (within a

selected problem) is currently not present – we just let the users solve a problem, either they finish or give up; i.e., rather than giving hints we give users different (simpler) problem to solve.

The following modules provide the core functionality of the system:

- *Problem simulators.* Simulators provide web interface for solving individual problems (puzzles).
- *Prediction module.* Based on the collected data it makes predictions about solving time for given user.
- *Recommendation module.* Based on predictions it recommends to a user an unsolved problem of suitable difficulty. Recommendations are based on the predicted times and on the session history (e.g., number of recent successes and failures).
- *Feedback module.* Based on the collected data it gives a user comparison with other users; particularly we provide immediate feedback after finishing problem solving (to support the flow phenomenon [4]).

Our focus at the moment is on the prediction module, which implements the model described above. The prediction module uses the iterative process for computing the parameter estimates. It would be computationally expensive to recompute all parameter estimates after each solved problem. Thus when user j solves problem instance i we update only parameters a_i, b_i, c_i, θ_j and only occasionally we run complete update of all parameters (i.e., full run of the iterative computation).

For every solved problem instance we store not only a final solving time, but also every performed move. In this way we collect extensive data about human problem solving. These data may be useful for further analysis of human problem solving behaviour and more detailed research into problem difficulty (in a similar way as in our previous research [6,12]).

4.2 Evaluation of Predictions

We have evaluated two approaches for estimating parameters: our implementation of the iterative estimation process (as described above) and estimation using a generic non-linear least squares method (using R software). The iterative computation converges very quickly (for practical use 3 iterations are enough) and both methods provide very similar results.

Student abilities θ are approximately normally distributed and the relation between ability and logarithm of time is nearly linear, see Fig. 6. These results support assumptions on which the model is based (see discussion in Section 2.3).

To evaluate a quality of predictions, we compare predictions based on the one parameter model with the baseline metric "mean time to solve a problem". Both prediction methods were trained using 90% of data and evaluated on the remaining 10% of data. Fig. 5 shows predictions and solving times for the Graphs and functions problem. Table 1 compares the results using the Spearman correlation coefficient. We have also evaluated other metrics like the Pearson correlation coefficient, root mean square error and mean absolute error, the results are similar.

Table 1. Evaluation of quality of predictions measured by Spearman correlation coefficients

Problem	Baseline	Model	Improvement (%)
Robot Karel	0.45	0.80	77.78
Nurikabe	0.42	0.73	73.81
Regular expressions	0.47	0.73	55.32
Graphs and functions	0.51	0.77	50.98
Slitherlink	0.64	0.84	31.25
Sokoban	0.67	0.80	19.40
Tents puzzle	0.58	0.67	15.52
Robot programming	0.62	0.71	14.52
Tilt maze	0.71	0.78	9.86
Region division	0.52	0.57	9.62
Rush Hour puzzle	0.78	0.84	7.69
Number maze	0.78	0.82	5.13

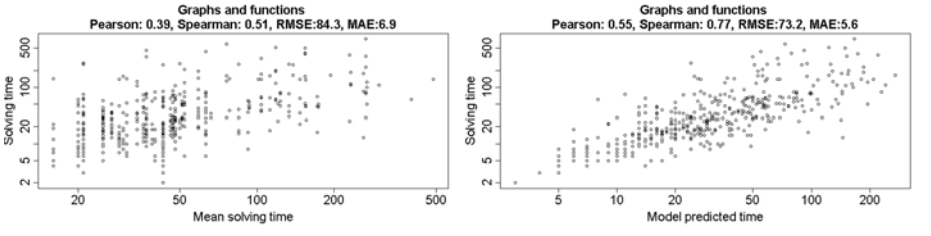


Fig. 5. Prediction versus real problem solving data: comparison of prediction based on mean time (left) and on the one parameter model (right)

As the table shows, the model leads to improvement of 5-80% in precision of the prediction. As expected, the improvement is modest in cases of simple puzzles which contain a "luck factor" (e.g., mazes), and it is more pronounced for problems, where problem solving skill plays significant role (pedagogical problems or more complex logic puzzles like Nurikabe or Slitherlink).

There is no significant difference between quality of predictions based on the one parameter model and the three parameter model. We suppose that the three parameter models needs more data to make a difference for predictions. Nevertheless, even with current data the three parameter model brings an additional insight – as we now illustrate on one of the examples.

4.3 Insight Gained from Parameter Values

Let us illustrate the insight gained from the values of three parameter model on the problem "Graphs and functions", which is described in introduction. Fig. 6 shows the collected data and values of model parameters for the three examples illustrated in Fig. 1. All three problems have similar basic difficulty (parameter b), but they differ in the other two parameters. The "Logarithm wings" problem

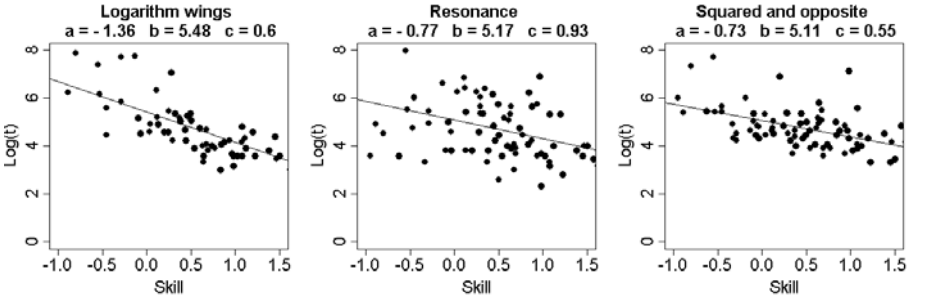


Fig. 6. "Graphs and functions" problem – three specific examples, for each of them we provide the collected data and values of parameters of the three parameter model

has small randomness and large discrimination; the "Resonance" problem has large randomness and small discrimination; and the "Squared and opposite" problem has small randomness and small discrimination.

These parameters provide a valuable insight, which can be potentially used for further improving intelligent tutoring systems. Problems with small discrimination and large randomness clearly depend more on luck than on ability and thus are probably not a very good pedagogical examples (so we may want to filter out such examples). At the beginning of the problem solving session (when we do not have a good estimate of student ability), we may prefer problems with small discrimination (so that we have higher confidence in solving time estimation), later we may prefer problems with higher discrimination (so that we select problems "tuned" for a particular student).

5 Conclusions and Future Work

We propose a novel variation on the item response theory where we do not focus on correctness of answers but on the time to solve a problem. Our model is given by a function which for a problem solving ability gives a probabilistic distribution of time to solve a problem. We provide a specific model with three parameters and discuss methods for parameter estimation. We evaluate the model and apply it in a problem solving tutor, which is already used in education.

This work lays foundations for future work in several directions. First, it would be useful to extend the model to deal with unfinished attempts (when a student spends some time trying to solve a problem and then abandons the problem, we do not include this information in our computation, although it can plausibly improve our parameter estimates). Second, the problem solving tutor can be extended by including an inner loop (hints for problem solving), and more sophisticated recommendations (e.g., using session history). Third, the collected data could be used to analyse causes of difficulty of particular problems (in a similar way as in our previous work [6,12]).

References

1. Anderson, J.R., Boyle, C.F., Reiser, B.J.: Intelligent tutoring systems. Science 228(4698), 456–462 (1985)
2. Baker, F.B.: The basics of item response theory. University of Wisconsin (2001)
3. Csikszentmihalyi, M.: Beyond boredom and anxiety. Jossey-Bass (1975)
4. Csikszentmihalyi, M.: Flow: The psychology of optimal experience. HarperPerennial, New York (1991)
5. De Ayala, R.J.: The theory and practice of item response theory. The Guilford Press (2008)
6. Jarušek, P., Pelánek, R.: What determines difficulty of transport puzzles? In: Proc. of Florida Artificial Intelligence Research Society Conference, FLAIRS (2011)
7. Kantor, P.B., Ricci, F., Rokach, L., Shapira, B.: Recommender systems handbook. Springer, Heidelberg (2010)
8. Koedinger, K.R., Anderson, J.R., Hadley, W.H., Mark, M.A.: Intelligent tutoring goes to school in the big city. International Journal of Artificial Intelligence in Education 8(1), 30–43 (1997)
9. Koedinger, K.R., Corbett, A.T., Ritter, S., Shapiro, L.: Carnegie Learning's Cognitive Tutor: Summary research results. White paper. Available from Carnegie Learning Inc., 1200 (2000)
10. Kotovsky, K., Hayes, J.R., Simon, H.A.: Why are some problems hard? Evidence from tower of Hanoi. Cognitive psychology 17(2), 248–294 (1985)
11. Kotovsky, K., Simon, H.A.: What Makes Some Problems Really Hard: Explorations in the Problem Space of Difficulty. Cognitive Psychology 22(2), 143–183 (1990)
12. Pelánek, R.: Difficulty rating of sudoku puzzles by a computational model. In: Proc. of Florida Artificial Intelligence Research Society Conference, FLAIRS (2011)
13. Pizlo, Z., Li, Z.: Solving combinatorial problems: The 15-puzzle. Memory and Cognition 33(6), 1069 (2005)
14. Simon, H.A., Newell, A.: Human problem solving. Prentice-Hall (1972)
15. Van der Linden, W.J.: A lognormal model for response times on test items. Journal of Educational and Behavioral Statistics 31(2), 181 (2006)
16. Van Der Linden, W.J.: Conceptual issues in response-time modeling. Journal of Educational Measurement 46(3), 247–272 (2009)
17. Vanlehn, K.: The behavior of tutoring systems. International Journal of Artificial Intelligence in Education 16(3), 227–265 (2006)

Generic Heuristic Approach to General Game Playing

Jacek Mańdziuk[1] and Maciej Świechowski[2]

[1] Faculty of Mathematics and Information Science,
Warsaw University of Technology, Warsaw, Poland
j.mandziuk@mini.pw.edu.pl
[2] Phd Studies at Systems Research Institute,
Polish Academy of Sciences, Warsaw, Poland
m.swiechowski@ibspan.waw.pl

Abstract. General Game Playing (GGP) is a specially designed environment for creating and testing competitive agents which can play variety of games. The fundamental motivation is to advance the development of various artificial intelligence methods operating together in a previously unknown environment. This approach extrapolates better on real world problems and follows artificial intelligence paradigms better than dedicated single-game optimized solutions. This paper presents a universal method of constructing the heuristic evaluation function for any game playable within the GGP framework. The algorithm embraces distinctive discovery of candidate features to be included in the evaluation function and learning their correlations with actions performed by the players and the game score. Our method integrates well with the UCT algorithm which is currently the state-of-the-art approach in GGP.

1 Introduction

Computer systems able to play a particular game such as chess or checkers have always been in the interest of Artificial Intelligence (AI). One of the most prominent examples is Deep Blue [1] - chess-playing machine which successfully challenged Garri Kasparov. Such programs, however, are equipped with game specific knowledge and heavily rely on computational power rather than intelligent behavior. General Game Playing (GGP) represents a new trend in AI focused on the ability of playing many different games previously unknown to the playing system. Given the game rules written in the so-called GDL (Game Description Language) [2] a playing agent takes various actions towards learning and mastering the game. This includes analysis of the game rules, application of various learning and searching mechanisms, logic-based reasoning methods, efficient knowledge representation and many other techniques [3,4,5]. Integration of all these elements formulates an interesting and challenging research task. Before playing a game an agent is a *Tabula Rasa* - no game specific features should be assumed *a priori*. GGP took its name from the competition proposed by Stanford Logic Group in 2005. It is held annually at AAAI (or IJCAI in

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 649–660, 2012.

2011) conferences. Playing environment contains central unit called Gamemaster and remote playing agents called Game Players. Game Players communicate via http protocol with the Gamemaster only whose role is to provide players with the rules in the GDL, running the game, sending control messages and receiving responses. Gamemaster also includes its own GDL reasoning mechanism in order to validate legality of the players' moves and updates the state. If an agent responded with an illegal move, a random move would be selected for them. Each agent is then notified about moves performed by other players. The contest features two timers: a move clock and a start clock. The first one counts time available for notifying the Gamemaster about selected action and the latter represents time for preparation before the actual start of the game. Hence, the start clock defines room for application of various pre-game learning strategies.

1.1 The Class of Considered Games

Any game which is finite, deterministic and synchronous can be played within GGP framework. The term finite should be understood as finite number of players and available actions (moves) in any game state and finite number of states. One distinguished state is marked as initial and at least one as terminal. Each terminal state has goal values defined for each player. Goal values range from 0 to 100. Due to deterministic nature of the game a state can change only as a result of performed move and there is no randomness. Players perform moves simultaneously (synchronously) during the update phase, but turn-based games can be easily simulated with the use of no-operation moves. In this scenario, for all players but one players the no-operation move is the only legal move available for them in the current state.

1.2 Game Description Language

GDL is a formal first-order logic language with the structure strictly following Datalog [2], which in turn is a subset of Prolog. Terms used in game descriptions compose sentences that are true in particular states. There are a few distinguished keywords which cannot be redefined. As an example a partial listing of Tic-Tac-Toe game written in GDL is presented below.

```
(role xplayer) (role oplayer)
(init (cell 1 1 b)) (init (cell 1 2 b))
   ***
(init (cell 3 3 b))
(init (control xplayer))
(<= (next (cell ?m ?n x))
    (does xplayer (mark ?m ?n))
    (true (cell ?m ?n b)))
   ***
(<= (next (control oplayer))   (true (control xplayer)))
(<= (row ?m ?x)
    (true (cell ?m 1 ?x))
```

```
    (true (cell ?m 2 ?x))
    (true (cell ?m 3 ?x)))
   ***
(<= (legal ?w (mark ?x ?y))
    (true (cell ?x ?y b))
    (true (control ?w)))
(<= (legal xplayer noop)   (true (control oplayer)))
(<= (goal xplayer 100)    (line x))
(<= (goal xplayer 50)
    (not (line x))
    (not (line o))
    (not open))
(<= (goal xplayer 0)   (line o))
   ***
(<= terminal   (line x)) (<= terminal   (line o)) (<= terminal (not
open))
```

[A subset of Tic-Tac-Toe game definition downloaded from Dresden GGP Server [6].]
A complete set of keywords consists of the following elements: *role, init, true, does, next, legal, goal, terminal, distinct*. They are used to define the initial state, legal moves, state update procedure as well as game terminal states and goals accomplishment. A more detailed description of all keywords can be found in [2]. There are also logical operators available in GDL, such as *not, or,* and $<=$. A special symbol ? is used to make an argument a variable - in this case a set of symbols satisfying the truth condition is to be calculated. For example: relation (cell ?m 1 ?x) has two variable arguments (?m ?x) and one constant (1). Negation and recursion, in restricted form, are both part of the language too.

2 State-of-the-Art

GGP annual competition provides an environment for testing the strength of game playing algorithms. Last years were dominated by two winning approaches: CadiaPlayer (2007, 2008) presented in [7] and Ary (2009, 2010) described in [8]. Both agents rely on performing Monte Carlo simulations (MCS) aimed at learning the game and incrementally building the game tree. This solution was inspired by Go playing agents [9]. MCS perform random play from the current state to the terminal state. The goal value is then obtained and stored in the current node of the tree. Storing the entire tree in memory using all visited nodes on simulation paths would quickly exceed the available memory. Therefore, only one node, representing the first action, is added after a single simulation [7]. The most popular variant of MCS is known as Upper Confidence Bounds Applied for Trees (UCT) method [9] which efficiently keeps balance between exploration and exploitation. As the name suggests, the UCT method is a generalization of the Upper Confidence Bounds (UCB) [10] which can be used, for example, to learn the payoff distribution of slot machines in a casino. The goal of UCT is to perform MCS as wisely as possible taking advantage of the knowledge acquired so far. In each node the algorithm checks if all possible moves in the associated

position have already been tried at least once during simulation (therefore they possess initial MC estimations). If not, one of the unvisited child nodes is chosen at random. Otherwise (i.e. in case all successors of the current node have been visited at least once), the move a^* is chosen according to the following rule:

$$a^* = arg \max_{a \in A(s)} \left\{ Q(s,a) + C\sqrt{\frac{ln\,[N(s)]}{N(s,a))}} \right\} \quad (1)$$

where a - is an action; s - the current state; $A(s)$ - a set of actions available in state s; $Q(s,a)$ - an evaluation of performing action a in state s; $N(s)$ - a number of previous visits of state s; $N(s,a)$ - the number of times an action has been sampled in state s; C - a coefficient defining a degree to which the bonus (second component) should be considered. UCT-based players, such as the above-mentioned Ary and Cadia build a game tree gradually. Each node stores the average payoff, obtained by those MCS, in which it was visited on the path of play. Simulations are not terminated when the start time elapses but continue through the entire GGP episode. During the actual game, if a player chooses an action stored in a node, that node becomes a new tree root (i.e. all higher branches are deleted, because they are not needed anymore). A new simulation always runs from the current game state. Most of the UCT based players share the basic idea described above but differ by employing specific search control mechanism to optimize tree exploration [11].

3 Automatic Construction of the Evaluation Function

One of the potential enhancements of purely simulation-based UCT implementation is combining it with the use of some kind of evaluation function. Due to wide variety of GGP games it is hardly (if at all) possible to design such a generally-applicable function *a priori* and only tune its coefficients for a particular game. Despite the above difficulties, several researchers have explored this possibility and some heuristic approaches to General Game Playing used predefined candidates for the evaluation procedure. ClunePlayer [12] considered mobility, payoff and termination stability. Fluxplayer [13], harnessed predefined syntactic templates for common game features like board definition or successor relation. Fluxplayer's heuristic construction mechanism is an extension to the idea derived earlier in [15]. Another noteworthy approach, adopted by OGRE player [16], focuses mainly on board games. It uses the so-called evaluators. The game structure evaluators are *distance-initial*, *distance-to-target*, *count-pieces* and *occupied-columns* whereas game definition evaluators are *count-moves*, *depth*, *exact*, *pattern* and *purse* [16]. OGRE came 4-th out of 12 entrants in 2006 competition winning 34% of the matches [17].

Generally speaking, all the above-mentioned methods were geared towards standard two players board games and, in random environment, they tend to lose againts UCT-based players. Our approach constructs features that are completely independent of particular game definition. The key difference is that no

predefined templates are present. Its underpinning idea is related to identification of meaningful numbers from the symbol representation. Before going into details let us introduce database-like vocabulary used to describe the elements of GDL.

- A ***row*** is a complete term in GDL that describes the game fact in a particular state. By a fact we mean a statement which is true. A row consists of a name of the fact and its arguments (called ***symbols***).
- A ***table*** is a name of the row; in our example it is *cell*. ***TableRows*** are all rows sharing a common name;
- A ***column*** is a set of symbols at a fixed position in rows' argument lists.

The proposed algorithm for automatic construction of the evaluation function for a given GGP game consists of three phases: selection, construction and play.

3.1 Selection Phase

The aim of this phase is to select the candidate features for heuristic function. The main idea is to track cardinality of three kinds of objects:

- For a given table count its rows (***TableRows***);
- For a given table, column index and symbol count the symbol occurrences in the corresponding column in the table (***ColumnSymbols***);
- For a given table, column index and symbol extract the set of rows with the matching symbol in the corresponding column in a given table ***(SymbolRows)***.

The essential part of the algorithm is the way the features are counted. This issue is described in detail below.

Selection phase - step 1 - parallel simulations. In the first selection phase TableRows and ColumnSymbols are found. Not all of them are selected but only those with **occurrence count varying** during the game in a manner which depends on the performed moves. It means that, if for a current game state all legal moves produce pairwise equal changes to the object's quantity, then such occurrence is neglected by the algorithm. The motivation behind this constraint is to discard all features a player has no impact on (such as counters, timers, control, board cells count etc.) and focus on real move consequences. In order to perform the selection, N simulations with random move making are launched in parallel. Parameter N can be tuned depending on how much time is available. Tests show that 3-4 parallel simulations are usually sufficient. This phase terminates when there is only one or none unfinished simulations left. At each simulation step TableRows and ColumnSymbols are counted independently for each simulation and their counters are tested against each other. If a difference occurs, an object (a table with all its TableRows or ColumnSymbol) is marked as changing and excluded from further tests. It is important to note that a difference is computed only between the same steps of each of (different) simulations. No difference is computed between consecutive steps. Objects marked as changing are stored for further use.

Selection phase - step 2 - extracting symbols. In the second step of selection phase, only one complete random simulation is performed. Let (s_1, s_2, s_3, ..., s_{n-1}, s_n) represent consecutive states present during the simulated game. After reaching each state s_i two additional random moves are simulated that lead to hypothetical states s_{ij} and s_{ik}. The difference between the two new states s_{ij} and s_{ik} is analyzed from the heuristic selection viewpoint. The main simulation continues as if it was not affected by the two moves and SymbolRows sets are constructed for each symbol. The general idea behind this part of the algorithm is to filter symbols which express the most important features within the relation. The most important features are usually dynamic and the rest of symbols which they appear with, represent their properties which vary from state to state. Therefore features supposed to play important role will change their set of properties often. The following measure of symbols' variation was used:

$$val = 1 - \frac{2 * |A_{ij} \cap A_{ik}|}{|A_{ij}| + |A_{ik}|} \tag{2}$$

where A_{ij} and A_{ik} denote SymbolRows for a particular symbol in states s_{ij} and s_{ik}. Formula (2) is used to calculate variation of each symbol during the selection step. The most varying symbol (with the highest computed value) at each step is marked as changing and becomes a candidate for further heuristic weighting. If a selected symbol has already been marked before, this new selection is ignored.

3.2 Construction of a Heuristic Function

During the selection phase some objects marked as changing are captured. These can either be TableRows, ColumnSymbols or SymbolRows. The occurrence count is correlated with the actions selected by a player during the game. The purpose of the construction phase is to approximate the correlation factor by assigning weights to the counters. Here come MCS with preferable UCT enhancement which are run until the time is up. These MCS are used to assign weights to discovered elements of the evaluation function. The following pseudocode describes the weight-learning phase:

```
ConstructHeuristic(TimeLimit,Player)
While(currentTime < TimeLimit)
    Start a new simulation S
    While S not finished
        SavedStates->Push(S->State)
        S->Advance
    If IsSuccess(S->State,Player)
        CountOccurences(SavedStates)
      CountAverages()
      AddAverages(WinAverageList);
    Else If IsFail(S->State,Player)
    CountOccurences(SavedStates)
      CountAverages()
      AddAverages(LossAverageList);
For each heuristic object:
```

```
    Count winAverage
    Count lossAverage
  weight = C*(winAverage - lossAverage)/MaxValue
End
```

[A pseudocode of the weights-learning phase.]

Although the detailed algorithm appears to be rather complicated, its underlying idea is quite simple. During simulations the counting procedure is performed and the game-average result is computed. This value is put into collection of either 'win values' or 'loss values' depending on the game result assigned to the player for whom the heuristic is being defined. The distinction between a won and lost game can be defined in various ways. In our approach it was:

```
AverageResult =
(MaxResult - MinResult)/2 If(Result > AverageResult)
    Win = true
Else If(Result < AverageResult)
    Loss = true;
Else //no action
    Win = Loss = false
```

[A pseudocode for determining the win or loss. MaxResult and MinResult come from the GDL definition.]

In the final step of this phase, game-average values from win and loss collections are transformed into single averages for won and lost games independently. For each counter the maximum occurrence ever (MaxValue) is monitored and used for the normalization purpose (see an example below).

```
//Data structures after a completed simulation:
CurrentGameValues = [6,4,2,0] //symbol occurrences
WinAverageValues = [4.5, 4.5]
LossAverageValues = [3,2]
MaxValue = 6
//Computing current average
CurrentAverage = (6+4+2+0)/4 = 3
//Let assume the game was won
WinAverageValues = [4.5, 4.5, 3] //3 is added
//Computing single average values for won and lost games
WinAverage = (4.5+4.5+3)/3 = 4
LossAverage = (3+2)/2 = 2.5
```

[Illustration of how the average values are computed.]

The heuristic value for each object is computed according to the following formula:

$$weight = C * \frac{WinAverage - LossAverage}{MaxValue} \tag{3}$$

where C is a constant parameter for each of the three types of heuristic elements. In the current implementation of the system we use $C = 1.0$ for both TableRows

and ColumnSymbols and $C = 0.2$ for SymbolRows. Concrete rows, counted for symbols, have lesser weights in order to force them to be used if no legal action positively changes TableRows or ColumnSymbols. The final evaluation function is a linear combination of the numbers of occurrences of the elements multiplied by the computed weights.

3.3 The Use of the Heuristic Function

The evaluation method constructed as described in sections 3.1 - 3.2 takes game state as an input, performs weights calculation according to (3) and returns a single floating point value. Such automatically constructed heuristic can be taken advantage of in several ways by the playing agent. It can be used for

(1) evaluation of each legal move in order to choose the heuristically best one;
(2) incremental building of a min-max inspired game tree (in that case a distinct heuristic function is maintained for each player);
(3) sorting unplayed actions in the classic MC + UCT solution (evaluation function helps to determine which branches should be tried first);
(4) to replace the whole or part of the MC phase with certain probability in the classic MC + UCT approach [14].

In the experimental evaluation of the proposed method our focus was on facets (2). We decided to fully utilize start clock on heuristic function construction and move clock for min-max tree search. Such distribution was chosen because it is natural and easy to maintain. However, for some games a different balance might be more beneficial.

4 Empirical Results

An agent using the heuristic function was tested against reference UCT solution based on CadiaPlayer description [7] in several games downloaded from the Dresden General Game Playing Server site [6]. The main criterion for choosing games was to focus on most widely recognized games of various rules and complexity. The games included bomberman, breakthrough, checkers, chess, connnectfour, farmers, othello, pacman, tic-tac-toe, sheep and wolf and wallmaze. Each of these 11 games was played 80 times with four different pairs of clocks settings (start clock, move clock), i.e. (16T,2T), (32T,4T), (64T,8T) and (128T,16T) where T is a game-specific parameter proportional to the average time of one random simulation from the beginning to the end for particular game. The exact value depends on game complexity. In each case, in half of the games the Heuristic Player (our algorithm) was making the first move and it the remaining half of them the UCT was the initial player.

In majority of tested games interesting features were selected for the heuristics. For example high value is always assigned in chess to the TableRow **check** making the player perform checking the opponent whenever possible. Our player achieves better win ratio in almost all games for the shortest of tested times,

Table 1. Percentage results between the Heuristic Player and UCT for short times. The interpretation is the following: Heuristic Player win ratio - UCT win ratio (the remaining games are ties). The results in favor for the Heuristic Player are bolded.

Game	**HP vs UCT**. Clocks = [16T,2T]	**HP vs UCT.** Clocks = [32T,4T]
Bomberman	6% - 94%	11% - 86%
Breakthrough	50% - 50%	50% - 50%
Checkers	**64%** - 22%	**66%** - 20%
Chess	**14%** - 0%	**35%** - 10%
Connectfour	**48%** - 40%	**45%** - 44%
Farmers	36% - 64%	32% - 68%
Othello	**50%** - 25%	29% - 49%
Pacman	**78%** - 22%	**75%** - 25%
Tic-Tac-Toe	**55%** - 25%	30% - 33%
Sheep and Wolf	**89%** - 11%	**76%** - 24%
Wallmaze	**3%** - 0%	**6%** - 0%

with bomberman and farmers being the only exceptions. The results prove that the evaluation function constructed during the preparation time has a positive impact on playing quality. With the increase of time the UCT becomes a stronger opponent. It is mainly due to a greater impact of MCS performed during the move clock. With extreme time limits, most parts of the tree constructed by UCT approach will have an exact goal values fetched directly from terminal states, whereas min-max algorithm requires a full tree expansion in order to fetch at least one real goal value. This is a key difference between the methods in terms of a tree search. For the longest time settings the heuristic player remained superior in five games. Chess and checkers are games which are in favor for our method in a most significant way, whereas bomberman and farmers are games at which the UCT is undeniably better. Below we present two evaluation functions obtained for chess and bomberman, respectively.

Table 2. Percentage results between the Heuristic Player and UCT for longer times. See description of Table 1.

Game	**HP vs UCT**. Clocks = [64T,8T]	**HP vs UCT.** Clocks = [128T,16T]
Bomberman	21% - 70%	14% - 77%
Breakthrough	48% - 52%	37% - 63%
Checkers	**84%** - 10%	**74%** - 16%
Chess	**45%** - 9%	**23%** - 4%
Connectfour	45% - 46%	41% - 51%
Farmers	14% - 86%	11% - 89%
Othello	38% - 49%	30% - 54%
Pacman	**55%** - 45%	**51%** - 49%
Tic-Tac-Toe	44% - 46%	24% - 58%
Sheep and Wolf	**70%** - 30%	**58%** - 42%
Wallmaze	**29%** - 25%	**34%** - 30%

Chess. The evaluation function primarily forces to check the opponent whenever possible and rewards having a greater number of particular pieces. Piece types have various levels of importance assigned. Moreover, board positions where a threat to adversary king's initial location (coordinates) is applicable are slightly favored. The most varying symbols are discovered as *wp, wn, wb, wq, wr, bp, bn, bb, bq, br* for *cell*. These symbols represent chess pieces. For example *wn* stands for white knight and *bq* stands for black queen.

1. **TableRows:**
   a (check, 0.8); (pawn_moved_two, -0.006); (piece_has_moved, -0.25).
   These symbols represent one-time actions that can occur after a move.
2. **ColumnSymbols:**
   a For cell at column 2: (b,-0.008); (bb, -0.08); (bn, -0.04); (bp, 0.1); (bq; -0.21); (br; -0.12); (wb, 0.29); (wn, 0.16); (wp, 0.11); (wq, 0.52); (wr; 0).
   Pieces counts (known as strength of the material) are captured here.
   b For check at column 0: (black, 0.36); (white, -0.43).
   The white player is advised to check the black player.
   c More values are discovered for all symbols in check, pawn_moved_two, piece_has_moved, but with little impact.
3. **SymbolRows:**
   a All rows with varying symbols that occurred during simulations, e.g. (cell a 4 wq, 0.012). Values are within the range (-0.02, 0.02).
   Particular board cells with concrete pieces are evaluated here.

Bomberman. This is an example of a game for which usually only one varying symbol is discovered. It is 1 at the first index of *blockedeast* table. It turned out that one random simulation in step 2 of the selection phase (3.1) is insufficient since, if acting randomly, a player has 50% chance of dying because of its own bomb in the first turn of the game. The only legal action is to place a bomb or move in unblocked direction. The probability of losing in a few turns drastically increases. UCT is capable of finding the safe path if given enough time.

1. **TableRows:** (location, -0.1).
2. **ColumnSymbols:**
   a For table location, column 0: (bomb0, -0.5); (bomb1, -0.11); (bomb2, -0.05); (bomb3, 0).
   b For location, columns 1 and 2: not meaningful values.
3. **SymbolRows:** (blockedeast 1 2, 0) (blockedeast 1 7, 0).

The term *location* which appeared above describes rows with the following structure *(location ?object ?x ?y)*. Its column 0 (?object) contains symbols representing object located at (?x,?y) coordinates (columns 1,2). Possible objects are bomberman, bomberwoman, bomb0, bomb1, bomb2 and bomb3. The term blockedeast means, if present, that east direction is not available at particular coordinates. The game description includes also north direction which was

not selected by the algorithm. The goal of the game is to avoid bombs (of any players) and make the opponent die by a bomb.

5 Conclusion

A novel approach to building a heuristic evaluation function for General Game Playing has been presented. The proposed algorithm clearly outperforms UCT in four out of eleven games while losing visibly in three games. The introduced heuristic evaluation is constructed in a fully automatic way. Instead of using predefined categories it counts occurrences of carefully filtered elements of three different kinds. GDL description is lexical by nature and there are no ready-to-use numbers encoded (even mathematical operators must be explicitly defined for all possible arguments of lexical symbols). Our solution not only extracts numbers but also assesses their usefulness. Only elements whose values' variability is caused by a player's move are considered which is an essential idea of the algorithm. Their usefulness is further remodeled by their correlation with a game score. Experiments show that the method is well suited for games in which some 'objects' are created, destroyed or moved. Objects can be of any type like money, pieces, buildings, obstacles etc. The evaluation function can be inserted at several stages of GGP scenario. It can be used to guide MC or UCT search, which vastly improves the quality of play for games it normally performs bad at and moderately decreases the quality of play for games it has advantage in. The integration may even proceed a step further - the agent may discover, during the learning phase, whether using plain heuristic approach or guided UCT gives better results. Better strategy may by dynamically chosen for the actual play. The other way to improve the method would be to incorporate it into a complex agent featuring various heuristic functions and equipped with a feedback-based learning mechanism to choose the best suited evaluation for a currently playing game.

The main weak point of the proposed solution is that quality of the computed function greatly depends on the numbers of simulations which fall into won and lost categories. If a game lasts for a very long time or a tie is the typical result, the evaluation function may be inaccurate. Such games are also difficult to master by UCT approaches because most of the results propagated in a tree are ties. Our current work concentrates on extension of the proposed method towards automatic discovery of correlations among particular game elements (game aspects) which would allow capturing their dynamic (changing in time) mutual dependencies. This issue, in its general form, is highly challenging. To the authors' knowledge no universal solution for automatic discovery of such correlations exists, even in well-researched classical board games domain (chess, checkers, Go, Othello, etc.).

Acknowledgments. Maciej Świechowski would like to thank Foundation for Polish Science under International PhD Projects in Intelligent Computing. Project financed from The European Union within the Innovative Economy Operational Programme 2007-2013 and European Regional Development Fund.

References

1. Newborn, M.: Kasparov versus Deep Blue: Computer Chess Comes of Age. Springer, Heidelberg (1997)
2. Love, N., Hinrichs, T., Haley, D., Schkufza, E., Genesereth, M.: General Game Playing: Game Description Language Specification. Technical Report LG-2006-01 (2006), http://games.stanford.edu
3. Genesereth, M., Love, N.: General Game Playing: Overview of the AAAI competition. AI Magazine 26, 62–72 (2005)
4. Walędzik, K., Mańdziuk, J.: CI in General Game Playing - To Date Achievements and Perspectives. In: Rutkowski, L., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2010, Part II. LNCS, vol. 6114, pp. 667–674. Springer, Heidelberg (2010)
5. Mańdziuk, J.: Knowledge-Free and Learning-Based Methods in Intelligent Game Playing, ch. 14.5, vol. 276. Springer, Heidelberg (2010)
6. Dresden GGP Server, http://euklid.inf.tu-dresden.de:8180/ggpserver
7. Bjornsson, Y., Finnsson, H.: CadiaPlayer: A Simulation-Based General Game Player. IEEE Transactions on Computational Intelligence and AI in Games 1(1), 4–15 (2009)
8. Mėhat, J., Cazenave, T.: Ary, a General Game Playing Program, Board Games Studies Colloqium, Paris (2010)
9. Gelly, S., Wang, Y.: Exploration and Exploitation in Go: UCT for Monte-Carlo Go. In: 20th Annual Conference on Neural Information Processing Systems, NIPS (2006)
10. Auer, P.: Using upper confidence bounds for online learning. In: FOCS 2000, Proceedings of the 41st Annual Symposium on Foundations of Computer Science (2000)
11. Bjornsson, Y., Finnsson, H.: Simulation Control in General Game Playing Agents. In: Proc. IJCAI 2009 Workshop on General Game Playing, GIGA 2009 (2009)
12. Clune, J.: Heuristic evaluation functions for general game playing. In: Proc. AAAI Nat. Conf. on Artificial Intelligence, pp. 1134–1139. AAAI Press, Vancouver (2007)
13. Schiffel, S., Thielscher, M.: Fluxplayer: A successful general game player. In: Proceedings of the AAAI National Conference on Artificial Intelligence, pp. 1191–1196. AAAI Press, Vancouver (2007)
14. Walędzik, K., Mańdziuk, J.: Multigame Playing by Means of UCT Enhanced with Automatically Generated Evaluation Functions. In: Schmidhuber, J., Thórisson, K.R., Looks, M. (eds.) AGI 2011. LNCS (LNAI), vol. 6830, pp. 327–332. Springer, Heidelberg (2011)
15. Kuhlman, G., Dresner, K., Stone, P.: Automatic Heuristic Construction in a Complete General Game Player. In: Proceedings of the Twenty-First National Conference on Artificial Intelligence, pp. 1457–1462 (2006)
16. Kaiser, D.: Automatic Feature Extraction for Autonomous General Game Playing Agents. In: Proceedings of the Sixth Intl. Joint Conf. on Autonomous Agents and Multiagent Systems (2007)
17. Love, N.: 2006 General Game Playing Competition Results, http://euklid.inf.tu-dresden.de:8180/ggpserver (accessed 2006)

The SiMoL Modeling Language for Simulation and (Re-)Configuration

Iulia Nica and Franz Wotawa

Technische Universität Graz, Institute for Software Technology,
Inffeldgasse 16b/2, A-8010 Graz, Austria
{inica,wotawa}@ist.tugraz.at

Abstract. From automotive and up to telecommunication industry, configuration and simulation are used for solving complex problems connected to the ever growing number of components, which have to work together. To assist these needs, many tools are nowadays available. Modeling languages like Matlab/Simulink or Modelica are often used to model the dependencies between the components of physical systems. However these are less suitable for the area of knowledge-based systems. In this paper, we present a modeling language, which combines the two different directions. SiMoL is an object-oriented language that allows representing systems comprising basic and hierarchical components. We state the syntax and the semantics of the language, referring also to the implementation of SiMoL, which is based on the MINION constraint solver. Furthermore, we discuss how the obtained model can be used for simulation and re-configuration.

1 Introduction

The adaptation of technical systems after deployment to ensure the desired system's functionality over time is an important task and can never be avoided. Reasons for adaptation are necessary corrections due to faults in system parts, changes in user requirements, or changes of technology among others. All activities necessary for increasing the lifetime of a system and retaining its usefulness are summarized under the general term maintenance. When considering the overall cost of systems during the whole lifetime, maintenance accounts for more than 50 percent. Any support provided for maintenance potentially reduces costs or provides improved results while retaining costs at the same level.

In our research, we focus on system changes due to changes in requirements. For example, consider a cellular network where the base stations are initially configured to ensure current and future needs to some extent. Due to changes in the environment, i.e., new apartment buildings constructed in reach of the base station or an increased use of cellular networks for data communication, the base station or even the local topology of the network has to be adapted. This adaption can more or less be classified as a re-configuration problem where the current system's structure, behavior, and the new requirements are given as input. Changes in the structure and behavior of the system in

M. Bieliková et al. (Eds.): SOFSEM 2012, LNCS 7147, pp. 661–672, 2012.

order to cope with the changes in the requirements are a solution of the re-configuration problem.

In order to provide a method for computing solutions for a given re-configuration problem we need to state the problem in a formal way. Therefore, we require a modeling language for stating systems comprising components and their relationships. In principle, formal languages like first order logic or constraint languages would be sufficient for this purpose. But using such languages usually is not easy and prevents systems based on such languages to be used in practice. Hence, there is a strong need for easy to learn and use modeling languages that are expressive enough to state configuration problems. The SiMoL language we introduce in this paper serves this purpose. The language itself is from a syntactical point of view close to Java. The idea behind SiMoL is to provide a language that can be used for (restricted) simulation and configuration at the same time.

SiMoL is an object-oriented language with multiple inheritance and allows for stating constraints between variables. Beside the basic data types like integer and boolean, SiMoL makes use of component instances. All component instances are statically declared. In this paper we discuss the syntax and the semantics of SiMoL and show how the language can be used for re-configuration purposes.

2 Related Research

Over time, the AI community has developed a large variety of configuration tools that fitted the different necessities and goals in each practical area, thus creating a strong foundation for newcomers.

As preamble to our approach, we will shortly recall a couple of configuration systems, which illustrate the main approaches in the field of knowledge-based configuration: from rule- or structure-based configuration, to constraint-, resource- and case-based methods.

In the early 80s, Digital Equipment Corporation was already using the R1/XCON program in the selling process, by automatically selecting the computer system components based on the customer's requirements. [1] implemented XCON as a production-rule-based system, containing in 1989 about 11.500 rules, specified in OPS5 language [2]. But the weak structure offered by rules leaded to problems in managing the control flow of the configuration task. Due to its architecture, XCON offered no user interaction while solving the configuration task.

Another historical system is SICONFEX [3], which had a similar using purpose as XCON, but offered a completely different solution. The input data was represented by hardware components, software specifications and intended functionality. In this case, the system used many techniques in order to get a better structuring of the domain knowledge: rules, inheritance mechanisms, domain procedures, conceptual hierarchies, schemes for describing objects and LISP Code.

Other configuration systems include ConBaCon (Constraint-Based Configuration) [4] and CAWICOMS (Customer-Adaptive Web Interface for the Configuration Of products and services with Multiple Suppliers) [5]. ConBaCon treats the special case of re-configuration, using the conditional propagation of constraint networks and has its

own input language - ConBaConL. In [4], the authors present ConBAConL, a ”largely declarative specification language”, by means of which one can specify the object hierarchy, the context-independent constraints and the context constraints. Furthermore, the constraints are divided into Simple Constraints, Compositional Constraints and Conditional Constraints. Although successfully integrated in industry, the performance problems were observed in case of large products/systems, as a result of the large number of generated constraint variables and associated CE (consistency-ensuring) constraints within the solution. In [6], the author tackles these issues by clustering the ConBaCon model.

The scope of the CAWICOMS project was the development of a Business-To-Business framework for distributed products and services configurations. In [5], an application scenario for semantic Web services is presented, choosing as example the domain of telecommunication services. In order to define a common language for representing the properties of configurable products and services, the authors use a hierarchical approach of related ontologies. By means of the DAML+OIL language, a flexible product ontology for complex, customizable products can be modeled, the domain knowledge being consistently represented in XML. For the configuration task, ILOGs domain-independent and Java-based JConfigurator was adopted. JConfigurator implements Generative Constraints Satisfaction for solving complex configuration problems. Actually, the configuration task is executed on a distributed architecture, i.e., each involved configurator has only a partial view on the product model. The communication between the configurators occurs by means of XML-based SOAP messaging and Web Services.

LAVA is another successful automated configurator [7], used in the complex domain of telephone switching systems. It makes use of generative constraints and is the successor of COCOS [8], a knowledge-based, domain independent configuration tool. The modeling language is ConTalk, an enhanced version of LCON that follows the Smalltalk notation. A ConTalk constraint is a statement which describes a relationship between components ports or between the attributes values. During configuration, the inheritance hierarchy of component types is exploited and each time a component is generated, a new constraint object is instantiated for that component and propagated to all its related components, ports and attributes.

A powerful configuration system that combines constraint programming(CP) with a description logic(DL) is the ILOG (J)Configurator [9]. The combined CP-DL language, in which the configuration problem is formulated provides, on the one hand, the constraints, needed in the decision process, and on the other hand, the constructs of the description logic, able to deal with unknown universes. When solving the problem, the constructs of description logic, which are well-suited to model the configuration specific taxonomic and partonomic relations, are mapped on constraints and thus the wide range of constraint solving algorithms may be used.

Products like COSMOS [10] and KIKon [11] used the resource-based configuration approach, by seeing the components as resources and splitting the system in subfunctionalities (COSMOS) or sealing the obtained configurations in order to store them for future usage (KIKon). Among currently available products on the market, we

mention well-known systems like: Tacton Configurator[1] [12] (uses logic programming, object-oriented principles, SICSTUS Prolog [13] and its object-oriented extension SICSTUS Objects), SAP Product Configurators[2] (SAP ERP Variant Configurator -since 1994, provides high-level configuration in SAP ERP systems and SAP CRM IPC (Internet Pricing and Configuration)- since 2000, has a standalone configuration engine; both support constraints and procedural dependencies and use a Truth Maintenance System (TMS) as problem solving module [14]) or camos Configurator[3] [10] (the components are represented in a class tree, during the configuration process, the user can detect the inconsistencies - if they appear).

The other field of interest for our research has been the modeling languages currently used for simulation of technical systems. Matlab/Simulink[4] and Modelica[5] are the most famous ones in the area of dynamic systems modeling and simulation. When working with Simulink, the user is capable of modeling the desired system in the graphical interface, based on the large library of standard components (called blocks). As part of the standard library, Simulink includes a block intended to allow users to develop and implement their own custom routines: the S-Function block, which can include creation of new algorithms not directly supported by Simulink and reuse of existing code [15]. Also making use of predefined model building blocks, Modelica, on the other side, is an equation-based object-oriented language with multi-domain modeling capability. Although both of them are complex languages, capable of modeling a great variety of components, neither Simulink or Modelica can be used for re-configuration purposes, as the description of the configuration specific knowledge is, due to technical reasons, almost impossible. For instance, in Modelica, one cannot directly represent any fields (data variables associated with a class and its instances), if they have an unknown value. And this is inevitable in the implementation of non-monotonic reasoning, which is required in solving the configuration problem. In Matlab/Simulink on the other side, the differential equations are not even direct representable.

Throughout the rest of this paper, we present our modeling language - SiMoL. SiMoL can be applied in both simulation and re-configuration domains, using the powerful mechanism of constraint solving and hence being highly scalable for complex simulation and re-configuration tasks.

3 An Example

In this paper we make use of the following small example to discuss SiMoL, as well as re-configuration using SiMoL for modeling systems. Figure 1 depicts a small system comprising 4 components, i.e., a power supply (PS), an acceleration sensor (AS), a GPS sensor (GPS), and a communication device (CD). The communication device is used for sending the measured sensor information to a server. The power supply is

[1] http://www.tacton.com/en/
[2] http://www.sap.com/
[3] http://www.camos.de/
[4] www.mathworks.com
[5] www.modelica.com

for providing electricity to the connected components. All these components have a behavior and provide functionality.

For the purpose of specifying functionality we introduce a function fct that maps a component to a set of attributes, which indicate a certain functionality. For our example, we introduce the attributes **ad**, **gps**, **comm** to state the acceleration sensor functionality, the gps functionality, and the ability for communication respectively.

$$fct(AS) = \{\mathbf{ad}\}$$
$$fct(GPS) = \{\mathbf{gps}\}$$
$$fct(CD) = \{\mathbf{comm}\}$$

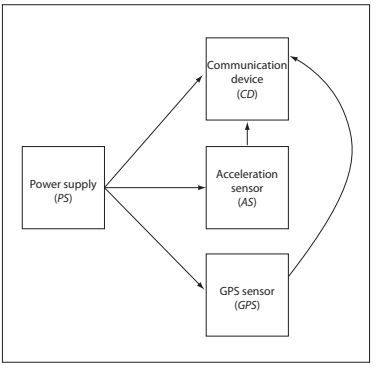


Fig. 1. A small sensor systems

We now specify additional constraints of the system. The following constraint formally represents the requirement that the power provided by PS must be larger or at least equivalent to the sum of the power consumption of the other components:

$$power(PS) \geq power(AS) + power(GPS) + power(CD)$$

Moreover, we state that the device has to provide at least **ad**, **gps**, **comm** functionality.

$$fct(AS) \cup fct(GPS) \cup fct(CD) \supseteq \{\mathbf{ad}, \mathbf{gps}, \mathbf{comm}\}$$

Finally, we have the requirement that the sum of the cost of each part of the device is not allowed to exceed a certain pre-defined maximum cost.

$$cost(PS) + cost(AS) + cost(GPS) + cost(CD) \leq max\_cost$$

In configuration we are interested in providing specific implementations of the components PS, AS, GPS, and CD such that all requirements are fulfilled and no constraint is violated. Hence, what we do now for our running example, is to introduce

Table 1. The component instances for our small sensor system

Generic Component	**Instance 1**	**Instance 2**
PS	$PS_1 : costs(PS_1) = 10$, $power(PS_1) = 10$	$PS_2 : costs(PS_2) = 20$, $power(PS_2) = 15$
AS	$AS_1 : costs(AS_1) = 2$, $power(AS_1) = 4$	$AS_2 : costs(AS_2) = 20$, $power(AS_2) = 1$
GPS	$GPS_1 : costs(GPS_1) = 6$, $power(GPS_1) = 5$	
CD	$CD_1 : costs(CD_1) = 10$, $power(CD_1) = 10$	$CD_2 : costs(CD_2) = 20$, $power(CD_2) = 4$

specific instances of the generic components with different costs and power consumptions. Table 1 summarizes all the used concrete component implementations.

A valid configuration is now a set of components that fulfills all constraints. For example, when assuming maximum cost of 60, the set $\{PS_1, AS_2, GPS_1, CD_2\}$ is a valid configuration but $\{PS_2, AS_2, GPS_1, CD_2\}$ is not because of violation of the cost constraint.

Throughout this paper we make use of this example and show how SiMoL can be used for modeling such systems.

4 SiMoL Definition

In order to get a clear view on SiMoL, we further present the SiMoL syntax, together with the existing tokens and the corresponding language semantics.

4.1 SiMoL Syntax

As already mentioned, SiMoL uses a Java-like syntax and the common conventions compass most of the defined tokens: identifiers for any type of component and attribute, integer and boolean literals, separators, arithmetic and relational operators ($+, -, *, /, =$, $<, >, <=, >=, !=$), special tokens - comments, reserved words and literals.

Additionally, SiMoL offers support for using *units of measurement*, thus creating a more realistic model.

Another feature of the language, that provides direct control over the possible values of a component attribute, is the *initialization of attributes with integer valued ranges*, represented either by *a sorted list of integer values* or by a *bounded range of integer numbers*. We depict this feature in the following GPS component definition:

```
component GPS{
     attribute int power, costs;
     constraints{
       power={4,6,10} W;
       costs={10..30};
       }}
```

A program written in SiMoL contains basically 3 sections:

- *a knowledge base declaration section*, which is optional. It is used to organize components belonging to a specific domain or problem (similar to a Java package):

  ```
  kbase SensorSystem;
  ```

- *an import declaration section*, also optional:

  ```
  import Sensor.*;
  ```

- *a component definition section*, that is the main constructing unit of a SiMoL program and it is mandatory. Generally, each component will posses a set of attributes and will introduce constraints in the system. The attributes declaration is marked

by the `attribute` keyword, whilst the relations stated between the component attributes and new-component instance-declaration statements appear enclosed in the `constraints{ ... }` block. By convention, an empty component definition section is not allowed, i.e., if the constraints block is missing, we have to declare at least one attribute for the current component. For example, we accept the following definition:

```
component PS{
     attribute int power, costs;
}
```

Furthermore, in the case of *derived components*, the opposite holds: even with no attributes declared, we may state constraints over the inherited attributes. For instance:

```
component AS{
     attribute int power, costs;
     constraints{
       power={4,6} W;
       costs={2..30};
       }}
component AS1 extends AS{
     constraints{
       power=4 W; //equivalent with: power={4} W;
       costs=2;
       }}
```

More details about the implemented inheritance mechanism will be given in subsection 4.2.

In the constraints section, we may have the following types of statements:

- an *empty statement*: `;`,
- a *component instance declaration*, with the possibility of initializing its attributes:

  `GPS1 gps1;` or `GPS1 gps1{costs=100};`

  Using this kind of statements, we define the subcomponent hierarchy in our model, i.e., the partonomy relations. The cardinality of these relations (i.e., the number of subcomponents which can be connected to a certain component) is always finite - we cannot have an unlimited number of components in our model.
- an *arithmetic or/and boolean expression*:

  `ps1.power>=as1.power+gps2.power+cd1.power;`
- a *conditional block*:

```
if(cd1.costs<cd2.costs)
     cost=cost+cd1.costs;
else cost=cost+cd2.costs;
```

- a *forall block*:

```
forall(AS1){
     power=10 W;
     costs={1..10};}
```

- an *exist statement*, for instance:

```
exist(at_most(1),GPS1,costs=30);
```

Due to space reasons, we only mention the built-in functions `min`, `max`, `sum`, `product`, meant to ease the manipulation of large sets of component instances.

4.2 SiMoL Inheritance Mechanism

The ability to extend the functionality and behavior of existing components is of great importance for the taxonomic structure of a configuration domain. In any object oriented languages, the taxonomy relations are represented through the inheritance mechanism. We designed SiMoL with multiple inheritance. In order to demonstrate the necessity of this feature, let us consider the following scenario. For our small system described in Section 3, we introduce a new requirement that refers to a specific signal modulation which can be accomplished by a new component - a modem (M). The modem receives the measured sensor information and transmits the modified signal to the communication device. The function fct from Section 3 will similarly depict for M the modulation-demodulation functionality :

$$fct(M) = \{\textbf{mdm}\}$$

Now the additional constraints of the system become:

$$power(PS) \geq power(AS) + power(GPS) + power(CD) + power(M)$$

$$fct(AS) \cup fct(GPS) \cup fct(CD) \cup fct(M) \supseteq \{\textbf{ad}, \textbf{gps}, \textbf{comm}, \textbf{mdm}\}$$

$$cost(PS) + cost(AS) + cost(GPS) + cost(CD) + cost(M) \leq max\_cost$$

The problem appears if the pre-defined maximum cost is always exceeded, because of the new added component. In other words, we cannot afford both a modem and a communication device. Therefore, a new component type - a communication device with integrated modem (MCD)- will solve the case (under the assumption that $cost(MCD) \leq cost(CD) + cost(M)$). In SiMoL, the MCD definition has the following syntax:

```
component MCD extends CD,M {
     constraints{
       power={4,6} W;
       costs={2..30};
       }}
```

4.3 Semantics of SiMoL

We now specify the semantics of the language SiMoL where we rely on mathematical equations. In particular, we map every statement to a mathematical equation, and combine these equations for a component, taking care of component inheritance and component instances.

For each component C defined in SiMoL we have a set of equations $constr_0$ that is defined within the `constraints { ... }` block. Moreover, the component C also receives equations from its super components and the instances used in the component definition. For example, when specifying `GPS1 gps;` in the constraints section of C, a new instance of `GPS1` is generated. All constraints of `GPS1` are added to the constraints of C. Because of the possibility of having more than one instance of a component, we have to rename the variables used in the constraints of an instance. For this purpose we assume a function $replace$ that takes constraints M and a name N and changes all variables x in M to $N.x$.

$$constr(C) = constr_0(C) \cup constr_I(C) \cup constr_V(C)$$

where $constr_I$ are the constraints inherited from the super components of C

$$constr_I(C) = \bigcup_{C' \in super(C)} constr(C')$$

$constr_V$ are the constraints coming from the components used in the definition of C (and requiring variable renaming using the function $replace$ that add a new pre-fix to the variables used in the components in order to make them unique)

$$constr_V(C) = \bigcup_{(C',N) \in vd\_inst(C)} replace(constr(C'), N)$$

where $vd\_inst(C)$ are the variables used in the constraints of those instances defined in the component C,

and finally, $constr_0$ are the constraints defined in C directly.

$$constr_0 = \bigcup_i C_i$$

A constraint C_i in the constraint definition of C usually takes one of the following forms:

- C_{attr_val} : *attribute-equals-value/s* constraints, formulated with $=$ operator and applied on component attributes together with one single integer/boolean value or with a set of values;
- C_{attr_attr} : *attribute-equals-attr* constraints, formulated with $=$ operator and applied on component attributes;
- C_{num} : *numeric constraints*, formulated with basic relational operators over numeric expressions;
- C_{cond} : *conditional constraints*,
  $if(C_x$ is satisfied) C_y must be satisfied $else\ C_z$ must be satisfied;

– C_{exist} : *existence constraints*,
$exist(at\_least(NR)\,|at\_most(NR)|NR, C, ATTR = VALUE)$, with the meaning that at most, at least or exactly NR components of a given type C have $ATTR = VALUE$.

By means of combining all the upper defined constraints with the multiple inheritance mechanism and the inner component instances declarations, we manage to model the three configuration knowledge forms defined in the chapter on configuration from [16]: the catalog knowledge, the structural knowledge and the component constraints. Hence, we find the expressiveness of our language sufficient for modeling a large number of configuration problems, from simple option selection problems to more complex cases.

As already mentioned, the implementation of SiMoL is based on the MINION constraint solver [17]. In order to simulate the given configuration, we applied a translation function from the SiMoL program to MINION specific input. The systems modeled in SiMoL are always finite. Hence, the complexity of the *SiMoL2MINION* mapping algorithm is polynomial (O(N)), where N is the size of the SiMoL program.

For instance, for a SiMoL program of 12 LOC, the number of constraints of the resulted MINION constraint system is 12; for 104 SiMoL LOC we had 118 MINON constraints, whereas for another SiMoL program of 70 LOC, the resulted constraint system was of size 250. The *SiMoL LOC/ MINION constraints* ratio depends mainly on the type of declared instances - simple instances or vectors of components - and on the complexity of the arithmetic operations. The experiments showed that, even in systems with over 300 component instances, each component possessing in average more than 4 attributes, the time needed to check whether a configuration is valid or not was about 0.18 seconds. In this case, the MINION program had about 700 constraints.

In our implementation, the simulation task consists in checking whether the current configuration, i.e., the system modeled in SiMoL, is consistent. As a result, we may get two possible outcomes: either the system consistency, which means the simulation was successful, or system inconsistency, i.e., there exist no values in the attributes domains that can satisfy all the component requirements. For our small system, the configuration $\{PS_2, AS_2, GPS_1, CD_2\}$ is reported as invalid in less than 0.01 seconds. In the next section, we will discuss this case and suggest methods for re-configuration.

5 Re-Configuration

The characteristics of SiMoL presented in the previous sections demonstrate its usefulness also in the domain of knowledge-based (re-)configuration. We further assume the re-configuration process is triggered by changes in user requirements, in this way, the input for the problem consists in the new requirements together with the current system's structure and behavior, whereas the output is a valid configuration. Changes in the structure and behavior of the system in order to cope with the changes in the requirements represent a solution to the re-configuration problem. When making these changes, we rely on the taxonomic structure of the domain, i.e., the fix set of components given in the knowledge base and their relationships.

For a clear description of the approach, let us go back to our invalid configuration for the small sensors system, namely $\{PS_2, AS_2, GPS_1, CD_2\}$. With respect to the

knowledge base, we assumed that we have 4 generic components PS, AS, CD and GPS, each with the specific implementations depicted in Table 1. In the described configuration, the requirement that the maximum cost is 60 must be fulfilled. Then, the overall cost of the system containing the components $PS2$, $AS2$, $GPS1$, $CD2$ exceeds the pre-defined maximum cost, and thus the constraint $cost(PS) + cost(AS) + cost(GPS) + cost(CD) \leq max\_cost$ is violated. The process of searching for a consistent configuration implies replacing the components (one at a time) from the original configuration with their corresponding sister-components, i.e., components with the same super component. When the new component instance is declared, its constraints replace the old ones in $constr_V(C)$ and the new attributes are propagated across the constraints from $constr_0(C)$. If all the constraints are now satisfied, then we found a solution to our re-configuration problem. In our case, if we change the power supply, for instance, we get the valid configuration $\{PS_1, AS_2, GPS_1, CD_2\}$. Generally, for increasing the search performance, we need to determine two specific orderings: on the one hand, for selecting the component that should be replaced and, on the other hand, for discriminating between sister-components, in case we have more than two components with the same parent. We mention that the re-configuration mechanism is still in early development stage, i.e., it might be subject to change.

6 Conclusion

In this paper, we have presented SiMoL - a new functional-based, declarative modeling language, that serves simulation and re-configuration purposes. The novelty of our approach is designing a language that is easy to learn and capable of modeling large and complex systems. We achieve this by using a state of the art constraint solver. In addition, if due to changes in the user requirements the simulation fails, the configurator offers an alternative model of the system such that the system functionality is kept and the new requirements are fulfilled. SiMoL can coupe with large models and be also efficient with respect to computation time (simulation). Although re-configuration is not fully implemented for the SiMoL language, several ideas are currently analyzed and implemented, such that in the near future a fully working configurator can be used for SiMoL models.

Due to the generality of the language, its scalability with respect to size of the model and solving time (simulation and reconfiguration), but also due to its high flexibility when modeling systems, SiMoL is a multi-purpose language suited for modeling systems specific to a wide range of domains, e.g., telecommunication, automotive systems, electric networks.

In future research we mainly focus on providing a sound and complete configuration algorithm that takes SiMoL models and requirements as input and computes valid configurations as output.

Acknowledgments. The work presented in this paper has been supported by the BRIDGE research project Simulation and configuration of Mobile networks with M2M Applications (SIMOA), which is funded by the FFG.

References

1. McDermott, J.: R1: An Expert in the Computer Systems Domain. In: Proceedings of First National Conference on Artificial Intelligence, AAAI 1980. Stanford University, Stanford (1980)
2. Forgy, C.L., McDermott, J.: OPS, A Domain-Independent Production System Language. In: Proceedings of the Fifth International Joint Conference on Artificial Intelligence, pp. 933–939. MIT (1977)
3. Lehmann, E., Enders, R., Haugeneder, H., Hunze, R., Johnson, C., Schmid, L., Struss, P.: SICONFEX - ein Expertensystem für die Konfigurierung eines Betriebssystems. In: Proceedings of GI Jahrestagung 1985, pp. 792–805 (1985)
4. John, U., Geske, U.: Reconfiguration of Technical Products Using ConBaCon. In: Proceedings of WS on Configuration at AAAI 1999, Orlando (1999)
5. Felfernig, A., Friedrich, G., Jannach, D., Zanker, M.: Semantic Configuration Web Services in the CAWICOMS Project. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 192–205. Springer, Heidelberg (2002)
6. John, U.: Solving large configuration problems efficiently by clustering the ConBaCon model. In: Proceedings of the 13th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: Intelligent Problem Solving: Methodologies and Approaches. Springer-Verlag New York, Inc. (2000)
7. Fleischanderl, G., Friedrich, G.E., Haselböck, A., Schreiner, H., Stumptner, M.: Configuring large systems using generative constraint satisfaction. IEEE Intelligent Systems & their Applications, 59–68 (1998)
8. Stumptner, M., Haselböck, A., Friedrich, G.: COCOS - a tool for constraint-based, dynamic configuration. In: Proceedings of the 10th IEEE Conference on AI Applications (CAIA), San Antonio (1994)
9. Junker, U., Mailharro, D.: The logic of ILOG (J)Configurator: Combining Constraint Programming with a Description Logic. In: Proceedings of IJCAI 2003 Configuration WS, pp. 13–20 (2003)
10. Günter, A., Kreuz, I., Kühn, C.: Kommerzielle Software-Werkzeuge für die Konfigurierung von technischen Systemen. In: Proceedings of KI 1999, pp. 61–65 (1999)
11. Emde, W., Beilken, C., Bording, J., Orth, W., Petersen, U., Rahmer, J., Spenke, M., Voss, A., Wrobel, S., Birlinghoven, S.: Configuration of Telecommunication Systems in KIKon (1996)
12. Orsvärn, K., Axling, T.: The Tacton View of Configuration Tasks and Engines. In: AAAI 1999 Workshop on Configuration, the 16th National Conference on Artificial Intelligence, pp. 127–130 (1999)
13. Prolog, S.: SICStus Prolog 3.12.2 (2005), https://www.sics.se/sicstus/docs/3.12.2/html/sicstus/
14. Haag, A.: Experiences with Product Configuration (2010), http://www.minet.unijena.de/dbis/lehre/ss2010/konfsem/
15. Henson, W.: Real time Control and Custom Components in the Matlab Environment. Technical report
16. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York (2006)
17. Jefferson, C., Kotthoff, L., Moore, N., Nightingale, P., Petrie, K.E., Rendl, A.: The Minion Manual, Minion Version 0.12 (2011), http://minion.sourceforge.net/files/Manual012.pdf/

Author Index