Peter van Emde Boas Frans C.A. Groen
Giuseppe F. Italiano Jerzy Nawrocki
Harald Sack (Eds.)

LNCS 7741

# SOFSEM 2013: Theory and Practice of Computer Science

39th International Conference on Current Trends in Theory and Practice of Computer Science
Špindlerův Mlýn, Czech Republic, January 2013
Proceedings

Springer

Lecture Notes in Computer Science 7741

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

Peter van Emde Boas  Frans C. A. Groen
Giuseppe F. Italiano  Jerzy Nawrocki
Harald Sack (Eds.)

# SOFSEM 2013: Theory and Practice of Computer Science

39th International Conference on Current Trends
in Theory and Practice of Computer Science
Špindlerův Mlýn, Czech Republic, January 26-31, 2013
Proceedings

Springer

Volume Editors

Peter van Emde Boas
University of Amsterdam, The Netherlands
E-mail: p.vanemdeboas@uva.nl

Frans C. A. Groen
University of Amsterdam, The Netherlands
E-mail: f.c.a.groen@uva.nl

Giuseppe F. Italiano
University of Rome Tor Vergata, Italy
E-mail: italiano@disp.uniroma2.it

Jerzy Nawrocki
Poznan University of Technology, Poland
E-mail: jerzy.nawrocki@put.poznan.pl

Harald Sack
Hasso Plattner Institute for Software Systems Engineering
Potsdam, Germany
E-mail: harald.sack@hpi.uni-potsdam.de

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-35842-5 e-ISBN 978-3-642-35843-2
DOI 10.1007/978-3-642-35843-2
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012954777

CR Subject Classification (1998): G.2.2, F.2.2, H.3.3-5, D.2.13, D.2.11, D.2.2, I.2.3-4, H.5.3, H.4.3, E.1, F.1.1, D.1.6

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 2013, the 39th Conference on Current Trends in Theory and Practice of Computer Science, held January 26–31, 2013, in Hotel Bedřichov, 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 (10 this year) and an in-depth coverage of novel research results in selected areas of computer science.
SOFSEM 2013 was organized around the following four tracks:

- *Foundations of Computer Science* (Chair: Giuseppe F. Italiano, University of Rome "Tor Vergata", Italy)
- *Software and Web Engineering* (Chair: Jerzy Nawrocki, Poznan University of Technology, Poland)
- *Data, Information and Knowledge Engineering* (Chair: Harald Sack, University of Potsdam, Germany)
- *Social Computing and Human Factors* (Chair: Frans C.A. Groen, Intelligent Systems Lab Amsterdam, The Netherlands)

With its four tracks, SOFSEM 2013 covered the latest advances in research, both theoretical and applied, in leading areas of computer science. The SOFSEM 2013 Program Committee consisted of 68 international experts from 21 different countries, representing the track areas with outstanding expertise.

An integral part of SOFSEM 2013 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 of presenting student projects on the theory and practice of computer science and to giving students feedback on both the originality of their scientific results and on their work in progress. The papers presented at the Student Research Forum and at the poster section were published in the local proceedings.

In response to the call for papers, SOFSEM 2013 received 117 submissions by 297 authors from 39 different countries of 5 continents (Americas, Asia, Australia, and Europe): full papers were provided for 98 (83 %) of them. From these, there were: 57 submissions in the Foundations of Computer Science Track (58 %) and 55 submissions (42 %) in the remaining three tracks: 22 in the Software and Web Engineering Track (23 %), 16 in the Data, Information and Knowledge Engineering Track (16 %), and 3 in the Social Computing and Human Factors Track (3 %).

After a detailed review process (using the EasyChair Conference System) with at least 3 reviewers per paper, a careful electronic selection procedure was carried

out within each track between August 29th and September 21th, 2012. Following strict criteria of quality and originality, 37 papers were selected for presentation at SOFSEM 2013. From these, there were: 22 submissions in the Foundations of Computer Science Track (59 %) and 15 (41 %) in the other three tracks: 8 in the Software and Web Engineering Track (21 %), 5 in the Data, Information and Knowledge Engineering Track (15 %), and 2 in the Social Computing and Human Factors Track (5 %).

From 98 submitted full papers, 22 submissions were provided by students, and 8 of them were accepted into regular tracks (see above). Based on the recommendations of the Chair of the Student Research Forum and with the approval of the Track Chairs and Program Committee members, 10 student papers were chosen for the SOFSEM 2013 Student Research Forum.

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 the authors of contributed papers. We thank all authors for their prompt responses to our editorial requests.

SOFSEM 2013 was the result of a considerable effort by many people. We would like to express our special thanks to:

- The SOFSEM 2013 Program Committees of 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 2013 Organizing Committee consisting of Martin Řimnáč (Chair), Július Štuller, Pavel Tyl, and Dana Kuželová for the support and preparation of all aspects of the conference
- The Action M Agency, in particular Milena Zeithamlová, for the local arrangements of SOFSEM 2013

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 2013.

Finally, we are very grateful for the financial support of the Czech Society for Cybernetics and Informatics, which enabled us to compose a high-quality program of invited speakers and helped us to keep the student fees low.

November 2012

Peter van Emde Boas
Giuseppe F. Italiano
Jerzy Nawrocki
Harald Sack
Frans C.A. Groen

# Organization

## Steering Committee

| | |
|---|---|
| Ivana Černá | Masaryk University, Brno, Czech Republic |
| Viliam Geffert | P.J. Šafárik University, Košice, Slovakia |
| Keith Jeffery | STFC Rutherford Appleton Laboratory, UK |
| Miroslaw Kutylowski | Wroclaw University of Technology, Poland |
| Jan van Leeuwen | Utrecht University, The Netherlands |
| Branislav Rovan | Comenius University, Bratislava, Slovakia |
| Petr Šaloun, *Observer* | Technical University of Ostrava, Czech Republic |
| Július Štuller, *Chair* | Institute of Computer Science, Academy of Sciences, Czech Republic |

## Program Committee

## PC General Chair

| | |
|---|---|
| Peter van Emde Boas | University of Amsterdam, The Netherlands |

## Track Chairs

| | |
|---|---|
| Giuseppe F. Italiano | University of Rome "Tor Vergata", Italy |
| Jerzy Nawrocki | Poznan University of Technology, Poland |
| Harald Sack | University of Potsdam, Germany |
| Frans C.A. Groen | Intelligent Systems Lab Amsterdam, The Netherlands |

## Student Research Forum Chair

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

## PC Members

| | |
|---|---|
| Ioannis Anagnostopoulos | Lamia, Greece |
| Sören Auer | Leipzig, Germany |
| Roman Barták | Prague, Czech Republic |
| Petr Berka | Prague, Czech Republic |
| Mark de Berg | Eindhoven, The Netherlands |
| Mária Bieliková | Bratislava, Slovakia |
| Přemek Brada | Pilsen, Czech Republic |

| | |
|---|---|
| Ivan Bratko | Lubljana, Slovenia |
| Ondřej Čepek | Prague, Czech Republic |
| Radu G. Danescu | Cluj Napoca, Romania |
| Daniel Delling | Silicon Valley, USA |
| Feng Dong | Luton, UK |
| Werner Dubitzky | Ulster, UK |
| Peter van Emde Boas | Amsterdam, The Netherlands |
| Uwe Egly | Vienna, Austria |
| Thomas Erlebach | Leicester, UK |
| Fedor V. Fomin | Bergen, Norway |
| Dimitris Fotakis | Athens, Greece |
| Gerhard Friedrich | Klagenfurt, Austria |
| Loukas Georgiadis | Ioannina, Greece |
| Siegfried Handschuh | Galway, Ireland |
| Pavel Herout | Pilsen, Czech Republic |
| Martin Homola | Bratislava, Slovakia |
| Andreas Hotho | Würzburg, Germany |
| Zbigniew Huzar | Wroclaw, Poland |
| Anna Ingolfsdottir | Reykjavik, Iceland |
| Riko Jacob | Zürich, Switzerland |
| Klaus Jansen | Kiel, Germany |
| Galina Jirásková | Košice, Slovakia |
| Evangelos Kranakis | Carleton, Canada |
| Milos Kravcik | Aachen, Germany |
| Gregory Kucherov | Paris, France |
| Michal Laclavík | Bratislava, Slovakia |
| Pedro Lima | Lisboa, Portugal |
| Martin Lopez-Nores | Vigo, Spain |
| Leszek Maciaszek | Wroclaw, Poland |
| Lech Madeyski | Wroclaw, Poland |
| Ulrich Mayer | Frankfurt, Germany |
| Roberto Navigli | Rome, Italy |
| Pavol Návrat | Bratislava, Slovakia |
| Anton Nijholt | Enschede, The Netherlands |
| Jakob Nordström | Stockholm, Sweden |
| Andreas Nürnberger | Magdeburg, Germany |
| Miroslaw Ochodek | Poznan, Poland |
| Catuscia Palamidessi | Palaiseau, France |
| Adrian Paschke | Berlin, Germany |
| Dino Pedreschi | Pisa, Italy |
| Tassilo Pellegrini | Vienna, Austria |
| Andreas Pieris | Oxford, UK |
| Giovanni Pighizzini | Milan, Italy |
| Valentina Presutti | Rome, Italy |
| Karel Richta | Prague, Czech Republic |

| | |
|---|---|
| Liam Roditty | Rehovot, Israel |
| Matthew Rowe | Milton Keynes, UK |
| Giuseppe Serra | Firenze, Italy |
| Hadas Shachnai | Haifa, Israel |
| Marc Stevens | Amsterdam, The Netherlands |
| Petr Šaloun | Ostrava, Czech Republic |
| Jiří Šíma | Prague, Czech Republic |
| Július Štuller | Prague, Czech Republic |
| Tarmo Uustalu | Tallinn, Estonia |
| Valentino Vranić | Bratislava, Slovakia |
| Jaroslav Zendulka | Brno, Czech Republic |
| Wolf Zimmermann | Halle Wittenberg, Germany |

## Subreviewers

| | |
|---|---|
| Deepak Ajwani | Rostislav Horčík |
| Judie Attard | Sangxia Huang |
| Matt Austern | Philipp Hupp |
| Balabhaskar Balasundaram | Hans Huttel |
| Tomáš Balyo | Kazuo Iwama |
| Marco Bernardo | Jakub Jurkiewicz |
| Daniela Besozzi | Marta Kasprzak |
| Andreas Brandstädt | Ramtin Khosravi |
| Dorota Bród | Magnus Knuth |
| Matteo Cimini | Nitish Korula |
| Jeremy Debattista | Thomas Krennwallner |
| Yuxin Deng | Petr Kučera |
| Ehab Elsalamouny | Alexander Kurz |
| Moran Feldman | Martin Kutrib |
| Henning Fernau | Michael Lampis |
| Thomas Fernique | Marco Lanzagorta |
| Eldar Fischer | Massimo Lauria |
| Tiziano Flati | Mark Lawson |
| Dominik D. Freydenberger | Tobias Lieber |
| Bernd Gaertner | Markus Lohrey |
| Zsolt Gazdag | Marco Lübbecke |
| Rong Ge | Marco Manna |
| Viliam Geffert | Bodo Manthey |
| Petr Glivický | Tomáš Masopust |
| Petr Gregor | Isabella Mastroeni |
| Elad Haramaty | Arie Matsliah |
| Leo Hatvani | Jannik Matuschke |
| Reiko Heckel | Carlo Mereghetti |
| Christian Hentschel | George Mertzios |
| Lukáš Holý | Peter Bro Miltersen |

Matteo Mio
Matthias Mnich
Tobias Mömke
Michael Morak
Andrea Moro
Mogens Nielsen
Nicolas Nisse
Alexander Okhotin
Claudio Orlandi
Beatrice Palano
Prakash Panangaden
Charis Papadopoulos
Chellapilla Patvardhan
Lukáš Poláček
Petros Potikas
Lars Prädel
Matteo Pradella
Narad Rampersad
Alexander Razborov
Roman Redziejowski
Pierre-Alain Reynier
Christina Robenek
Joshua Sack
Yoshifumi Sakai
Federico Santaron
Petr Savický
Cezary Sobaniec
Petr Sosík
Nadine Steinmetz
Jaroslav Šnajberk
Tami Tamir
Angelo Troina
Miroslav Tůma
Irek Ulidowski
David Wajc
Bartosz Walter
Volker Weichert
Martin Wimmer
Stanislav Žák

## Organization

**SOFSEM 2013** was organized by:

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

## Organizing Committee

| | |
|---|---|
| Martin Řimnáč, *Chair* | Institute of Computer Science, Prague, Czech Republic |
| Július Štuller | Institute of Computer Science, Prague, Czech Republic |
| Pavel Tyl | Technical University Liberec, Czech Republic |
| Dana Kuželová | Institute of Computer Science, Prague, Czech Republic |
| Milena Zeithamlová | 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

## Data, Information, and Knowledge Engineering



## Social Computing and Human Factors

# Theory of Multi Core Hypervisor Verification

Ernie Cohen[1], Wolfgang Paul[2], and Sabine Schmaltz[2]

[1] Microsoft
ecohen@microsoft.com
[2] Saarland University, Germany
{wjp,sabine}@wjpserver.cs.uni-saarland.de

**Abstract.** From 2007 to 2010, researchers from Microsoft and the Verisoft XT project verified code from Hyper-V, a multi-core x-64 hypervisor, using VCC, a verifier for concurrent C code. However, there is a significant gap between code verification of a kernel (such as a hypervisor) and a proof of correctness of a real system running the code. When the project ended in 2010, crucial and tricky portions of the hypervisor product were formally verified, but one was far from having an overall theory of multi core hypervisor correctness even on paper. For example, the kernel code itself has to set up low-level facilities such as its call stack and virtual memory map, and must continue to use memory in a way that justifies the memory model assumed by the compiler and verifier, even though these assumptions are not directly guaranteed by the hardware. Over the last two years, much of the needed theory justifying the approach has been worked out. We survey progress on this theory and identify the work that is left to be done.

## 1 Introduction and Overview

Low-level system software is an important target for formal verification; it represents a relatively small codebase that is widely used, of critical importance, and hard to get right. There have been a number of verification projects targetting such code, particularly *operating system* (OS) kernels. However, they are typically designed as providing a proof of concept, rather than a viable industrial process suitable for realistic code running on modern hardware. One of the goals of the Verisoft XT project [1] was to deal with these issues. Its verification target, the hypervisor Hyper-V [2] was highly optimized, concurrent, shipping C/assembler code running on the most popular PC hardware platform (x64). The verification was done using VCC, a verifier for concurrent C code based on a methodology designed to maximize programmer productivity – instead of using a deep embedding of the language into a proof-checking tool where one can talk directly about the execution of the particular program on the particular hardware.

We were aware that taking this high-level view meant that we were creating a non-trivial gap between the abstractions we used in the software verification and the system on which the software was to execute. For example,

– VCC has an extension allowing it to verify x64 assembly code; why is its approach sound? For example, it would be unsound for the verifier to assume that hardware registers do not change when executing non-assembly code, even though they are not directly modified by the intervening C code.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 1–27, 2013.

- Concurrent C code (and to a lesser extent, the C compiler) tacitly assumes a strong memory model. What justifies executing it on a piece of hardware that provides only weak memory?
- The hypervisor has to manage threads (which involves setting up stacks and implementing thread switch) and memory (which includes managing its own page tables). But most of this management is done with C code, and the C runtime already assumes that this management is done correctly (to make memory behave like memory and threads behave like threads). Is this reasoning circular?

When we started the project, we had ideas of how to justify all of these pretenses, but had not worked out the details. Our purpose here is to i) outline the supporting theory, ii) review those parts of the theory that have already been worked out over the last few years, and iii) identify the parts of the theory that still have to be worked out.

### 1.1 Correctness of Operating System Kernels and Hypervisors

Hypervisors are, at their core, OS kernels, and every basic class about theoretical computer science presents something extremely close to the correctness proof of a kernel, namely the simulation of $k$ one-tape *Turing machines* (TMs) by a single $k$-tape TM [3]. Turning that construction into a simulation of $k$ one-tape TMs by a single one-tape TM (*virtualization* of $k$ *guest* machines by one *host* machine) is a simple exercise. The standard solution is illustrated in Figure 1. The tape of the host machine is subdivided into tracks, each representing the tape of one of the guest machines (*address translation*). Head position and state of the guest machines are stored on a dedicated field of the track of that machine (a kind of *process control block*). Steps of the guests are simulated by the host in a round robin way (a special way of *scheduling*). If we add an extra track for the data structures of the host and add some basic mechanisms for communications between guests (*inter process communication*) via *system calls*, we have nothing less than a one-tape TM kernel. Generalizing from TMs to an arbitrary computation model $M$ (and adding I/O-devices), one can specify an $M$ kernel as a program running on a machine of type $M$ that provides

- virtualization: the simulation of $k$ guest machines of type $M$ on a single host machine of type $M$
- system calls: some basic communication mechanisms between guests, I/O devices, and the kernel

At least as far as the virtualization part is concerned, a kernel correctness theorem is essentially like the Turing machine simulation theorem, and can likewise be conveniently expressed as a forward simulation. For more realistic kernels, instead of TMs we have processors, described in dauntingly large manuals, like those for the MIPS32 [4] (336 pages), PowerPC [5] (640 pages), x86 or x64 [6, 7] (approx. 1500, resp. 3000 pages). The TM tape is replaced by RAM, and the tape head is replaced by a *memory management unit* (MMU), with address translation driven by in-memory *page tables*. Observe that a mathematical model of such machine is part of the definition of correctness for a 'real' kernel.

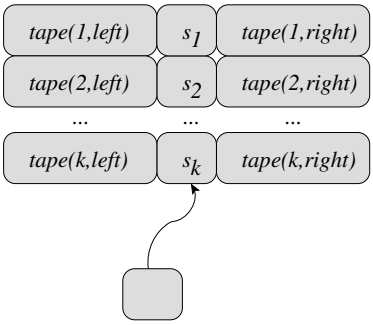



**Fig. 1.** Simulating $k$ Turing machines with 1 $k$-band Turing machine

Hypervisors are kernels whose guests are themselves operating systems or kernels, i.e. each guest can run several *user processes*. In terms of the TM simulation, each guest track is subdivided into subtracks for each user, each subtrack having its own process control block; the actual tape address for the next operation of a process can be calculated from its tape address, the layout of the subtrack within the track, and the layout of the track itself. In a real kernel, the address of a memory access is calculated from a virtual address using two levels of address translation, the first level of traslation provided by the guest to users via the *guest page tables* (GPTs), and the second provided by the hypervisor to the guest. On many recent processors, this second level of address translation is provided in hardware by a separate set of *host page tables*. On processors providing only a single level of translation, it is possible to take advantage of the fact that the composition of two translations is again a translation, and so can be provided by a single set of page tables. Because these *shadow page tables* (SPTs) correspond to neither the guest nor the host page tables, they are constructed on the fly by the hypervisor from the GPTs, and the hypervisor must hide from the guest that translation goes through these tables rather than the GPTs. Thus, the combined efforts of the hypervisor and the MMU simulate a virtual MMU for each guest.

### 1.2 Overview

We discuss the following seven theories in the remainder of the paper:

**Multi-Core ISA-sp.** We define a nondeterministic concurrent *instruction set architecture* (ISA) model, suitable for system programming. In addition to processor cores and main memory, it includes low-level (but architecturally visible) features such as store buffers, caches, and memory management units. Ideally, this would be given in (or at least derived from) the system programmer's manuals published by the chip manufacturers. In reality, many subtle (but essential) details are omitted from these manuals. Indeed, hardware manufacturers often deliberately avoid commiting themselves to architectural boundaries, to maximize their flexibility in optimizing the implementation, and many details leveraged by real operating systems (such as details concerning the walking of page tables) are shared only with their most

important customers, under agreements of nondisclosure. Such fuzzy architectural boundaries are acceptable when clients are writing operating systems; a progammer can choose whether to program to a conservative model (e.g., by flushing translations after every change to the page tables) or program more aggressively to a model that takes advantage of architectural details of the current processor generation. But such a fuzzy model is fatal when trying to build an efficient hypervisor, because the architectural specification must both be strong enough for client operating systems to run correctly, yet weak enough that it can be implemented efficiently on top of the available hardware.

We provide evidence for the particular model we use and for the particular ways in which we resolved ambiguities of the manuals in the following way: i) we define a simplified ISA-sp that we call MIPS-86, which is simply MIPS processor cores extended with x86-64 like architecture features (in particular, memory system and interrupt controllers), ii) we reverse engineer the machine in a plausibly efficient way at the gate level, and iii) we prove that the construction meets the ISA-sp, and iv) we confirm with OS engineers that the model is sufficiently strong to support the memory management algorithms used in real operating systems. The correctness theorems in this theory deal with the correctness of hardware for multi-core processors at the gate level.

**ISA Abstraction.** Multi-core machines are primarily optimized to efficiently run ordinary user code (as defined in the user programming manuals). In this simplified instruction set (ISA-u), architectural details like caches, page tables, MMUs, and store buffers should be transparent, and multithreaded programs should see sequentially consistent memory (assuming that they follow a suitable synchronization discipline). A naive discipline combines lock-protected data with shared variables, where writes to shared variables flush the store buffer. A slightly more sophisticated and efficient discipline requires a flush only when switching from writing to reading [8]. After proper configuration, a simulation between ISA-sp and ISA-u has to be shown in this theory for programs obeying such disciplines.

**Serial Language Stack.** A realistic kernel is mostly written in a high-level language (typically C or C++) with small parts written in macro assembler (which likewise provides the stack abstraction) and even smaller parts written in plain assembler (where the implementation of the stack using hardware registers is exposed, to allow operations like thread switch). The main definition of this theory is the formal semantics of this computational model. The main theorem is a combined correctness proof of optimizing compiler + macro assembler for this mixed language. Note that compilers translate from a source language to a clean assembly language, i.e. to ISA-u.

**Adding Devices.** Formal models for at least two types of devices must be defined: regular devices and interrupt controllers (the APIC in x86/64). A particularly useful example device is a hard disk – which is needed for booting. Interrupt controllers are needed to handle both external interrupts and interrupt-driven interprocess communication (and must be virtualized by the hypervisor since they belong to the architecture). Note that interrupt controllers are very particular kinds of devices in the sense that they are interconnected among each other and with processor cores in a way regular devices are not: They inject interrupts collected from regular

devices and other interrupt controllers directly into the processor core. Thus, interrupt controllers must be considered specifically as part of an ISA-sp model with instantiable devices[1]. Crucial definitions in this theory are i) sequential models for the devices, ii) concurrent models for ISA-sp with devices, and iii) models for single core processors semantics of C with devices (accessed through *memory mapped I/O* (MMIO)). The crucial theorems of this theory show the correctness of drivers at the code level.

**Extending the Serial Language Stack with Devices to Multi-Core Machines.** The crucial definition of this theory is the semantics of concurrent 'C + macro assembly + ISA-sp + devices'. Besides ISA-sp, this is *the* crucial definition of the overall theory, because it defines the language/computational model in which multi-core hypervisors are coded. Without this semantics, complete code level verification of a hypervisor is not meaningful. Essentially, the ownership discipline of the ISA abstraction theory is lifted to the C level; in order to enable the implementation of the ownership discipline, one has to extend serial C with volatile variables and a small number of compiler intrinsics (fences and atomic instructions). In this theory there are two types of major theorems. The first is compiler correctness: if the functions of a concurrent C program obeying the ownership discipline are compiled separately, then the resulting ISA-u code obeys the ownership discipline and the multi-core ISA-u code simulates the parallel C code. The second is a reduction theorem that allows us to pretend that a concurrent C program has scheduler boundaries only just before actions that race with other threads (I/O operations and accesses to volatile variables).

**Soundness of VCC and its Use.** Programs in the concurrent C are verified using VCC. In order to argue that the formal proofs obtained in this way are meaningful, one has to prove the soundness of VCC for reasoning about concurrent C programs, and one has to show how to use VCC in a sound way to argue about programs in the richer models.

Obviously, for the first task, syntax and semantics of the annotation language of VCC has to be defined. VCC annotations consist essentially of "ghost" (a.k.a. "auxilliary" or "specification") state, ghost code (used to facilitate reasoning about the program, but not seen by the compiler) and annotations of the form "this is true here" (e.g. function pre/post-conditions, loop invariants, and data invariants). Then three kinds of results have to be proven. First, we must show that if a program (together with its ghost code) is certified by VCC, then the "this is true here" assertions do in fact hold for all executions. Second, we must show that the program with the ghost code simulates the program without the ghost code (which depends on VCC checking that there is no flow from ghost state to concrete state, and that all ghost code terminates). Third, we must show that the verification implies that the program conforms to the Cohen/Schirmer [8] ownership discipline (to justify VCC's assumption of a sequentially consistent model of concurrent C).

To reason about richer programming models with VCC, we take advantage of the fact that the needed extensions can be encoded using C. In particular, one can

[1] MIPS-86 provides such an ISA-sp model with interrupt controllers and instantiable devices – albeit currently at a level where caches are already invisible.

add additional ghost data representing the states of processor registers, MMUs and devices to a C program; this state must be stored in "hybrid" memory that exists outside of the usual C address space but from which information can flow to C memory. We then represent assembly instructions as function calls, and represent active entities like MMUs and devices by concurrently running C threads.

**Hypervisor Correctness.** The previous theories serve to provide a firm foundation for the real verification work, and to extend classical verification technology for serial programs to the rich computational models that are necessarily involved in (full) multi-core hypervisor verification. Verification of the hypervisor code itself involves several major components, including i) the implemention of a large numbers of 'C + macro assembly + assembly' threads on a multi-core processor with a fixed small number of cores, ii) for host hardware whose MMUs do not support two levels of translations, the correctness of a parallel shadow page table algorithm, iii) a TM-type simulation theorem showing virtualization of ISA-sp guests by the host, and iv) correct implementation of system calls.

## 2 ISA Specification and Processor Correctness

### 2.1 Related Work

For single core RISC (*reduced instruction set computer*) processors, it is well understood how to specify an ISA and how to formally prove hardware correctness. In the academic world, the papers [9] and [10] report the specification and formal verification of a MIPS-like processor with a pipelined core with forwarding and hardware interlock, internal interrupts, caches, a fully IEEE compliant pipelined floating point unit, a Tomasulo scheduler for out of order execution, and MMUs for single-level pages tables. In industry, the processor core of a high-end controller has been formally verified [11]. To our knowledge, there is no complete formal model for any modern commercial CISC (*complex instruction set computer*); until recently, the best approximations to such a model were C simulators for large portions of the instruction set [12–14].

The classical memory model for multi-core processors is Lamport's sequentially consistent shared memory [15]. However, most modern multi-core processors provide efficient implementations only of weaker memory models. The most accurate model of the memory system of modern x86/64 architectures, "x86-tso", is presented in [16]. This model abstracts away caches and the memory modes specifying the cache coherence protocol to be used, and presents the memory system as a sequentially consistent shared memory, with a separate FIFO store buffer for each processor core. It is easy to show that the model collapses if one mixes in the same computation cacheable and non cacheable memory modes on the same address (accesses in non cacheable memory modes bypass the cache; accesses in different non cacheable modes have different side effects on the caches). That the view of a sequentially consistent shared memory can be maintained even if of one mixes in the same computation accesses to the same address with different "compatible" memory modes/coherence protocols is claimed in the classical paper introducing the MOESI protocol [17], but we are not aware of any proof of this fact.

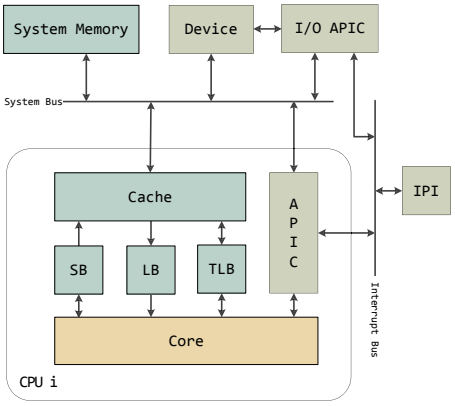



**Fig. 2.** x86-64 processor model consisting of components whose steps are interleaved non-deterministically

Another surprising observation concerns correctness proofs for cache coherence protocols. The model checking literature abounds with papers showing that certain invariants of a cache system are maintained in an interleaving model where the individual cache steps are atomic. The most important of these invariants states that data for the same memory address present in two caches are identical and thus guarantees a consistent view on the memory at all caches. For a survey, see [18]. These results obviously provide an important step towards the provably correct construction of a sequentially consistent shared memory. Apart from our own results in [19], we are not aware of a gate-level construction of hardware main memory, caches, cache controllers, and the busses connecting them for which it has been proved that parallel execution of hardware accesses to the caches simulates the high-level cache model.

## 2.2 Modeling an x86-64-Like ISA-sp

A formal model of a very large subset of the x64 ISA-sp was constructed as part of the Hyper-V verification project, and is presented in [20]. This 300 page model specifies 140 general purpose and system programming instructions. Due to time constraints, the model omits debug facilities, the alignment check exception, virtual-8086 mode, virtual interrupts, hardware task-switching, system management mode, and devices other than the local APICs. The MMX extension of the instruction set is formalized in the complementary thesis [21]. The instruction set architecture is modeled by a set of communicating nondeterministic components as illustrated in Figure 2. For each processor, there is a processor core, MMU, store buffer, caches (which become visible when accesses of non cacheable and cacheable memory modes to the same address are mixed in the same computation), and a local APIC for interrupt handling. The remaining components (shared between the cores) are main memory and other devices. Sizes of caches, buffers, and translation look aside buffers (TLBs) in the MMU are unbounded in the model, but the model is sufficiently nondeterministic to be implemented by an implementation using arbitrary specific sizes for each of these. In the same spirit, caches

and MMUs nondeterministically load data within wide limits, allowing the model to be implemented using a variety of prefetch strategies. Nevertheless, the model is precise enough to permit proofs of program correctness.

As mentioned in the introduction, an accurate ISA-specification is more complex than meets the eye. Only if the executed code obeys a nontrivial set of software conditions, the hardware interprets instructions in the way specified in the manuals. In RISC machines, the alignment of accesses is a typical such condition. In pipelined machines, the effects of certain instructions only become visible at the ISA level after a certain number of instructions have been executed, or after an explicit pipeline flush. In the same spirit, a write to a page table becomes visible at the ISA level when the instruction has left the memory stages of the pipe, the write has left the store buffer, *and* previous translations effected by this write are flushed from the TLB by an INVLPG instruction (which in turn does only become visible when it has left the pipe). In a multi-core machine, things are even more complicated because a processor can change code and page tables of other processors. In the end, one also needs *some* specification of what the hardware does if the software violates the conditions, since the kernel generally cannot exclude their violation in guest code. In turn, one needs to guarantee that guest code violating software conditions does not violate the integrity of other user processes or the kernel itself. Each of these conditions exposes to the ISA programmer details of the hardware, in particular of the pipeline, in a limited way.

Obviously, if one wants to verify ISA programs, one has to check that they satisfy the software conditions. This raises the problem of how to identify a *complete* set of these conditions. In order to construct this set, we propose to reverse engineer the processor hardware, prove that it interprets the instructions set, and collect the software conditions we use in the correctness proof of the hardware. Reverse engineering a CISC machine as specified in [20] is an extremely large project, but if we replace the CISC core by a MIPS core and restrict memory modes to 'write back' (WB) and 'uncacheable' (UC) (for device accesses), reverse engineering becomes feasible. A definition of the corresponding instruction set called 'MIPS-86' fits on 44 pages and can be found in [22].

### 2.3 Gate Level Correctness for Multi-core Processors

A detailed correctness proof of a multi-core processor for an important subset of the MIPS-86 instruction set mentioned above can be found in the lecture notes [19]. The processor cores have classical 5 stage pipelines, the memory system supports memory accesses by bytes, half words, and words, and the caches implement the MOESI protocol. Caches are connected to each other by an open collector bus and to main memory (realized by dynamic RAM) by a tri-state bus. There are no store buffers or MMUs, yet. Caches support only the 'write back' mode. The lecture notes contain a gate-level correctness proof for a sequentially consistent shared memory on 60 pages. Integrating the pipelined processor cores into this memory system is not completely trivial, and proving that this implements the MIPS-86 ISA takes another 50 pages. The present proof assumes the absence of self-modifying code.

Dealing with tri-state busses, open collector busses, and dynamic RAM involves design rules, which can be formulated but not motivated in a gate-level model. In analogy

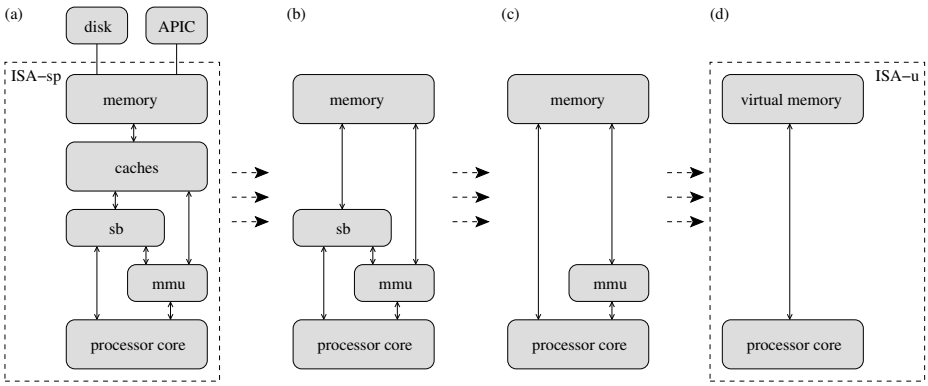



**Fig. 3.** Abstracting ISA-sp to ISA-u

to data sheets of hardware components, [19] therefore uses a detailed hardware model with minimal and maximal propagation delays, enable and disable times of drivers, and setup and hold times of registers as its basic model. This allows derivation of the design rules mentioned above. The usual digital hardware model is then derived as an abstraction of the detailed model.

### 2.4 Future Work

The correctness proof from [19] has to be extended in the following ways to cover MIPS-86

- introducing a read-modify-write operation (easy),
- introducing memory fences (easy),
- extending memory modes to include an uncacheable mode UC (easy),
- extending the ISA-sp of MIPS-86 with more memory modes and providing an implementation with a coherence protocol that keeps the view of a single memory abstraction if only cacheable modes are used (easy)
- implementing interrupt handling and devices (subtle),
- implementing an MMU to perform address translation (subtle),
- adding store buffers (easy), and
- including a Tomasulo scheduler for out-of-order execution (hard).

## 3 Abstracting ISA-sp to ISA-u

One abstracts ISA-sp to ISA-u in three steps: i) eliminating the caches, ii) eliminating the store buffers, and iii) eliminating the MMUs. A complete reduction (for a naive store buffer elimination discipline and a simplified ownership discipline) is given in [23].

### 3.1 Caches

To eliminate caches, an easy simulation shows that the system with caches $S$ simulates a system without caches $S'$; the coupling invariant is that for every address, the $S'$ value at that address is defined as the value cached at that address if it is cached (the value is unique, since all cache copies agree), and is the value stored in the $S$ memory if the location is uncached. One gets the processor view of Figure 3(b).

### 3.2 Store Buffers and Ownership Disciplines

In single-core architectures, an easy simulation shows that the system with FIFO store buffers $S$ simulates a system without store buffers $S'$: the value stored at an address in $S'$ is the value in the store buffer farthest away from memory (i.e., the last value saved) if the address appears in the store buffer, and is otherwise the value stored in the memory of $S$. For a single-core architecture, no extra software conditions are needed; a more careful proof can be found in [24]. One gets the view of Figure 3(c). For the multi-core architecture, store buffers can only be made invisible if the executed code follows additional restrictions.

A trivial (but highly impractical) discipline is to use only flushing writes (which includes atomic read-modify-write operations); this has the effect of keeping the store buffers empty, thus rendering them invisible. A slightly more sophisticated discipline is to classify each address as either shared or owned by a single processor. Unshared locations can be accessed only by code in the owning processor; writes to shared addresses must flush the store buffer. The proof that this simulates a system without store buffers is almost the same as in the uniprocessor case: for each owned address, its value in the $S'$ memory is the value in its owning processor according to the uniprocessor simulation, and for each shared address, the value is the value stored in memory.

A still more sophisticated discipline to use the same rule, but to require a flush only between a shared write and a subsequent share read on the same processor. In this case, a simple simulation via a coupling invariant is not possible, because the system, while sequentially consistent, is not linearizable. Instead, $S'$ issues a write when the corresponding write in $S$ actually emerges from the store buffer and hits the memory. $S'$ issues a shared read at the same time as $S$; this is consistent with the writes because shared reads happen only when there are no shared writes in the store buffer. The unshared reads and writes are moved to fit with the shared writes[2]. It is straightforward to extend this reduction to include shared "read-only" memory.

A final, rather surprising improvement to the last reduction discipline is to allow locations to change from one type to another programatically. For example, we would like to have a shared location representing a lock, where an ordinary operation on that lock (acquiring it) gives the thread performing that action ownership of some location

[2] Note that this means that in $S'$, an unshared write from one processor might be reordered to happen before a shared write from another processor, even though the shared write hits memory first, so while the execution is sequentially consistent, it is not "memory sequential consistent" as defined in [25], because it violates the triangle race condition. Allowing sequentially consistent executions with triangle races is an absolute requirement for practical reduction theorems for x86-TSO.

protected by that lock. Moreover, we might want to allow the set of locations protected by that lock to change, perhaps determined by data values. [8] gives a very general reduction theorem for x86-TSO that allows these things to be done in the most flexible way possible, by allowing the program to take ownership of shared data, give up ownership of data, and change it between being read-only and read-write, in ordinary ghost code. This theorem says that if you can prove, *assuming sequential consistency*, that a concurrent program (which includes ghost code that might change memory types of locations) follows (a minor modification of) the flushing discipline above, then the program remains sequentially consistent when executed under x86-TSO. The proof of this reduction theorem is much more difficult than the previous ones, because reducing a single TSO history requires reasoning from the absence of certain races in related sequentially consistent histories.

### 3.3 Eliminating MMUs

Modern processors use page tables to control the mapping of virtual to physical addresses. However, page tables provide this translation only indirectly; the hardware has to walk these page tables, caching the walks (and even partial walks) in the hardware TLBs. x86/64 machines require the system programmer to manage the coherence of the TLBs in response to changes in the page tables. The simplest way to make MMUs invisible is to set up a page table tree that represents an injective translation (and does not map the page tables themselves), before switching on virtual memory. It is an easy folklore theorem that the resulting system simulates unvirtualized memory; a proof can be found in [24]. One gets the view of Figure 3(d). However, this is not how real kernels manage memory; memory is constantly being mapped in and unmapped. The easiest way to do this is to map the page tables in at their physical addresses (since page table entries are based on physical, rather than virtual, page frame numbers). At the other extreme, one can model the TLBs explicitly, and keep track of those addresses that are guaranteed to be mapped to a particular address in all possible complete TLB walks (and to not have any walks that result in a page fault), and to keep track of a subset of these addresses, the "valid" addresses[3], such that the induced map on these addresses is injective. Only those addresses satisfying these criteria can be read or written. This flexibility is necessary for kernels that manage memory agressively, trying to minimize the number of TLB flushes. Essentially, this amounts to treating the TLB in the same way as a device, but with the additional proof obligation connecting memory management to reading and writing, through address validity. This, however, we currently consider future work.

### 3.4 Mixed ISA-sp and ISA-u Computations

In a typical kernel, there is a stark contrast between the kernel code and the user programs running under the kernel. The kernel program needs a richer model that includes system instructions not accessible to user programs, but at the same time the kernel can

[3] Note that validity of an address is, in general, different for different processors in a given state, since they flush their TLB entries independently.

be written using a programming discipline that eliminates many irrelevant and mathematically inconvenient details. For example, if the kernel is being proved memory-safe, the programming model in the kernel does not have to assign semantics to dereferencing of null pointers or overrunning array bounds, whereas the kernel must provide to user code a more complex semantics that takes such possibilities into account. Similarly, the kernel might obey a synchronization discipline that guarantees sequential consistency, but since user code cannot be constrained to follow such a discipline, the kernel must expose to user code the now architecturally-visible store buffers. In the case of a hypervisor, the guests are themselves operating systems, so the MMU, which is conveniently hidden from the hypervisor code (other than boot code and the memory manager), is exposed to guests.

### 3.5 Future Work

Extension of the work in [23] to a full a proof of the naive store buffer reduction theorem should not be hard. In order to obtain the reduction theorem with dirty bits, it is clearly necessary to extend the store buffer reduction theorem of [8] to machines with MMUs. This extension is not completely trivial as MMUs directly access the caches without store buffers. Moreover MMUs do not only perform read accesses; they write to the 'accessed' and 'dirty' bits of page table entries. One way to treat MMUs and store buffers in a unified way is to treat the TLB as shared data (in a separate address space) and the MMU as a separate thread (with an always-empty store buffer). This does not quite work with the store buffer reduction theorem above; because the TLB is shared data, reading the TLB to obtain an address translation for memory access (which is done by the program thread) would have to flush the store buffer if it might contain a shared write, which is not what we want. However, the reduction theorem of [8] can be generalized so that a shared read does not require a flush as long as the same read can succeed when the read "emerges" from the store buffer; this condition is easily satisfied by the TLB, because a TLB of unbounded capacity can be assumed to grow monotonically between store buffer flushes.

## 4 Serial Language Stack

### 4.1 Using Consistency Relations to Switch between Languages

As explained in the introduction, realistic kernel code consists mostly of high-level language code, with some assembler and possibly some macro assembler. Thus, complete verification requires semantics for programs composed of several languages $L_k$ with $0 \leq k < n \in \mathbb{N}$. Since all these languages are, at some point, compiled to some machine code language $L$, we establish for each $L_k$ that programs $p \in L_k$ are translated to programs $q \in L$ in such a way that *computations* $(d^i)$ – i.e. sequences of configurations $d^i, i \in \mathbb{N}$ – of program $q$ simulate computations $(c^i)$ of program $p$ via a consistency relation $consis(c, d)$ between high level configurations $c$ and low level configurations $d$. Translation is done by compilers and macro assemblers. Translators can be optimizing or not. For non-optimizing translators, steps of language $L_k$ are translated into one

or more steps of language $L$. One shows that, for each computation $(c^i)$ in the source language, there is a step function $s$ such that one has

$$\forall i : consis(c^i, d^{s(i)})$$

If the translator is optimizing, consistency holds only at a subset of so called 'consistency points' of the source language. The translator does not optimize over these points. Let $CP(i)$ be a predicate indicating that configuration $c^i$ is a consistency point. Then an optimizing translator satisfies

$$\forall i : CP(i) \rightarrow consis(c^i, d^{s(i)})$$

Note that the consistency relation and the consistency points together specify the compiler. The basic idea to formulate mixed language semantics is very simple. We explain it here only for two language levels (which can be thought of as machine code and high level abstract semantics, as in Figure 4); extension to more levels is straightforward and occurs naturally when there is a model stack of intermediate languages for compilation[4]. Imagine the computations $(c^i)$ of the source program and $(q^j)$ of the target program as running in parallel from consistency point to consistency point. We assume the translator does not optimize over changes of language levels, so configurations $c^i$ where the language level changes are consistency points of the high level language. Now there are two cases

- switching from $L_k$ to $L$ in configuration $c^i$ of the high level language: we know $consis(c^i, d^{s(i)})$ and continue from $d^{s(i)}$ using the semantics of language $L$.
- switching from $L$ to $L_k$ in configuration $d^j$ of the low level language: we try to find a configuration $c'$ of the high level language such that $consis(c', d^j)$ holds. If we find it we continue from $c'$ using the semantics of the high level langue. If we do not find a consistent high level language configuration, the low level portion of the program has messed up the simulation and the semantics of the mixed program switches to an error state.

In many cases, switching between high-level languages $L_k$ and $L_l$ by going from $L_k$ to shared language $L$, and from the resulting configuration in $L$ to $L_l$ can be simplified to a direct transition from a configuration of $L_k$ to a configuration of $L_l$ by formalizing just the compiler calling convention and then proving that the resulting step is equivalent to applying the two involved consistency relations (e.g., see [26]). This explains why the specification of compilers necessarily enters into the verification of modern kernels.

### 4.2 Related Work

The formal verification of a non-optimizing compiler for the language C0, a type safe PASCAL-like subset of C, is reported in [27]. The formal verification of an optimizing

[4] Note in particular, that, when two given high level language compilers have an intermediate language in common, we only need to switch downwards to the highest level shared intermediate language.

compiler for the intermediate language C-minor is reported in [28]. Mixed language semantics for C0 + in line assembly is described in [29] as part of the Verisoft project [11]. Semantics for C0 + external assembly functions is described in [30]. Both mixed language semantics were used in sebsequent verification work of the Verisoft project. As the underlying C0 compiler was not optimizing, there was only a trivial calling convention. Note that the nontrivial calling conventions of optimizing compilers produce proof goals for external assembly functions: one has to show that the calling conventions are obeyed by these functions. Only if these proof goals are discharged, one can show that the combined C program with the external functions is compiled correctly.

### 4.3 A Serial Language Stack for Hypervisor Verification

Similar to [28], we use an intermediate language C-IL with address arithmetic and function pointers. The semantics of C-IL *together* with a macro assembler obeying the same calling conventions is described in [26]. Calls from C-IL to macro assembly and vice versa are allowed. To specify the combined semantics, one has to describe the ABI (i.e. the layout of stack frames and the calling convention used). In [31], an optimizing compiler for C-IL is specified, a macro assembler is constructed and proven correct, and it is shown how to combine C-IL compiler + macro assembler to a translator for combined C-IL + macro assembly programs. As explained in the introduction, extension of this language stack to C-IL + macro assembly + assembly is necessary to argue about saving and restoring the base and stack pointers during a process switch or a task switch. This can be done using the construction explained in subsection 4.1.

### 4.4 Future Work

We believe that, for the consistency relations normally used for specifying compilers and macro assemblers, the mixed language semantics defined in subsection 4.1 is essentially deterministic in the following sense: if $consis(c, d)$ holds, then $d$ is unique up to portions of $c$ which will not affect the future I/O behavior of the program (e.g. non reachable portions of the heap). A proof of such a theorem should be written down.

## 5 Adding Devices

The obvious way to add devices is to represent them as concurrent threads, and to reason about the combined program in the usual way. This approach is justified only if the operational semantics of the language stack executing the program in parallel with the device models simulates the behavior of the compiled code running on the hardware in parallel with the devices. This is already nontrivial, but is further complicated by the addition of interrupts and interrupt handlers. It is obviously undesirable to introduce interrupt handling as a separate linguistic concept, so the natural way to model an interrupt handler is as a concurrent thread. However, the relationship between a program and an interrupting routine is somewhat closer than that between independent threads; for example, data that might be considered "thread local" in the context of a concurrent

program might nevertheless be modified by an interrupt handler, which requires careful management of when interrupts are enabled and disabled. Another complication that arises with many kinds of devices is the need to capture and model real-time constraints.

### 5.1 Related Work

Formal methods have been extremely successful identifying large classes of frequent bugs in drivers [32]. In contrast, *complete* formal verification results of even of the most simple drivers have only recently appeared. An obvious prerequisite is a formal model of devices that can be integrated with processor models both at the hardware and ISA level [33]. At the hardware level, processor and devices work in parallel in the same clock domain. At the hardware level, the models of some devices are completely deterministic; an example is a dedicated device producing timer interrupts. But these models also can have nondeterministic portions, e.g. the response time (measured in hardware cycles) of a disk access. When we lift the hardware construction to the ISA model, one arrives in a natural way at a nondeterministic concurrent model of computation: processor and device steps are interleaved in an order not known to the programmer at the ISA level or above. This order observed at the ISA level can be constructed from the hardware construction and the nondeterminism stemming from the device models. A formal proof for the correctness of such a concurrent ISA model for a single core 'processor + devices' was given in [34, 35]. The hardware construction for catching the external interrupts and the corresponding correctness argument are somewhat tricky due to an - at first sight completely harmless - nonstandard specification in the instruction set of the underlying processor, which was taken from [36]. There, external interrupts are defined to be of type 'continue', i.e. the interrupted instruction is completed before the interrupt is handled. In the MIPS-86 instruction set [22] mentioned above, this definition was changed to reflect standard specification, where external interrupts are of type 'repeat', i.e. the interrupted instruction is not executed immediately, but is instead repeated after the run of the handler.

Now consider a system consisting of a (single core) processor and $k$ devices as shown in Figure 5, and consider a run of a driver for device $i$. Then one wants to specify the behavior of the driver by pre and post conditions. For example, if the driver writes a page from the processor to the disk, the precondition would state that the page is at a certain place in processor memory and the post condition would specify that it is stored at a certain place on the memory of the disk. To prove this one has to show that the other devices do not interfere with the driver run. Indeed one can show an order reduction theorem showing that if during the driver run i) interrupts of other devices are disabled and ii) the processor does not poll the devices, then in a driver run with arbitrary interleaving all steps of devices $\neq i$ can be reordered such that they occur after the driver run without affecting the result of the computation. A formal proof of this result is given in [30, 37]. At the same place and in [38] the integration of devices into the serial model stack of the Verisoft project (resulting in C0 + assembly + devices) and the formal verification of disk drivers is reported.

Note that the above reorder theorem for device steps has the nontrivial hypothesis that there are no side channels via the environment, i.e. the outside world between

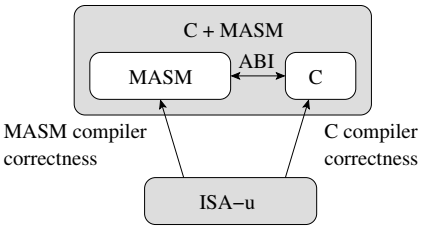



**Fig. 4.** Combined semantics of C and Macro Assembler

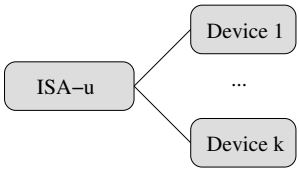



**Fig. 5.** ISA model with devices

the devices. This is not explicitely stated; instead it it is implicitly assumed by formalizing Figure 5 in the obvious way. For example, if device 2 is a timer triggering a gun aimed at device 1 during the driver run of device 1, the post condition is false after the run because the device is not there any more. Side channels abound of course, in particular in real time systems. If device 1 is the motor control and device 2 the climate control in a car, then the devices *are* coupled in the environment via the power consumption.

### 5.2 Multi-core Processors and Devices

Only some very first steps have been made towards justifying verification of multi-core processors along with devices. The current MIPS-86 instruction set contains a generic device model and a specification of a simplified APIC system consisting of an I/O APIC (a device shared between processors that distributes device interrupts) and local APICs (processor local devices for handling external and inter processor interrupts). Rudimentary models of interrupt controllers for various architectures have been built as parts of VCC verifications.

### 5.3 Future Work

Instantiating the generic device model of MIPS-86 with an existing formal disk model is straightforward. Justification of the concurrent 'multi-core processor + devices model' of the MIPS-86 ISA requires of course the following steps

- extending the MIPS-ISA hardware from [19] with the pipelined interrupt mechanism from [10]. The catching and triggering of external interrupts needs to be modified to reflect that external interrupts are now of type 'repeat'. This should lead to a simplification of the construction.
- reverse engineering of hardware APICs and the mechanism for delivering inter processor interrupts (IPIs).
- showing that the hardware constructed in this ways interprets the MIPS-86 ISA. This proof should be simpler than the proof in [34].

Three more tasks remain open: i) proving a reorder theorem for driver runs, ii) the reduction theorem from multi-core ISA-sp to ISA-u has to be generalized to the situation, where the hardware contains devices, and iii) devices must be integrated in the serial model stack (resulting in C-IL + macro assembly + assembly + devices) along the lines of [30, 37, 38]. These results would justify language-level reasoning about device drivers in multi-core systems.

Formally verifying secure booting is another interesting research direction: 'secure boot' guarantees that the verified hypervisor (if we have it) is indeed loaded and run. This involves the use of verified hardware and libraries for crypto algorithms. Such libraries have already formally been verified in the Verisoft project [11].

## 6 Extending the Serial Language Stack to Multi-core Computations

Language stacks deal with languages at various levels and translations between them. A 'basic' language stack for multi-core computations consists simply of i) a specification of some version of 'structured parallel C', ii) a compiler from this version of parallel C to the ISA of a multi-core machine and iii) a correctness proof showing simulation between the source program and the target program. Definitions of versions of structured parallel C (or intermediate languages close to it) and correctnes proofs for their compilers proceed in the flowing way:

- one starts with a small-step semantics of the serial version of the high level language; configurations of such semantics have program rests/continuations, stacks, and global memory. Configurations for parallel C are easily defined: keep program rests and stack local for each thread; share the global variables among threads. Computations for unstructured parallel C are equally easy to define: interleave steps of the small steps semantics of the individual threads in an arbitrary order.
- compiling unstructured parallel C to multi-core machines tends to produce very inefficient code. Thus one structures the computation by restricting accesses to memory with an ownership discipline very similar to the one of [8]. Different versions of parallel C differ essentially by the ownership discipline used. As a directive for the compiler, variables which are allowed to be unowned and shared (such that they can e.g. be used for locks) are declared as volatile. Accesses to volatile variables constitute *I/O-points* of the computation.
- Compilers do not optimize over I/O-points, thus I/O-points are consistency points. Except for accesses to volatile variables, threads are simply compiled by serial compilers. Code produced for volatile accesses has two portions: i) possibly a fence instruction draining the local store buffer; clearly this is only necessary if the target machine has store buffers, and ii) an appropriate atomic ISA instruction.
- Compiler correctness is then argued in the following steps: i) The compiled code obeys the ownership discipline of the target language in such a way that volatile accesses of the compiled code correspond to volatile accesses of the source code, i.e. I/O points are preserved. Then one proves both for the source language and the target language that, due to the ownership discipline, memory accesses between

I/O points are to local, owned, or shared-read-only addresses only. This implies at both language levels an order reduction theorem restricting interleavings to occur at I/O points only. We call such an interleaving an *I/O-block schedule*. iii) One concludes simulation between source code and target code using the fact that I/O points are compiler consistency points and thus in each thread compiler consistency is maintained by the serial (!) computations between I/O-points.

### 6.1 Related Work

The 'verified software toolchain' project [39] presently deals with a 'basis' language stack. C minor is used as serial source language. The serial compiler is the formally verified optimizing compiler from the CompCert project [28]. Permissions on memory are modified by operations on locks – this can be seen as a kind of ownersip discipline. The target machine has sequentially consistent shared memory in the present work; draining store buffers is identified as an issue for future work. Proofs are formalized in Coq. In the proofs the permission status of variables is maintained in the annotation language. We will return to this project in the next section.

### 6.2 Extending the Language Stack

A crucial result for the extension of a language stack for 'C + macro assembly + ISA-sp + devices' to the multi-core world is a general order reduction theorem that allows to restrict interleavings to I/O-block schedules for programs obeying the ownership discipline, even if the changes of language level occur in a single thread of a concurrent program. Besides the volatile memory accesses, this requires the introduction of additional I/O points: i) at the first step in hypervisor mode ($ASID = 0$) after a switch from guest mode ($ASID \neq 0$) because we need compiler consistency there, and ii) at any step in guest mode because guest computation is in ISA-sp and we no not want to restrict interleavings there. An appropriate general reorder theorem is reported in [40]. Application of the theorem to justify correctness of compilation across the language stack for a version of parallel C-IL without dirty bits and a corresponding simple handling of store buffers is reported in [23].

### 6.3 Future Work

The same reorder theorem should allow to establish correctness of compilation across the language stack for a structured parallel C-IL with dirty bits down to ISA-u with dirty bits. However, in order to justify that the resulting program is simulated in ISA-sp with store buffers one would need a version of the Cohen-Schirmer theorem for machines with MMUs.

The language stack we have introduced so far appears to establish semantics and correctness of compilation for the complete code of modern hypervisors, provided shadow pages tables (which we introduced in the introduction) are not shared between processors. This restriction is not terribly severe, because modern processors tend more and more to provide hardware support for two levels of translations, which renders shadow

page tables unnecessary in the first place. As translations used by different processors are often identical, one can save space for shadow page tables by sharing them among processors. This permits the implemention of larger shadow page tables leading to fewer page faults and hence to increased performance. We observe that this introduces an interesting situation in the combined language stack: the shadow page tables are now a C data structure that is accessed concurrently by C programs in hypervisor mode *and* the MMUs of other processors running in guest mode. Recall that MMUs set accessed and dirty bits; thus both MMU und C program can read *and* write. Now interleaving of MMU steps and hypervisor steps must be restricted. One makes shadow page tables volatile and reorders MMU accesses of other MMUs immediately after the volatile writes of the hypervisor. To justify this, one has to argue that the MMUs of other MMUs running in guest mode never access data structures other than shadow page tables; with the modelling of the MMU as an explicit piece of concurrent code, this proof becomes part of ordinary program verification.

## 7 Soundness of VCC and Its Use

Formal verification with unsound tools and methods is meaningless. In the context of proving the correctness of a hypervisor using VCC as a proof tool, two soundness arguments are obviously called for: i) a proof that VCC is sound for arguing about pure structured parallel C. ii) a method to 'abuse' VCC to argue about machine components that are not visible in C together with a soundness proof for this method.

### 7.1 Related Work

In the Verisoft project, the basic tool for proving program code correct was a verification condition generator for C0 whose proof obligations were discharged using the interactive theorem prover Isabell-HOL. The soundness of the verification condition generator for C0 was established in a formal proof [41]. The proof tool was extended to handle 'external variables' and 'external functions' manipulating these variables. Components of configurations not visible in the C0 configuration of kernels (processor registers, configurations of user processes, and device state) were coded in these external variables. The proof technology is described in great detail in [38].

A formal soundness proof for a program analysis tool for structured parallel C is developed in the 'verified software toolchain' project [39].

### 7.2 Soundness of VCC

An obvious prerequisite for a soundness proof of VCC is a complete specification of VCC's annotation language and its semantics. VCC's annotations have two parts: i) a very rich language extension for ghost code, where ghost instructions manipulate both ghost variables and ghost fields which are added to records of the original implementation language, and ii) a rich assertion language referring to both implementation and ghost data. A complete definition of 'C-IL + ghost' can be found in [22] together with a proof that 'C-IL + ghost' is simulated by C-IL provided ghost code always terminates.

The VCC assertion language is documented informally in a reference manual and a tutorial [42]; the reference manual also has some rudimentary mathematical underpinnings. More of these underpinnings are described in various articles [43–45]. However, there is currently no single complete mathematical definition. Thus establishing the soundness of VCC still requires considerable work (see subsection 7.5).

### 7.3 Using VCC for Languages Other Than C

As C is a universal programming language, one can use C verifiers to prove the correctness of programs in any other programming language $L$ by i) writing in C a (sound) simulator for programs in $L$, followed by ii) arguing in VCC about the simulated programs, and iii) proving property transfer from VCC results to results over the original code given in language $L$. Extending 'C + macro assembly + assembly' programs with a simulator for program portions not written in C then allows to argue in VCC about such programs. A VCC extension for x64 assembly code is described in [46, 47] and was used to verify the 14K lines of macro assembly code of the Hyper-V hypervisor. In the tool, processor registers were coded in a straightforward way in a struct, a so called *hybrid* variable which serves the same role as an external variable in the Verisoft tool chain mentioned above. Coding the effect of assembly or macro assembly instructions amounts to trivial reformulation of the semantics of the instructions as C functions. Calls and returns of macro assembly functions are coded in a naive way. The extension supports gotos within a routine and function calls, but does not support more extreme forms of control flow, e.g. it cannot be used to prove the correctness of thread switch via change to the stack pointer.

Currently, however, there is a slight technical problem: VCC does currently not support hybrid variables directly. We cannot place hybrid variables in ghost memory, because information clearly flows from hybrid variables to implementation variables, and this would violates a crucial hypothesis in the simulation theorem between original and annotated program. If we place it into implementation memory, we have to guarantee that it is not reachable by address arithmetic from other variables. Fortunately, there currently is a possible workaround: physical addresses of modern processors have at most 48 bits and VCC allows up to 64 bit addresses. Thus hybrid variables can be placed in memory at addresses larger than $2^{48} - 1$. Future versions of VCC are planned to support hybrid memory as a third kind of memory (next to implementation and ghost memory) on which the use of mathematical types is allowed; in turn, the formalization of 'C-IL + ghost' should be extended to include this special hybrid memory.

The papers [31, 48] show the soundness of an assembler verification approach in the spirit of Vx86 relative to the mixed 'C-IL + macro assembly' language semantics of our language stack.

### 7.4 Verifying Device Drivers with VCC

One way to reason about device drivers is to use techniques from concurrent program reasoning. In a concurrent program, one can rarely specify a function on shared state via a pre and post condition on the state of the device, since other concurrent operations

may overlap the execution of the function. Instead, one can specify the function as a linearizable operation that appears to take place atomically at some point between invocation of the function and its return. In VCC, the usual idiom for such verification is to introduce a ghost data object representing the abstract state provided by the device in combination with the driver; the state of this object is coupled to the concrete states of the driver and the device via a coupling invariant. Associated with a function call is a ghost object representing the operation being performed; this operation includes a boolean field indicating whether the operation has completed; an invariant of the operation is that in any step in which this field changes, it goes from false to true and the abstract state of the device changes according to the semantics of the operation. The abstract device has a field that indicates which operation (if any) is "executing" in any particular step, and has an invariant that the abstract state of the device changes only according to the invariant of the operation being performed (which might also be an operation external to the system).

However, another current limitation of VCC is that it allows ordinary C memory operations only on locations that act like memory. This means that it cannot directly encode devices where writing or reading a *memory mapped I/O* (MMIO) address has an immediate side effect on the device state; currently, the semantics of such operations have to be captured via intrinsics.

### 7.5 Future Work

A proof of the soundness of VCC apparently still requires the following three major steps:

- documenting the assertion language. This language is rich and comprises i) the usual assertions for serial code, ii) an ownership calculus for objects which is used in place of separation logic to establish frame properties, and iii) a nontrivial amount of constructs supporting arguments about concurrency.
- documenting the assertions which are automatically generated by VCC in order to i) guarantee the termination of ghost code, and ii) enforce an ownership discipline on the variables and a flushing strategy for store buffers.
- proving the soundness of VCC by showing i) ghost code of verified programs terminates; thus we have simulation between the annotated program and the implementation code, ii) assertions proven in VCC hold in the parallel C-IL semantics; this is the part of the soundness proof one expects from classical theory (this portion of the soundness proofs for VCC should work along the lines of soundness proofs for rely/guarantee logics – a proof outline is given in the VCC manual [42]), and iii) variables of verified programs obey ownership discipline and code of translated programs obeys a flushing discipline for store buffers; this guarantees correct translation to the ISA-sp level of the multi-core machine.

## 8 Hypervisor Correctness

Figure 7 gives a very high-level overview of the structure of the overall theory. After establishing the model stack and the soundness of proof tools and their application, what

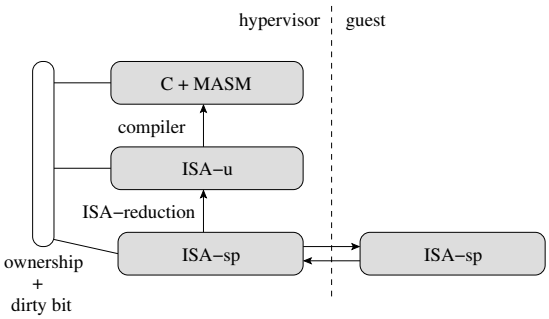



**Fig. 6.** Using ownership and dirty bit conditions to achieve simulation in a hypervisor model stack by propagating ownership downwards

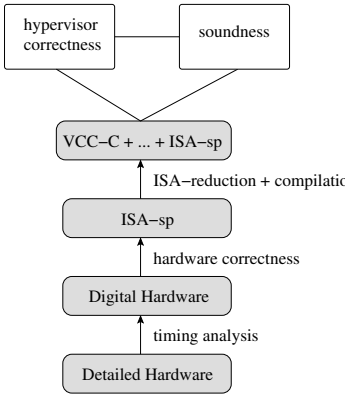



**Fig. 7.** High-level overview of the model stack and main theorems

is left to do is the actual work of program verification. Thus we are left in this survey paper with the task to outline the proof of a hypervisor correctness theorem which expresses virtualization of several ISA-sp machines enriched with system calls stated on 'C + macro assembly + ISA-u + ISA-sp'. Fortunately we can build on substantial technology from other kernel verification projects.

### 8.1 Related Work

The well known 'seL4' project [49] succeeded in formally verifying the C portion of an industrial microkernel comprising about 9000 lines of code (LOC), although that verification ignored a number of important hardware issues such as the MMU and devices, and used a rather unrealistic approach to interrupt handling, largely because that verification was based entirely on sequential program reasoning. In the Verisoft project [11] both the C portion and the assembly portion of two kernels was formally verified: i) the code of a small real real time kernel called OLOS comprising about 450 LOC [50] and ii) the code of a general purpose kernel called VAMOS of about 2500 LOC [51, 52]. In that project the verification of C portions and assembly portions was decoupled [29] in the following way: A generic concurrent model for kernels and their user processes called CVM (for 'communicating virtual machines') was introduced, where a so called 'abstract kernel' written in C communicates with a certain number of virtual machines $vm(u)$ (see Figure 8) programmed in ISA. At any time either the abstract kernel or a user process $vm(u)$ is running. The abstract kernel uses a small number of external functions called 'CVM primitives' which realize communication between the kernel, user processes and devices. The semantics of these user processes is entirely specified in the concurrent CVM model. To obtain the complete kernel implementation, the abstract kernel is linked with a few new functions and data structures, essentially process control blocks, page tables and a page fault handler in case the kernel supports

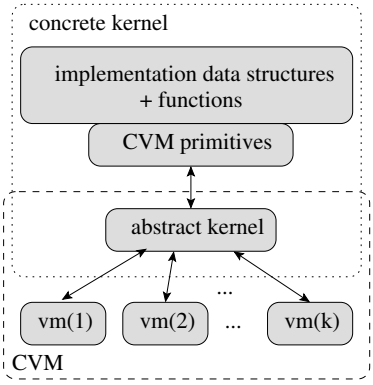



**Fig. 8.** The CVM kernel model

demand paging (e.g. like VAMOS does); CVM primitives are implemented in assembly language. The resulting kernel in called the 'concrete kernel'. Correctness theorems state that the CVM model is simulated in ISA by the compiled concrete kernel together with the user machines running in translated mode. Since the C portions of seL4 are already formally verified, one should be able to obtain a similar overall correctness result by declaring appropriate parts of seL4's C implementation as abstract kernel without too much extra effort.

### 8.2 Hypervisor Verification in VCC

That VCC allows to verify the implementations of locks has been demonstrated in [53]. Partial results concerning concurrent C programs and their interrupt handlers are reported in [54]. Program threads and their handlers are treated like different threads and only the C portions of the programs are considered; APICs and the mechanism for delivery of *inter processor interrupts* (IPIs) are not modeled. Thus the treatment of interrupts is still quite incomplete. The full formal verification of a small hypervisor written in 'C + macro assembly + assembly' in VCC using the serial language stack of Section 4 (which is also illustrated in Figure 6) and the proof technology described in Subsection 7.3 is reported in [31, 48]. The formal verification of shadow page table algorithms without sharing of shadow page tables between processors is reported in [23, 55].

### 8.3 Future Work

The following problems still have to be solved:

- Adding features to VCC that allow memory mapped devices to be triggered by reading or writing to an address that already has a value identical to the data written.
- Proving the correctness of a 'kernel layer' of a hypervisor. In order to provide guests with more virtual processors than the number $np$ of physical processors of the host,

one splits the hypervisor in a kernel layer and a virtualization layer. The kernel layer simulates large numbers $n$ of 'C + macro assembly + ISA-sp' threads by $np$ such threads. Implementation of thread switch is very similar to the switching of guests or of user processes. A data structure called thread control block (TCB) takes the role of process control block. Correctness proofs should be analogous to kernel correctness proofs but hinge on the full power of the semantics stack.

- The theory of interrupt handling in concurrent C programs and its application in VCC has to be worked out. The conditions under which an interrupt handler can be treated as an extra thread needs to be worked out. This requires to refine ownership between program threads and their interrupt handlers. For reorder theorems, the start and return of handler threads has to become an I/O-point. Finally, for liveness proofs, the delivery of IPI's (and the liveness of this mechanism) has to be included in the concurrent language stack and the VCC proofs.
- Proving actual hypervisor correctness by showing that the virtualization layer (which possibly uses shadow page tables depending on the underlying processor) on top of the kernel layer simulates an abstract hypervisor together with a number of guest machines and their user processes. Large portions of this proof should work along the lines of the kernel correctness proofs of the Verisoft project. New proofs will be needed when one argues about the state of machine components that cannot explicitly be saved at a context switch. Store buffers of sleeping guests should be empty, but both caches and TLBs of sleeping processors may contain nontrivial data, some or all of which might be flushed during the run of other guests.

## 9 Conclusion

Looking at the last section, we see that i) the feasibility of formal correctness proofs for industrial kernels has already been demonstrated and that ii) correctness proofs for hypervisors are not that much more complex, provided an appropriate basis of mixed language semantics and proof technology has been established. It is true that we have spent 6 of the last 7 chapters of this paper for outlining a paper theory of this basis. But this basis seems to be general enough to work for a large variety of hypervisor constructions such that, for individual verification projects, 'only' the proofs outlined in section 8 need to be worked out.

## References

1. Verisoft Consortium: The Verisoft-XT Project (2007-2010), http://www.verisoftxt.de/
2. Leinenbach, D., Santen, T.: Verifying the Microsoft Hyper-V Hypervisor with VCC. In: Cavalcanti, A., Dams, D. (eds.) FM 2009. LNCS, vol. 5850, pp. 806–809. Springer, Heidelberg (2009)
3. Hartmanis, J., Stearns, R.E.: On the Computational Complexity of Algorithms. Transactions of the American Mathematical Society 117, 285–306 (1965)
4. MIPS Technologies 1225 Charleston Road, Mountain View, CA: MIPS32 Architecture for Programmers Volume II: The MIPS32 Instruction Set, 2.5 edn. (2005)

5. Freescale semiconductor: Programming Environments Manual for 32-Bit Implementations of the PowerPC™ Architecture (2005)
6. Advanced Micro Devices: AMD64 Architecture Programmer's Manual: vol. 1-3 (2010)
7. Intel Santa Clara, CA, USA: Intel®64 and IA-32 Architectures Software Developer's Manual, vol. 1-3b (2010)
8. Cohen, E., Schirmer, B.: From Total Store Order to Sequential Consistency: A Practical Reduction Theorem. In: Kaufmann, M., Paulson, L. (eds.) ITP 2010. LNCS, vol. 6172, pp. 403–418. Springer, Heidelberg (2010)
9. Berg, C., Jacobi, C.: Formal Verification of the VAMP Floating Point Unit. In: Margaria, T., Melham, T.F. (eds.) CHARME 2001. LNCS, vol. 2144, pp. 325–339. Springer, Heidelberg (2001)
10. Beyer, S., Jacobi, C., Kröning, D., Leinenbach, D., Paul, W.: Instantiating Uninterpreted Functional Units and Memory System: Functional Verification of the VAMP. In: Geist, D., Tronci, E. (eds.) CHARME 2003. LNCS, vol. 2860, pp. 51–65. Springer, Heidelberg (2003)
11. Verisoft Consortium: The Verisoft Project (2003-2007), http://www.verisoft.de/
12. Oracle: Virtualbox x86 virtualization project, http://www.virtualbox.org
13. The Bochs open source community: The Bochs ia-32 emulator project, http://bochs.sourceforge.net
14. The Qemu open source community: Qemu processor emulator project, http://qemu.org
15. Lamport, L.: How to make a multiprocessor computer that correctly executes multiprocess programs. IEEE Trans. Comput. 28(9), 690–691 (1979)
16. Sewell, P., Sarkar, S., Owens, S., Nardelli, F.Z., Myreen, M.O.: x86-TSO: a rigorous and usable programmer's model for x86 multiprocessors. Commun. ACM 53(7), 89–97 (2010)
17. Sweazey, P., Smith, A.J.: A class of compatible cache consistency protocols and their support by the IEEE futurebus. In: Proceedings of the 13th Annual International Symposium on Computer Architecture, ISCA 1986, pp. 414–423. IEEE Computer Society Press, Los Alamitos (1986)
18. Chen, X., Yang, Y., Gopalakrishnan, G., Chou, C.T.: Efficient methods for formally verifying safety properties of hierarchical cache coherence protocols. Formal Methods in System Design 36, 37–64 (2010)
19. Paul, W.: A Pipelined Multi Core MIPS Machine - Hardware Implementation and Correctness Proof (2012), http://www-wjp.cs.uni-saarland.de/lehre/vorlesung/rechnerarchitektur2/ws1112/layouts/multicorebook.pdf
20. Degenbaev, U.: Formal Specification of the x86 Instruction Set Architecture. PhD thesis, Saarland University, Saarbrücken (2011)
21. Baumann, C.: Formal specification of the x87 floating-point instruction set. Master's thesis, Saarland University, Saarbrücken (2008)
22. Schmaltz, S.: Towards Pervasive Formal Verification of Multi-Core Operating Systems and Hypervisors Implemented in C (DRAFT). PhD thesis, Saarland University, Saarbrücken (2012)
23. Kovalev, M.: TLB Virtualization in the Context of Hypervisor Verification (DRAFT). PhD thesis, Saarland University, Saarbrücken (2012)
24. Degenbaev, U., Paul, W.J., Schirmer, N.: Pervasive Theory of Memory. In: Albers, S., Alt, H., Näher, S. (eds.) Efficient Algorithms. LNCS, vol. 5760, pp. 74–98. Springer, Heidelberg (2009)
25. Owens, S.: Reasoning about the Implementation of Concurrency Abstractions on x86-TSO. In: D'Hondt, T. (ed.) ECOOP 2010. LNCS, vol. 6183, pp. 478–503. Springer, Heidelberg (2010)

26. Schmaltz, S., Shadrin, A.: Integrated Semantics of Intermediate-Language C and Macro-Assembler for Pervasive Formal Verification of Operating Systems and Hypervisors from VerisoftXT. In: Joshi, R., Müller, P., Podelski, A. (eds.) VSTTE 2012. LNCS, vol. 7152, pp. 18–33. Springer, Heidelberg (2012)
27. Leinenbach, D.: Compiler Verification in the Context of Pervasive System Verification. PhD thesis, Saarland University, Saarbrücken (2008)
28. Leroy, X.: Formal verification of a realistic compiler. Communications of the ACM 52(7), 107–115 (2009)
29. Gargano, M., Hillebrand, M., Leinenbach, D., Paul, W.: On the correctness of operating system kernels. In: Hurd, J., Melham, T. (eds.) Theorem Proving in High Order Logics, Oxford, U.K. Springer (2005)
30. Alkassar, E.: OS Verification Extended - On the Formal Verification of Device Drivers and the Correctness of Client/Server Software. PhD thesis, Saarland University, Saarbrücken (2009)
31. Shadrin, A.: Mixed Low- and High Level Programming Languages Semantics.Automated Verification of a Small Hypervisor: Putting It All Together. PhD thesis, Saarland University, Saarbrücken (2012)
32. Ball, T., Levin, V., Rajamani, S.K.: A decade of software model checking with slam. Commun. ACM 54(7), 68–76 (2011)
33. Hillebrand, M., In der Rieden, T., Paul, W.: Dealing with i/o devices in the context of pervasive system verification. In: Proceedings of the 23rd IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD 2005), San Jose, CA, USA, October 2-5, pp. 309–316. IEEE (2005)
34. Tverdyshev, S.: Formal Verification of Gate-Level Computer Systems. PhD thesis, Saarland University, Computer Science Department (2009)
35. Hillebrand, M., Tverdyshev, S.: Formal Verification of Gate-Level Computer Systems. In: Frid, A., Morozov, A., Rybalchenko, A., Wagner, K.W. (eds.) CSR 2009. LNCS, vol. 5675, pp. 322–333. Springer, Heidelberg (2009)
36. Müller, S., Paul, W.: Computer Architecture, Complexity and Correctness. Springer (2000)
37. Alkassar, E., Schirmer, N., Starostin, A.: Formal Pervasive Verification of a Paging Mechanism. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 109–123. Springer, Heidelberg (2008)
38. Alkassar, E., Hillebrand, M.A., Leinenbach, D.C., Schirmer, N.W., Starostin, A., Tsyban, A.: Balancing the load: Leveraging semantics stack for systems verification, vol. 42(2-4), pp. 389–454 (2009)
39. Appel, A.W.: Verified Software Toolchain. In: Barthe, G. (ed.) ESOP 2011. LNCS, vol. 6602, pp. 1–17. Springer, Heidelberg (2011)
40. Baumann, C.: Reordering and simulation in concurrent systems. Technical report, Saarland University, Saarbrücken (2012)
41. Schirmer, N.: Verification of Sequential Imperative Programs in Isabelle/HOL. PhD thesis, Technische Universität München (2006)
42. Microsoft Research: The VCC webpage, http://vcc.codeplex.com
43. Cohen, E., Dahlweid, M., Hillebrand, M., Leinenbach, D., Moskal, M., Santen, T., Schulte, W., Tobies, S.: VCC: A Practical System for Verifying Concurrent C. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 23–42. Springer, Heidelberg (2009)
44. Cohen, E., Moskal, M., Schulte, W., Tobies, S.: Local Verification of Global Invariants in Concurrent Programs. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 480–494. Springer, Heidelberg (2010)
45. Cohen, E., Moskal, M., Tobies, S., Schulte, W.: A precise yet efficient memory model for C. Electron. Notes Theor. Comput. Sci. 254, 85–103 (2009)

46. Maus, S.: Verification of Hypervisor Subroutines written in Assembler. PhD thesis, Universität Freiburg (2011)
47. Maus, S., Moskal, M., Schulte, W.: Vx86: x86 Assembler Simulated in C Powered by Automated Theorem Proving. In: Meseguer, J., Roşu, G. (eds.) AMAST 2008. LNCS, vol. 5140, pp. 284–298. Springer, Heidelberg (2008)
48. Paul, W., Schmaltz, S., Shadrin, A.: Completing the Automated Verification of a Small Hypervisor – Assembler Code Verification. In: Eleftherakis, G., Hinchey, M., Holcombe, M. (eds.) SEFM 2012. LNCS, vol. 7504, pp. 188–202. Springer, Heidelberg (2012)
49. Klein, G., Andronick, J., Elphinstone, K., Heiser, G., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., Winwood, S.: seL4: Formal verification of an operating system kernel. Communications of the ACM 53(6), 107–115 (2010)
50. Daum, M., Schirmer, N.W., Schmidt, M.: Implementation correctness of a real-time operating system. In: 7th IEEE International Conference on Software Engineering and Formal Methods (SEFM 2009), Hanoi, Vietnam, November 23-27, pp. 23–32. IEEE (2009)
51. Daum, M., Dörrenbächer, J., Bogan, S.: Model stack for the pervasive verification of a microkernel-based operating system. In: Beckert, B., Klein, G. (eds.) 5th International Verification Workshop (VERIFY 2008). CEUR Workshop Proceedings, vol. 372, pp. 56–70. CEUR-WS.org (2008)
52. Dörrenbächer, J.: Formal Specification and Verification of a Microkernel. PhD thesis, Saarland University, Saarbrücken (2010)
53. Hillebrand, M.A., Leinenbach, D.C.: Formal verification of a reader-writer lock implementation in C. Electron. Notes Theor. Comput. Sci. 254, 123–141 (2009)
54. Alkassar, E., Cohen, E., Hillebrand, M., Pentchev, H.: Modular specification and verification of interprocess communication. In: Formal Methods in Computer Aided Design. IEEE (2010)
55. Alkassar, E., Cohen, E., Kovalev, M., Paul, W.: Verification of TLB Virtualization Implemented in C. In: Joshi, R., Müller, P., Podelski, A. (eds.) VSTTE 2012. LNCS, vol. 7152, pp. 209–224. Springer, Heidelberg (2012)

# Software Components in Computer Assisted Living?

Frantisek Plasil and Tomas Bures

Charles University, Prague, Czech Republic

**Abstract.** Component-based software engineering has developed mature techniques for modeling software by composition of components. They facilitate modeling of many kinds of systems, ranging from enterprise systems to embedded control systems. The common denominator of these systems is that their architecture is relatively static (i. e. the systems do not significantly evolve at runtime). This is however in strong contrast to characteristics of modern ubiquitous systems that aim at assisting humans in their lives (e.g. systems for smart-transportation, smart-energy, eldercare services) and that are one of the key priorities of EU R&D programs (e.g. FP7 ICT, ITEA2, ARTEMIS). Such systems are typically open-ended and need to dynamically evolve their architecture in response to changes in the physical world. In this talk, we investigate these future systems and outline challenges and ways of addressing their development via components.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, p. 28, 2013.

# Engineering Algorithms for Large Data Sets

Peter Sanders

Karlsruhe Institute of Technology
sanders@kit.edu

**Abstract.** For many applications, the data sets to be processed grow much faster than can be handled with the traditionally available algorithms. We therefore have to come up with new, dramatically more scalable approaches. In order to do that, we have to bring together know-how from the application, from traditional algorithm theory, and on low level aspects like parallelism, memory hierarchies, energy efficiency, and fault tolerance. The methodology of algorithm engineering with its emphasis on realistic models and its cycle of design, analysis, implementation, and experimental evaluation can serve as a glue between these requirements. This paper outlines the general challenges and gives examples from my work like sorting, full text indexing, graph algorithms, and database engines.

## 1 Algorithms for Large Data Sets

Application data sets from various sources have grown much faster than the available computational resources which are still governed my Moore's law but increasingly hit physical limitations like clock frequency, energy consumption, and reliability. To name just a few applications, one can mention sensor data from particle colliders like LHC at CERN, the world wide web, sequenced genome data – ultimately from most human individuals, or GPS traces from millions and millions of smart phone users that can yield valuable information, e.g., on the current traffic situation.

Large data sets are a fascinating topic for computer science in general and for algorithmics in particular. On the one hand, the applications can have enormous effects for our daily life, on the other hand they are a big challenge for research and engineering. The main difficulty is that a successful solution has to take into account issues from three quite different area of expertise: The particular application at hand, technological challenges, and the "traditional" areas of computer science know-how. I will focus on the algorithmic aspects here which will often be particularly interesting. Figure 1 illustrates this triangle of challenges. The problem is that, traditionally, individual persons and even teams are mostly proficient in only one of these three areas. The solution of this problem will have to bridge the gaps between the areas in several different ways. We probably have to stress interdisciplinary aspect of university education and we have to integrate technological aspects into the main stream of computer science teaching and research. For example, current research in algorithmics is still predominantly

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 29–32, 2013.

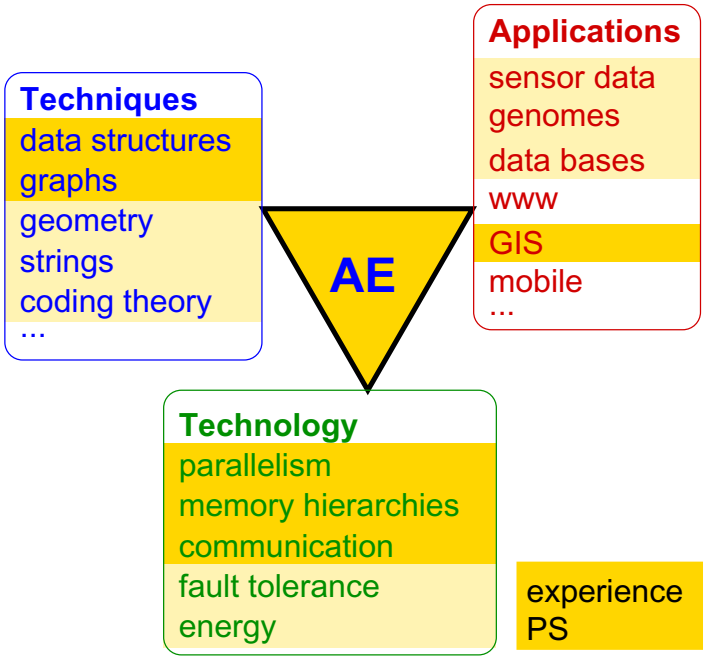



**Fig. 1.** Engineering algorithms for large data sets

using the von Neumann/RAM model with a sequential processor and homogeneous memory. In contrast, even non-large data applications need to take into account at least parallel processing and memory hierarchies if the application is in any way performance critical. Traditionalists might argue that the problems are already sufficiently challenging and difficult in the von Neumann model but obviously this is an argument from the ivory tower.

Algorithm engineering with its emphasis on realistic models and its cycle of design, analysis, implementation, and experimental evaluation can serve as a glue between these requirements. The talk will focus on examples from my work. This abstract mostly gives a few pointers to already existing papers. Much of the material presented in the talk is work in progress however including promising results in main-memory data bases, track reconstruction at CERN, genome sequencing, and phylogenetic tree reconstruction.

## 2 Examples from My Work

We have done a lot of work on sorting in some sense cumulating in a parallel external algorithm that won the sorting benchmark competition GraySort for large data sets in 2009 [9]. In contrast to previous codes, this algorithm also works for worst case inputs. This benchmark from the data base community

asks for the fastest machine/code sorting 100 Terabytes of 100 byte records on a file system. Interestingly, a similar code worked well for the category of the most energy efficient sorter JouleSort [2].

A more complex sorting problem asks for sorting all suffixes of a string. The resulting suffix array can be used for full text search, data compression and various applications in bioinformatics. Our simple linear time algorithm [6] has the additional advantage of being parallelizable [7] and externalizable [4].

An even more basic service is management of data on disk arrays. One interesting result is that parallel disks can emulate a single high capacity, high throughput logical disk allowing parallel access by using random redundant allocation of data: Any set of $N$ requested data blocks can be retrieved from $D$ disks in just $\lceil N/D \rceil + 1$ parallel I/O steps [13] (often, even the $+1$ can be dropped). Refinements for asynchronous access [11], variable block sizes [10], fault tolerance, heterogeneity [12] and many other issues are possible.

When processing large graphs, they have to be partitioned between many processors such that the interactions (e.g., number of cut edges) are small. We are intensively working on parallel and high quality graph partitioning algorithms for large graphs. Multilevel methods are the method of choice here since they combine near linear work with high quality [8].

With modern speedup techniques [3,5] for route planning in road networks, routing in continent sized networks can be done in the submillisecond range on a server and still without perceptible delay on a mobile device. However, for advanced applications we would like to integrate public transportation, historical congestion information, and real time information on the traffic situation. Currently we can handle historical information modelled as piece-wise linear functions [1] and a small number of traffic jams.

**Acknowledgements.** Partially supported by Helmholtz project Large-Scale Data Management and Analysis (LSDMA). We also would like to thank the DFG for several funding sources.

## References

1. Batz, G.V., Geisberger, R., Neubauer, S., Sanders, P.: Time-Dependent Contraction Hierarchies and Approximation. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 166–177. Springer, Heidelberg (2010)
2. Beckmann, A., Meyer, Sanders, P., Singler, J.: Energy-efficient sorting using solid state disks. In: 1st International Green Computing Conference, pp. 191–202. IEEE (2010)
3. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Engineering Route Planning Algorithms. In: Lerner, J., Wagner, D., Zweig, K.A. (eds.) Algorithmics. LNCS, vol. 5515, pp. 117–139. Springer, Heidelberg (2009)
4. Dementiev, R., Kärkkäinen, J., Mehnert, J., Sanders, P.: Better external memory suffix array construction. Special issue on Alenex 2005. ACM Journal of Experimental Algorithmics 12 (2008)

5. Geisberger, R., Sanders, P., Schultes, D., Vetter, C.: Exact routing in large road networks using contraction hierarchies. Transportation Science (2012)
6. Kärkkäinen, J., Sanders, P., Burkhardt, S.: Linear work suffix array construction. Journal of the ACM 53(6), 1–19 (2006)
7. Kulla, F., Sanders, P.: Scalable Parallel Suffix Array Construction. In: Mohr, B., Träff, J.L., Worringen, J., Dongarra, J. (eds.) PVM/MPI 2006. LNCS, vol. 4192, pp. 22–29. Springer, Heidelberg (2006); Parallel Computing 33, 605–612 (2007)
8. Osipov, V., Sanders, P., Schulz, C.: Engineering Graph Partitioning Algorithms. In: Klasing, R. (ed.) SEA 2012. LNCS, vol. 7276, pp. 18–26. Springer, Heidelberg (2012)
9. Rahn, M., Sanders, P., Singler, J.: Scalable distributed-memory external sorting. In: 26th IEEE International Conference on Data Engineering, pp. 685–688 (2010)
10. Sanders, P.: Reconciling simplicity and realism in parallel disk models. Special Issue on Parallel Data Intensive Algorithms and Applications. Parallel Computing 28(5), 705–723 (2002)
11. Sanders, P.: Asynchronous scheduling of redundant disk arrays. IEEE Transactions on Computers 52(9), 1170–1184 (2003); Short version in 12th ACM Symposium on Parallel Algorithms and Architectures, pp. 89–98 (2000)
12. Sanders, P.: Algorithms for Scalable Storage Servers. In: Van Emde Boas, P., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2004. LNCS, vol. 2932, pp. 82–101. Springer, Heidelberg (2004)
13. Sanders, P., Egner, S., Korst, J.: Fast concurrent access to parallel disks. Algorithmica 35(1), 21–55 (2003); Short version in 11th SODA, pp. 849–858 (2000)

# Core Stability in Hedonic Coalition Formation

Gerhard J. Woeginger

Department of Mathematics and Computer Science
TU Eindhoven, Netherlands

**Abstract.** In many economic, social and political situations individuals carry out activities in groups (coalitions) rather than alone and on their own. Examples range from households and sport clubs to research networks, political parties and trade unions. The underlying game theoretic framework is known as coalition formation.

This survey discusses the notion of core stability in hedonic coalition formation (where each player's happiness only depends on the other members of his coalition but not on how the remaining players outside his coalition are grouped). We present the central concepts and algorithmic approaches in the area, provide many examples, and pose a number of open problems.

**Keywords:** computational social choice, computational complexity, coalition formation, hedonic game.

## 1 Introduction

In economic, social and political situations individuals often carry out activities in groups (coalitions) rather than alone and on their own. Examples range from households, families and sport clubs to research networks, political parties and trade unions. The underlying game theoretic framework is known as *coalition formation*. In *hedonic coalition formation* each player's happiness/satisfaction only depends on the other members of his coalition, but not on how the remaining players outside his coalition are grouped together. The study of coalition formation in hedonic games goes back to the seminal paper [19] of Drèze & Greenberg.

A central question in coalition formation concerns the *stability* of a system of coalitions: if there is a possibility of increasing one's happiness/satisfaction by moving to another coalition or by merging or splitting or otherwise restructuring coalitions, players will react accordingly and the system will become unstable. The social choice literature knows a wide variety of stability concepts, as for instance the core, the strict core, the Nash stable set, the individually stable set, and the contractually individually stable set. A research line initiated by Banerjee, Konishi & Sönmez [5] and by Bogomolnaia & Jackson [8] concentrates on sufficient conditions that guarantee the existence of such stable solutions. Computational complexity issues related to hedonic coalition formation have first been investigated by Ballester [4] who establishes the NP-completeness of

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 33–50, 2013.

detecting core stable, Nash stable, and individually stable partitions (under appropriately chosen encodings of the input).

This paper zooms into *core stability*, a particularly active subarea of hedonic coalition formation games. We survey the central computational questions of this rich and colorful area, and we will see that their algorithmic behavior is surprisingly diverse. The underlying main definitions are introduced in Section 2. Each of the remaining sections discusses one particular type of hedonic game, summarizes the known results on algorithms and complexity, provides examples, and also poses a number of open problems. The open problems are marked in the following way: $(*)$ marks a problem that should be doable; $(**)$ means that the problem might be difficult; $(***)$ marks a hard and outstanding problem.

## 2 Basic Definitions and First Observations

Let $N$ be a finite set of players. A *coalition* is a non-empty subset of $N$. Every player $i \in N$ ranks all the coalitions containing $i$ via his preference relation $\preceq_i$; this order relation is reflexive ($S \preceq_i S$), transitive ($S \preceq_i T$ and $T \preceq_i U$ implies $S \preceq_i U$) and complete (at least one of $S \preceq_i T$ and $T \preceq_i S$ holds), but it is not necessarily anti-symmetric (so that $S \preceq_i T$ and $T \preceq_i S$ may hold simultaneously). The underlying strict order is denoted $\prec_i$, where $S \prec_i T$ means that $S \preceq_i T$ but not $T \preceq_i S$. If $S \prec_i T$ then player $i$ *prefers* participating in $T$ to participating in $S$, and if $S \preceq_i T$ then player $i$ *weakly prefers* participating in $T$ to participating in $S$.

A *partition* $\Pi$ is simply a collection of coalitions which partitions $N$; hence every coalition in $\Pi$ is non-empty, distinct coalitions are disjoint, and the union of all coalitions equals $N$. For a partition $\Pi$ and a player $i$, we denote by $\Pi(i)$ the unique coalition in $\Pi$ containing player $i$. The following definition introduces core stability, the key concept of this paper.

**Definition 1.** *A coalition $S$* blocks *a partition $\Pi$, if every player $i \in S$ strictly prefers $\Pi(i) \prec_i S$. A partition $\Pi$ is* core stable*, if there is no blocking coalition.*

Intuitively speaking, the players in a blocking coalition would like to separate and form their own coalition, which makes the underlying partition unstable. The game is hedonic, since the satisfaction/dissatisfaction of a player only depends on the other members of his coalition, but not on the grouping of the remaining players outside his coalition.

A closely related stability notion is strict core stability. A coalition $S$ *weakly blocks* a partition $\Pi$, if every player $i \in S$ weakly prefers $\Pi(i) \preceq_i S$, and if at least one player $j \in S$ strictly prefers $\Pi(j) \prec_j S$. A partition $\Pi$ is *strictly core stable*, if it has no weakly blocking coalition. Note that a strictly core stable partition is also core stable. While our main focus in this survey is on core stability, we will from time to time also point out results on strict core stability.

**Example 2.** *Consider a situation with three players $a, b, c$ that have the following preferences over their coalitions:*

*Preferences of player a:* $ab > ac > a > abc$
*Preferences of player b:* $bc > ab > b > abc$
*Preferences of player c:* $ac > bc > c > abc$

*There are only five possible partitions of the players: the partition* $\{abc\}$ *is blocked by* $a$, $\{ab, c\}$ *is blocked by* $bc$, $\{ac, b\}$ *is blocked by* $ab$, $\{bc, c\}$ *is blocked by* $ac$, *and* $\{a, b, c\}$ *is blocked by* $ab$. *Hence there is no core stable partition, and there also is no strictly core stable partition.*

| Problem: Core-Stability (Existence) |
|---|
| Instance: A hedonic game, that consists of a set $N$ of players and their preference relations $\preceq_i$. |
| Question: Does there exist a core stable partition of $N$? |

| Problem: Core-Stability (Verification) |
|---|
| Instance: A hedonic game, that consists of a set $N$ of players and their preference relations $\preceq_i$; a partition $\Pi$ of $N$. |
| Question: Does there exist a blocking coalition $S$ for partition $\Pi$? |

**Fig. 1.** The algorithmic problems around core stability

The central algorithmic questions are of course to decide whether a given game possesses a core stable partition (existence problem) and to check whether a given partition for a given game actually is core stable (verification problem). Both problems are formally specified in Figure 1. The precise computational complexity of these two problems depends on the way the preference relations are specified in the input, and we will see a variety of natural ways in the following chapters. Throughout we only consider representations of the input for which the preference relations can be evaluated in polynomial time: given a player $i$ and two coalitions $S$ and $T$ with $i \in S$ and $i \in T$, we can decide in polynomial time whether $S \preceq_i T$. Consequently we are also able to decide in polynomial time whether a given coalition $S$ blocks a given partition $\Pi$ of a given hedonic game.

Here is a rewording of the existence problem that clearly shows the quantifiers in the underlying question:

> Does there exist a partition $\Pi$ of the players, such that every coalition $S$ satisfies the property that $S$ is not blocking for $\Pi$?

Or even shorter: $\exists \Pi\ \forall S$: $\neg(S$ blocks $\Pi)$. This reworded formulation starts with an existential quantification, followed by an existential quantification, followed

by a property that can be verified in polynomial time, and hence is a $\Sigma_2^p$-formulation; see for instance Theorem 17.8 in Papadimitriou [34].

**Observation 3.** *As the preference relations can be evaluated in polynomial time, the existence version of* Core-Stability *is contained in the complexity class $\Sigma_2^p$ and the verification version is contained in NP.* □

The verification problem $\exists S$: ($S$ blocks $\Pi$) is the negation of the inner part of the existence problem $\exists \Pi\ \forall S$: $\neg$($S$ blocks $\Pi$). This yields the following straightforward connection between the two problems.

**Observation 4.** *If the verification problem of* Core-Stability *is polynomially solvable, then the existence version of* Core-Stability *is contained in the complexity class NP.* □

There are no further close connections between existence problem and verification problem. In particular, hardness of the verification problem does not necessarily imply hardness of the existence problem: under the enemy-oriented preferences in Section 4 existence is easy whereas verification is hard. We close this section with the common ranking property of Farrell & Scotchmer [21] for hedonic games in which all players have the same opinion about their coalitions. Since such a game has little potential for disagreement, there always is a core stable partition:

**Observation 5.** *(Farrell & Scotchmer [21]) Consider a hedonic game for which there exists a function $f : 2^N \to \mathbb{R}$ such that $S \preceq_i T$ (for any player $i$ with $i \in S$ and $i \in T$) always implies $f(S) \leq f(T)$. Then this game has a core stable partition.*

*Proof.* Pick a coalition $S$ that maximizes the value of function $f$. As the players in $S$ prefer $S$ to all other coalitions, they will never participate in a blocking coalition. Repeat this step for the remaining players. □

## 3 Complexity of the Most General Variants

We already noted earlier that the computational complexity of the Core-Stability existence and verification problem does heavily depend on the representation of the input.

The *trivial encoding* of an $n$-player game presents the preference relation of every player by explicitly listing and ranking *all* the $2^{n-1}$ coalitions that contain that player. If we ignore polynomial factors, the resulting encoding length is roughly $L \approx 2^n$. As the verification problem can be solved by searching through all $2^n \approx L$ coalitions, it is polynomially solvable under the trivial encoding. The existence problem can be solved similarly by searching through all partitions of the player set. Now recall that an $n$-element set has roughly $c^{n \log n}$ different partitions (where $c$ is some real constant; see for instance De Bruijn [10]),

so that the resulting time complexity would be roughly proportional to $L^{\log \log L}$. On the one hand this time complexity is bad, as it is not polynomially bounded in the encoding length $L$. But on the other hand this time complexity is quite good, as it has a very mild and innocent sub-exponential growth rate; this makes it virtually impossible that the existence problem could be NP-complete.

**Open Problem 6.** (∗∗) *Pinpoint the computational complexity of the* Core-Stability *existence problem under the trivial encoding.*

A good starting point for this open problem might perhaps be the complexity class LOGLOGNP introduced by Papadimitriou & Yannakakis in [35]. In any case the trivial encoding is inefficient, wasteful and unwieldy, and it definitely is not the right way for encoding preference structures in the real world.

A central assumption in cooperative game theory is *individual rationality*, which says that no player should receive less than what he could get on his own. In our framework a coalition $S$ is individually rational for player $i$ iff it satisfies $\{i\} \preceq_i S$. If a coalition is not individually rational, it is blocked by some single-player set and hence can never occur in a core stable partition. In this spirit Ballester [4] suggests to specify the preference relation for player $i$ by listing only his individually rational coalitions; we call this the *individually rational encoding* of the input.

**Theorem 7.** *(Ballester [4]) Under the individually rational encoding, the* Core-Stability *verification problem is polynomially solvable and the* Core-Stability *existence problem is NP-complete.*

*Proof.* The verification problem is straightforward, as the input explicitly lists all candidates for a blocking set. By Observation 4 then the existence problem lies in NP, and it remains to establish its NP-hardness.

The NP-hardness proof is done by a reduction from the NP-complete Exact Cover by 3-Sets (XC3) problem; see Garey & Johnson [24]. An instance of XC3 consists of a ground set $X$ and a system $\mathcal{T}$ of 3-element subsets of $X$. The problem is to decide whether there exists a partition of $X$ that only uses parts from $\mathcal{T}$. We introduce for every element $x \in X$ three corresponding players $x$, $x'$ and $x''$ in a hedonic game. The preferences of these players are as follows.

- The top choices of player $x$ are the triples $T \in \mathcal{T}$ with $x \in T$; he likes all of them equally much. Strictly below these sets he (strictly) ranks the three coalitions $xx'' \succ xx' \succ x$.
- Player $x'$ only wants to be in three coalitions which he ranks $xx' \succ x'x'' \succ x'$.
- Player $x''$ only wants to be in three coalitions which he ranks $x'x'' \succ xx'' \succ x''$.

We claim that the constructed hedonic game has a core stable partition if and only if the XC3 instance has a feasible partition. (Proof of if): Use the sets in the XC3 partition together with all sets $x'x''$. The resulting partition is core stable,

since every player $x \in X$ is in one of his most preferred coalitions. (Proof of only if): If a partition for the hedonic game puts a player $x \in X$ into one of the three coalitions $xx'', xx', x$, it cannot be core stable as the three players $x$, $x'$, $x''$ are essentially in the unstable situation of Example 2. Hence every player $x$ must be placed into a group from $\mathcal{T}$, and a core stable partition induces a partition of $X$ with all parts from $\mathcal{T}$. □

Ballester [4] also extends his NP-completeness result to the case where all preference relations are strict; as a side result this yields NP-completeness of deciding the existence of a strictly core stable partition under the individually rational encoding. Next we turn to so-called additive hedonic games, which form a common generalization of many other hedonic games that will be discussed lateron in this survey.

**Definition 8.** *A hedonic game is* additive*, if every player $i \in N$ has a real-valued preference function $v_i : N \to \mathbb{R}$ so that $S \preceq_i T$ holds if and only if $\sum_{j \in S} v_i(j) \leq \sum_{j \in T} v_i(j)$.*

In other words, in an additive hedonic game every player has a value for every other player, and the value of a coalition is simply the overall value of its members (from the view point of player $i$); hence every player can easily evaluate his profit from participating in a certain coalition. Additive preference structures allow a particularly natural and succinct representation, as they can be specified by $n^2$ numbers for $n$ players. Furthermore, additive preferences satisfy a number of desirable axiomatic properties; see Barberà, Bossert & Pattanaik [6].

Sung & Dimitrov [40] show that Core-Stability verification in additive hedonic games is strongly NP-complete; this also follows from Theorem 15. It took more time to fully understand the complexity of the Core-Stability existence problem for the additive case. On the positive side, Observation 4 tells us that the problem is contained in $\Sigma_2^p$. On the negative side, Sung & Dimitrov [41] proved it to be NP-hard, and later Aziz, Brandt & Seedig [3] extended the NP-hardness argument even to the symmetric case where $v_i(j) = v_j(i)$ holds for all players $i, j \in N$. Finally Woeginger [44] settled the problem by showing that is encapsulates the full difficulty of $\Sigma_2^p$.

**Theorem 9.** *(Woeginger [44]) In additive hedonic games, the* Core-Stability *existence problem is $\Sigma_2^p$-complete.* □

Next, let us turn to *strictly* core stable partitions in additive hedonic games. The arguments of Sung & Dimitrov [40] imply the NP-completeness of the verification question for this scenario. Sung & Dimitrov [41] prove NP-hardness of the existence question, but it seems very unlikely to me that this problem could actually be contained in NP.

**Open Problem 10.** (∗) *Establish $\Sigma_2^p$-completeness of deciding whether a given additive hedonic game has a* strictly *core stable partition.*

## 4 Preference Structures from Graphs

Dimitrov, Borm, Hendrickx & Sung [18] study hedonic games where each player views every other player either as a friend or as an enemy: the players form the vertices of a directed graph $G = (N, A)$, an arc $(x, y)$ from player $x$ to player $y$ means that $x$ considers $y$ a friend, and the absence of such an arc means that $x$ considers $y$ an enemy. We stress that here friendship is not a symmetric relation. For a player $x \in N$, we denote by $F_x$ the set of his friends and by $E_x$ the set of his enemies. Dimitrov & al [18] introduce two concrete preference structures that we will dub *friend-oriented* and *enemy-oriented*, respectively.

**Definition 11.** *Under friend-oriented preferences, player $x$ prefers coalition $T$ to coalition $S$ (that is, $S \preceq_x T$ with $x \in S$ and $x \in T$)*

- *if $|S \cap F_x| < |T \cap F_x|$, or*
- *if $|S \cap F_x| = |T \cap F_x|$ and $|S \cap E_x| \geq |T \cap E_x|$.*

*Under enemy-oriented preferences, player $x$ prefers coalition $T$ to coalition $S$*

- *if $|S \cap E_x| > |T \cap E_x|$, or*
- *if $|S \cap E_x| = |T \cap E_x|$ and $|S \cap F_x| \leq |T \cap F_x|$.*

Note that friend-oriented and enemy-oriented preferences both form special cases of additive preferences: in the friend-oriented case, we set $v_x(y) = |N|$ if $x$ considers $y$ a friend and $v_x(y) = -1$ otherwise; in the enemy-oriented case, we set $v_x(y) = 1$ if $x$ considers $y$ a friend and $v_x(y) = -|N|$ otherwise. Although the definitions of these two preference structures are symmetric to each other, the two resulting classes of hedonic games do behave very differently from each other. Let us start our discussion with the friend-oriented scenario.

**Theorem 12.** *(Dimitrov & al [18]) Under friend-oriented preferences, there always exists a core stable partition.*

*Proof.* We use the strongly connected components $C_1, \ldots, C_k$ of the directed graph as partition. Suppose for the sake of contradiction that there exists a blocking coalition $S$. Without much loss of generality we assume that $S$ intersects every component $C_i$ (as the non-intersecting components can be safely ignored). There exists a sink component $C_j$ without arcs to other components, and we distinguish two cases. If $C_j \not\subseteq S$, then one of the vertices in $S \cap C_j$ has an arc into $C_j - S$; hence this vertex has fewer friends in $S$ than in $C_i$, and $S$ is not blocking. If $C_j \subseteq S$, then every vertex in $C_j$ has the same number of friends in $C_j$ and in $S$, but strictly more enemies in $S$; hence $S$ is not blocking. □

(A closer look at this proof shows that the strongly connected components actually form a *strictly* core stable partition.) It is easy to see that in a core stable partition every coalition must be strongly connected, but there also exist examples with core stable partitions where every coalition is a proper subset of a strongly connected component: assume that players $A_1$ and $A_2$ are mutual

friends, that $B_1$ and $B_2$ are mutual friends, that $A_1$ is friendly towards $B_1$, and that $B_2$ is friendly towards $A_2$. Then the partition $\{A_1, A_2\}$ and $\{B_1, B_2\}$ is core stable.

**Open Problem 13.** $(*)$ *Is there a polynomial time algorithm for the* Core-Stability *verification problem under friend-oriented preferences?*

We turn to the enemy-oriented scenario, and we start with a crucial observation: if player $x$ considers player $y$ an enemy, then a core stable partition cannot place $x$ and $y$ together into the same coalition. Such a coalition would violate individual rationality, as player $x$ would rather stay alone than be together with $y$. Consequently under the enemy-oriented scenario only mutual (symmetric) friendship relations matter, and from now on we will assume that the underlying friendship graph $G$ actually is undirected. Note furthermore that in a core stable partition every coalition induces a clique in $G$.

**Theorem 14.** *(Dimitrov & al [18]) Under enemy-oriented preferences, there always exists a core stable partition.*

*Proof.* The game satisfies the common ranking property in Observation 5: set $f(S) = |S|$ if $S$ induces a clique and $f(S) = 0$ otherwise. □

Superficially, the results for friend-oriented preferences in Theorem 12 and for enemy-oriented preferences in Theorem 14 have a very similar smell. But the two problems differ a lot, if one actually wants to find such a core stable partition. As the strongly connected components of a directed graphs can be determined in polynomial time (see for instance Cormen & al. [15]), in the friend-oriented scenario core stable partitions are easy to find. On the other hand maximum cliques are NP-hard to find (Garey & Johnson [24]), and every core stable partition in the enemy-oriented scenario must contain such a maximum clique; hence in the enemy-oriented scenario core stable partitions are hard to find.

**Theorem 15.** *(Sung & Dimitrov [40]) Under enemy-oriented preferences, the* Core-Stability *verification problem is strongly NP-complete.*

*Proof.* By Observation 3 the verification problem lies in NP. NP-hardness is shown by a reduction from the NP-complete Maximum Clique problem; see Garey & Johnson [24]. An instance of the clique problem consists of an undirected graph $G' = (V', E')$ and an integer bound $k$. The problem is to decide whether the graph contains a clique on $k$ vertices.

We define a new graph $G = (V, E)$ by adding vertices and edges to $G'$: for every vertex $v \in V'$, we create $k-2$ new vertices that together with $v$ form a $(k-1)$-clique. Finally we define the partition $\Pi$ of $V$ whose parts are exactly the vertex sets of these $(k-1)$-cliques. We claim that in the constructed hedonic game for $G$ there is a blocking set for partition $\Pi$ if and only if the graph $G'$ has a clique of size $k$. (Proof of if): The clique of size $k$ forms a blocking set for $\Pi$.

(Proof of only if): If $G'$ has no clique of size $k$, the largest clique in graph $G$ has size $k-1$. Hence $\Pi$ assigns every player to his most preferred coalition. □

Next let us discuss *strictly* core stable partitions. The proof of Theorem 15 also implies NP-completeness of the strict verification problem. The strict existence problem seems to be fairly messy, and I would not be surprised if it turns out to be $\Sigma_2^p$-complete; note for instance that the path $P_n$ on $n \geq 2$ vertices and the cycle $C_n$ on $n \geq 4$ vertices allow a strictly core stable partition if and only if $n$ is even.

**Open Problem 16.** (∗) *Pinpoint the computational complexity of deciding whether a given hedonic game with enemy-oriented preferences has a* strictly *core stable partition.*

Here is another variation. For an undirected graph $G$ and a vertex $v \in G$, we let $\omega_G(v)$ denote the size of the largest clique that contains $v$. A partition of the vertices is called *wonderfully stable*, if every vertex $v$ ends up in a coalition of size $\omega_G(v)$. In the enemy-oriented scenario, a wonderfully stable partition puts every player into his most preferred coalition.

**Open Problem 17.** (∗∗) *Pinpoint the computational complexity of deciding whether a given undirected graph has a wonderfully stable partition.*

The wonderfully stable partition problem is NP-hard, but perhaps unlikely to be contained in NP. The problem is also unlikely to be $\Sigma_2^p$-complete, as it can be solved in polynomial time with a logarithmic number of calls to an NP-oracle: the oracle algorithm first determines the value of $\sum_{v \in N} \omega_G(v)$ by a binary search; every step in this binary search costs one call to the NP-oracle; then the algorithm asks the NP-oracle whether there exists a partition into cliques that reaches this value. This places the problem into the complexity class $\Theta_2^p$ which is believed to be a proper subset of $\Sigma_2^p$; see for instance Wagner [43] or Theorem 17.7 in [34] for more information on this class.

## 5 Anonymous Preference Structures

In a hedonic game with *anonymous preferences*, every player is indifferent about coalitions of the same size. Hence a player's preferences can be concisely specified by stating his ranking of coalition sizes. A natural example for anonymous preferences is a chess club where all even group sizes should be fine, whereas odd groups sizes would prevent the players from splitting into chess-playing pairs.

**Theorem 18.** *(Ballester [4]) Under anonymous preferences, the* Core-Stability *verification problem is polynomially solvable and the* Core-Stability *existence problem is NP-complete.*

*Proof.* In the verification problem, we search for a blocking coalition $S$ by checking the possible sizes $s = |S|$ one by one. For a fixed size $s$, it is easy to decide whether there are $s$ players who would be happier in a coalition of size $s$ than in their current coalitions. Since the verification problem is polynomially solvable, Observation 4 yields that the existence problem lies in NP.

The NP-hardness proof is done by a reduction from the Exact Cover by 3-Sets (XC3) problem as defined in the proof of Theorem 7. Let $T_1, \ldots, T_m$ be an enumeration of the triples in $\mathcal{T}$, and define $\text{num}(T_k) = 4k$. Here are our players:

- For every triple $T \in \mathcal{T}$, create $\text{num}(T) - 3$ players who like the two sizes $\text{num}(T) - 3$ and $\text{num}(T)$ and hate all other sizes.
- For every $x \in X$, create a single player $P(x)$ who only likes the sizes $\text{num}(T)$ for the triples $T$ with $x \in T$ and hates all the other sizes.

It can be seen that the constructed hedonic game has a core stable partition if and only if the XC3 instance has a feasible partition. □

Note that every player in the above NP-hardness argument has a primitive black-and-white view of the world: he (equally) likes some of the sizes, and he (equally) hates the remaining ones. Darmann & al [17] show that the problem remains NP-complete even if every player (equally) likes two sizes and (equally) hates all the remaining ones.

**Open Problem 19.** (∗∗) *Consider the anonymous hedonic game where every player $i$ (equally) likes the sizes $s$ in a certain interval $a_i \leq s \leq b_i$ and (equally) hates the remaining sizes. Is this problem polynomially solvable?*

## 6 Partition into Pairs

Throughout this section we only consider coalitions of size two. Hence the preferences of a player can be specified by simply listing his ranking of the other players, and the resulting hedonic games clearly are additive. Since in this case the Core-Stability verification problem is straightforward (by searching through all pairs), we concentrate on the existence problem.

There are two basic variants that are known as the *stable matching* (or *stable marriage*) problem and as the *stable roommate* problem. The stable matching problem has a bipartite structure: there are $n$ male and $n$ female players, and the only feasible pairs are man-woman couples. The stable roommate problem has a non-bipartite structure: there are $2n$ unisex players, and every possible pair is feasible.

### 6.1 Stable Matchings

The stable matching problem was introduced in the seminal paper by Gale & Shapley [22], one of the most cited papers in computational social choice.

A matching $\mu$ pairs the men with the women; the partner of man $M$ in the matching is denoted $\mu(M)$ and the partner of woman $W$ is denoted $\mu(W)$. A man $M$ and a woman $W$ form a *blocking pair* $(M, W)$ for matching $\mu$, if $M$ prefers $W$ to his woman $\mu(M)$ and if simultaneously $W$ prefers $M$ to her man $\mu(W)$.

Perhaps the most natural approach to stable matching would be the following iterative improvement procedure: Start with an arbitrary matching, and then iteratively find a blocking pair $(M, W)$ and improve the situation by replacing the two pairs $(M, \mu(M))$ and $(\mu(W), W)$ by the new pairs $(M, W)$ and $(\mu(W), \mu(M))$. The following example demonstrates that this idea may fail horribly.

**Example 20.** *(Tamura [42]) There are four men $A, B, C, D$ and four women $W, X, Y, Z$ with the following preference lists:*

| | |
|---|---|
| $A\colon X > Z > W > Y$ | $W\colon A > C > B > D$ |
| $B\colon Y > W > X > Z$ | $X\colon B > D > C > A$ |
| $C\colon Z > X > Y > W$ | $Y\colon C > A > D > B$ |
| $D\colon W > Y > Z > X$ | $Z\colon D > B > A > C$ |

*Assume that the iterative improvement procedure picks $AW, BX, CZ, DY$ as its starting point. The following lines show the dynamics of the resulting process.*

| | | |
|---|---|---|
| 1. | $\underline{A}W, BX, C\underline{Z}, DY$ | *$A$ and $Z$ are blocking* |
| 2. | $AZ, BX, \underline{C}W, D\underline{Y}$ | *$C$ and $Y$ are blocking* |
| 3. | $AZ, \underline{B}X, CY, D\underline{W}$ | *$B$ and $W$ are blocking* |
| 4. | $A\underline{Z}, BW, CY, \underline{D}X$ | *$D$ and $Z$ are blocking* |
| 5. | $A\underline{X}, BW, \underline{C}Y, DZ$ | *$C$ and $X$ are blocking* |
| 6. | $\underline{A}Y, B\underline{W}, CX, DZ$ | *$A$ and $W$ are blocking* |
| 7. | $AW, B\underline{Y}, CX, \underline{D}Z$ | *$D$ and $Y$ are blocking* |
| 8. | $AW, \underline{B}Z, C\underline{X}, DY$ | *$B$ and $X$ are blocking* |

*The last improvement yields the matching $AW, BX, CZ, DY$, so that we are back at our starting point. The process is cycling and will never terminate!*

Let us take a closer look at the instance in Example 20. There are 24 possible matchings, five of which are actually stable. If we start the iterative improvement procedure from one of the three matchings $AY, BZ, CW, DX$ or $AW, BZ, CY, DX$ or $AY, BX, CW, DZ$, then the process will eventually reach the stable matching $AW, BX, CY, DZ$ and terminate. But if we start the iterative improvement procedure from any of the remaining 16 matchings, then the procedure will cycle and does not terminate.

Fortunately, there exist better approaches for the stable matching problem: it can be shown that there *always* exists a stable solution, which furthermore can be computed in polynomial time by the celebrated Gale-Shapley algorithm [22]. As the books by Knuth [31], Gusfield & Irving [25], and Roth & Sotomayor [37] comprehensively analyze this algorithm and extensively cover the combinatorial facets of the problem, we only formulate a summarizing theorem.

**Theorem 21.** *(Gale & Shapley [22]) If all preferences are strict, a stable matching always exists and can be found in polynomial time.* □

We now would like to spend some lines on several closely related variants. Theorem 21 assumes that every player has a strict preference ranking of the other players. Allowing *ties* in the preference relations does not change much: a stable matching always exists and can be found in polynomial time (by breaking ties arbitrarily and then applying Gale-Shapley); see for instance Irving [27].

Allowing *incomplete preference lists* (but still forbidding ties) changes the situation a little bit. Now every player can exclude some other players with whom he does not want to be matched (formally this can be done by ranking the unwanted coalitions below the coalition where he stays alone). It turns out that also for this case a stable matching always exists and can be found in polynomial time by a slight modification of the Gale-Shapley algorithm. However a stable matching is not necessarily perfect: it will consist of some pairs and of some isolated singletons. Interestingly *every* stable matching has the same set of men and women paired up and the same set of men and women as singletons; see Gale & Sotomayor [23].

Simultaneously allowing both *incomplete preference lists* and *ties* messes things up a lot. A stable matching always exists and can be found in polynomial time, but the same instance can have very different stable matchings with varying numbers of pairs. Deciding whether there is a perfect stable matching (which pairs up all the players) is NP-complete; see Manlove, Irving, Iwama, Miyazaki & Morita [32]. In fact this perfect stable matching variant is NP-complete even if the preference list of every player lists only three acceptable partners; see Irving, Manlove & O'Malley [30].

Finally Irving & Leather [28] have shown that counting the number of stable matchings (in the classical version without ties and without incomplete preference lists) is #P-complete Chebolu, Goldberg & Martin [14] indicate that even approximate counting should be difficult.

### 6.2 Stable Roommates

The stable roommate problem is the non-bipartite unisex version of the stable matching problem. The following example demonstrates that there are roommate instances without stable matching (note the structural similarity between Example 2 and Example 22).

**Example 22.** *Consider a situation with four players $A, B, C, D$ that have the following preferences: player $A$ prefers $B > C > D$; player $B$ prefers $C > A > D$; player $C$ prefers $A > B > D$; and player $D$ prefers $A > B > C$. Note that none of $A, B, C$ wants to play with the unpopular dummy player $D$.*

*The matching $\{AB, CD\}$ is blocked by $BC$, and matching $\{AC, BD\}$ is blocked by $AB$, and matching $\{BC, AD\}$ is blocked by $AC$. Hence there is no core stable partition.*

A milestone paper by Irving [26] characterizes the roommate instances with core stable matchings.

**Theorem 23.** *(Irving [26]) For the stable roommate problem with strict preferences, the existence of a stable matching can be decided in polynomial time.* □

If we allow *incomplete preference lists* (but still forbid ties), a minor modification of Irving's algorithm [26] solves the stable roommate problem in polynomial time. If we allow *ties* in the preference relations, the stable roommate problem becomes NP-complete; see Ronn [36] and Irving & Manlove [29].

Arkin, Bae, Efrat, Okamoto, Mitchell & Polishchuk [2] discuss a metric variant of the stable roommate problem where every player is a point in a metric space with distance function $|\cdot|$. Player $P$ prefers being with player $X$ to being with player $Y$ if and only if $|PX| \leq |PY|$. This special case always has a stable matching, as it satisfies the common ranking property of Observation 5: just set $f(XY) = -|XY|$ for coalitions $XY$ of size two [2].

## 7 Partition into Triples

Generalizations of the classical Gale-Shapley stable matching problem (with men and women as the two genders) to three genders (men, women, dogs) usually are very messy. Alkan [1] seems to have been the first to publish a result on this 3-gender variant, by constructing a concrete example that does not allow a core stable matching. The preferences in Alkan's example are additively separable, and there are $n = 3$ men, women and dogs. Ng & Hirschberg [33] exhibit an even smaller bad instance with $n = 2$:

**Example 24.** *(Ng & Hirschberg [33]) Consider two men $M_1, M_2$, two women $W_1, W_2$ and two dogs $D_1, D_2$ that have the following preferences over the triples:*

$$
\begin{array}{lccccccc}
M_1\colon & M_1W_1D_2 & > & M_1W_1D_1 & > & M_1W_2D_2 & > & M_1W_2D_1 \\
M_2\colon & M_2W_2D_2 & > & M_2W_1D_1 & > & M_2W_2D_1 & > & M_2W_1D_2 \\
W_1\colon & M_2W_1D_1 & > & M_1W_1D_2 & > & M_1W_1D_1 & > & M_2W_1D_2 \\
W_2\colon & M_2W_2D_1 & > & M_1W_2D_1 & > & M_2W_2D_2 & > & M_1W_2D_2 \\
D_1\colon & M_1W_2D_1 & > & M_1W_1D_1 & > & M_2W_1D_1 & > & M_2W_2D_1 \\
D_2\colon & M_1W_1D_2 & > & M_2W_2D_2 & > & M_1W_2D_2 & > & M_2W_1D_2
\end{array}
$$

*There are only four possible partitions into two disjoint triples:*

*The partition $\{M_1W_1D_1,\ M_2W_2D_2\}$ is blocked by $M_1W_1D_2$.*
*The partition $\{M_1W_1D_2,\ M_2W_2D_1\}$ is blocked by $M_2W_1D_1$.*
*The partition $\{M_1W_2D_1,\ M_2W_1D_2\}$ is blocked by $M_1W_1D_2$.*
*The partition $\{M_1W_2D_2,\ M_2W_1D_1\}$ is blocked by $M_2W_2D_2$.*

*Hence there exists no core stable matching.*

Ng & Hirschberg [33] also establish the NP-completeness of deciding the existence of a core stable matching; this result has also been derived by Subramanian [39] (independently and by a very different approach).

Donald Knuth [31] proposes the 3-gender stable matching variant with so-called *cyclic preferences*: every man $M$ has a strict ordering of the women in the instance, every woman $W$ has a strict ordering of the dogs, and every dog $D$ has a strict ordering of the men. A triple $MWD$ is blocking for a given current partition into triples, if man $M$ prefers $W$ to his currently assigned woman, if woman $W$ prefers $D$ to her currently assigned dog, and if dog $D$ prefers $M$ to its currently assigned man. Boros, Gurvich, Jaslar & Krasner [9] prove by case distinctions that every cyclic instance with $n = 3$ has a core stable matching, and Eriksson, Sjöstrand & Strimling [20] extend this positive result to $n = 4$. The approaches in [9,20] are quite technical and involve much case analysis, and they do not seem to generalize to larger values of $n$.

**Open Problem 25.** (∗∗∗) *Prove that every instance of the 3-gender stable matching problem with cyclic preferences has a stable solution.*

Biró & McDermid [7] consider the case of cyclic preferences with unacceptable partners: every man finds certain women unacceptable, every woman hates certain dogs, and every dog dislikes certain men. Under this scenario there exist instances without stable solution, and deciding the existence of a stable solution is NP-complete.

Danilov [16] discusses a related (but much easier) special case where every man primarily cares about women and where every woman primarily cares about men (and where the preferences of the dogs are arbitrary). This special case always has a stable matching. In a first step, we find a stable matching for the underlying 2-gender instance that consists of men and women. The preferences of men and women in this 2-gender instance are their primary rankings in the 3-gender instance. In the second step, we find a stable matching for the 2-gender instance with dogs on the one side and on the other side the man-woman pairs from the first step. The preferences of the dogs on man-woman pairs are copied from the 3-gender instance. The preferences of the man-woman pairs on the dogs are always fixed according to the man in the pair: the pair $MW$ prefers dog $D$ to dog $D'$, if and only if in the 3-gender instance man $M$ prefers triple $MWD$ to triple $MWD'$. Everything else follows from the Gale-Shapley Theorem 21.

In the 3-dimensional roommate problem all players have the same gender and every triple is a potential coalition. Ng & Hirschberg [33] establish NP-completeness of deciding the existence of a core stable matching in the 3-dimensional roommate problem. Arkin, Bae, Efrat, Okamoto, Mitchell & Polishchuk [2] discuss the following geometric variant of the 3-dimensional roommate problem in the Euclidean plane with distance function $|\cdot|$: every player is a point in the Euclidean plane, and player $P$ prefers triple $PX_1X_2$ to triple $PY_1Y_2$ if and only if $|PX_1| + |PX_2| < |PY_1| + |PY_2|$. (Note that here the matching problem is 3-dimensional, whereas the underlying geometric space is 2-dimensional. Note furthermore that the preference structure

is additive.) Arkin & al. [2] exhibit a highly structured instance that does not possess a core stable matching; the computational complexity of this special case however remains open.

**Open Problem 26.** (∗∗) *Settle the complexity of the Euclidean 3-dimensional roommate problem as described in the preceding paragraph.*

## 8 Preference Structures from Maxima and Minima

Cechlárová & Romero-Medina [13] investigate hedonic games where every player ranks his coalitions according to the most or least attractive member of the coalition. Similarly as in the additive games in Definition 8, every player $i \in N$ has a real-valued function $v_i : N \to \mathbb{R}$ that measures his addiction to each of the other players. For a coalition $S$, we define $v_i^{\max}(S) = \max_{j \in S} v_i(j)$ as player $i$'s addiction to the best member of $S$.

**Definition 27.** *Under max-preferences, player $i$ prefers coalition $T$ to coalition $S$ (that is, $S \preceq_i T$ with $i \in S$ and $x \in T$)*

- *if $v_i^{\max}(S) < v_i^{\max}(T)$, or*
- *if $v_i^{\max}(S) = v_i^{\max}(T)$ and $|S| \geq |T|$.*

An important special case of this scenario are max-preferences *without ties*; this means that for distinct players $a$ and $b$ the values $v_i(a)$ and $v_i(b)$ assigned to them by player $i$ are always distinct.

**Theorem 28.** *(Cechlárová & Hajduková [11]) Under max-preferences, the* Core-Stability *verification problem is polynomially solvable.*

*Proof.* How would we verify the core stablitiy of a given partition $\Pi$? The main idea is to check step by step for $k = 1, 2, \ldots, |N|$ whether there exists a blocking coalition $S$ *of size at most* $k$. For checking a concrete value $k$, we construct an auxiliary directed graph $G_k$ on the vertex set $N$; an arc $i \to j$ means that player $i$ strictly prefers every coalition $S$ with $j \in S$ and $|S| \leq k$ to his current coalition $\Pi(i)$. Formally the graph $G_k$ contains the arc $i \to j$ if $v_i^{\max}(\Pi(i)) < v_i(j)$ holds, or if $v_i^{\max}(\Pi(i)) = v_i(j)$ and $|S| > k$.

If graph $G_k$ contains a directed cycle of length at most $k$, the corresponding vertices form a blocking coalition of size at most $k$. Vice versa, a blocking coalition of size at most $k$ induces a subgraph in $G_k$ with a cycle of length at most $k$. The shortest cycle in a directed graph can be found in polynomial time; see for instance Cormen & al [15]. □

**Theorem 29.** *(Cechlárová & Hajduková [11]) Under max-preferences without ties, there always exists a core stable partition.*

*Proof.* Make every player $i$ point at the player whom he likes most. Then the underlying directed graph contains a cycle. Pick the players along such a cycle as coalition $S$, and repeat this procedure for the remaining graph. □

The algorithm in the proof of Theorem 29 is essentially the famous top-trading-cycle algorithm of David Gale for the house swapping game (see for instance [38]). On the negative side Cechlárová & Hajduková [11] prove that under max-preferences *with ties* the Core-Stability existence problem is NP-complete.

In a closely related line of research Cechlárová & Hajduková [12] investigate *min-preferences* where every player ranks his coalitions according to the *least* attractive member in the coalition. Under this scenario unstable partitions always have small blocking sets of size at most 2, so that the Core-Stability verification problem is straightforward to solve. Furthermore stable partitions always consist of small coalitions of size at most 3. For min-preferences *without ties*, Cechlárová & Hajduková [12] design a modification of Irving's roommate algorithm (Theorem 23) that solves the Core-Stability existence problem in polynomial time. They also show that for min-preferences *with ties* the existence problem is NP-complete.

## References

1. Alkan, A.: Non-existence of stable threesome matchings. Mathematical Social Sciences 16, 207–209 (1988)
2. Arkin, E.M., Bae, S.W., Efrat, A., Okamoto, K., Mitchell, J.S.B., Polishchuk, V.: Geometric stable roommates. Information Processing Letters 109, 219–224 (2009)
3. Aziz, H., Brandt, F., Seedig, H.G.: Stable partitions in additively separable hedonic games. In: Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2011), pp. 183–190 (2001)
4. Ballester, C.: NP-completeness in hedonic games. Games and Economic Behavior 49, 1–30 (2004)
5. Banerjee, S., Konishi, H., Sönmez, T.: Core in a simple coalition formation game. Social Choice and Welfare 18, 135–153 (2001)
6. Barberà, S., Bossert, W., Pattanaik, P.K.: Ranking sets of objects. In: Barberà, S., Hammond, P.J., Seidl, C. (eds.) Handbook of Utility Theory, vol. II, pp. 893–977. Kluwer Academic Publishers (2004)
7. Biró, P., McDermid, E.: Three-sided stable matchings with cyclic preferences. Algorithmica 58, 5–18 (2010)
8. Bogomolnaia, A., Jackson, M.O.: The stability of hedonic coalition structures. Games and Economic Behavior 38, 201–230 (2002)
9. Boros, E., Gurvich, V., Jaslar, S., Krasner, D.: Stable matchings in three-sided systems with cyclic preferences. Discrete Mathematics 289, 1–10 (2004)
10. de Bruijn, N.G.: Asymptotic Methods in Analysis. North-Holland, Amsterdam (1958)
11. Cechlárová, K., Hajduková, J.: Computational complexity of stable partitions with $B$-preferences. International Journal of Game Theory 31, 353–364 (2002)
12. Cechlárová, K., Hajduková, J.: Stable partitions with $W$-preferences. Discrete Applied Mathematics 138, 333–347 (2004)

13. Cechlárová, K., Romero-Medina, A.: Stability in coalition formation games. International Journal of Game Theory 29, 487–494 (2001)
14. Chebolu, P., Goldberg, L.A., Martin, R.A.: The Complexity of Approximately Counting Stable Matchings. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) APPROX and RANDOM 2010. LNCS, vol. 6302, pp. 81–94. Springer, Heidelberg (2010)
15. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press (2001)
16. Danilov, V.I.: Existence of stable matchings in some three-sided systems. Mathematical Social Sciences 46, 145–148 (2003)
17. Darmann, A., Elkind, E., Kurz, S., Lang, J., Schauer, J., Woeginger, G.: Group Activity Selection Problem. In: Goldberg, P.W., Guo, M. (eds.) WINE 2012. LNCS, vol. 7695, pp. 156–169. Springer, Heidelberg (2012)
18. Dimitrov, D., Borm, P., Hendrickx, R., Sung, S.-C.: Simple priorities and core stability in hedonic games. Social Choice and Welfare 26, 421–433 (2006)
19. Drèze, J., Greenberg, J.: Hedonic coalitions: Optimality and stability. Econometrica 48, 987–1003 (1980)
20. Eriksson, K., Sjöstrand, J., Strimling, P.: Three-dimensional stable matching with cyclic preferences. Mathematical Social Sciences 52, 77–87 (2006)
21. Farrell, J., Scotchmer, S.: Partnerships. The Quarterly Journal of Economics 103, 279–297 (1988)
22. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. American Mathematical Monthly 69, 9–15 (1962)
23. Gale, D., Sotomayor, M.A.O.: Some remarks on the stable matching problem. Discrete Applied Mathematics 11, 223–232 (1994)
24. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco (1979)
25. Gusfield, D., Irving, R.W.: The Stable Marriage Problem: Structure and Algorithms. MIT Press (1989)
26. Irving, R.W.: An efficient algorithm for the stable roommates problem. Journal of Algorithms 6, 577–595 (1985)
27. Irving, R.W.: Stable marriage and indifference. Discrete Applied Mathematics 48, 261–272 (1994)
28. Irving, R.W., Leather, P.: The complexity of counting stable marriages. SIAM Journal on Computing 15, 655–667 (1986)
29. Irving, R.W., Manlove, D.F.: The stable roommates problem with ties. Journal of Algorithms 43, 85–105 (2002)
30. Irving, R.W., Manlove, D.F., O'Malley, G.: Stable marriage with ties and bounded length preference lists. Journal of Discrete Algorithms 43, 213–219 (2009)
31. Knuth, D.E.: Mariages stables et leurs relations avec d'autres problèmes combinatoires [Stable marriage and its relation to other combinatorial problems]. CRM Proceedings and Lecture Notes, vol. 10. Les Presses de l'Université de Montréal (1997)
32. Manlove, D.F., Irving, R.W., Iwama, K., Miyazaki, S., Morita, Y.: Hard variants of stable marriage. Theoretical Computer Science 276, 261–279 (2002)
33. Ng, C., Hirschberg, D.S.: Three-dimensional stable matching problems. SIAM Journal on Discrete Mathematics 4, 245–252 (1991)
34. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley (1994)
35. Papadimitriou, C.H., Yannakakis, M.: On limited nondeterminism and the complexity of the V-C dimension. Journal of Computer and System Sciences 53, 161–170 (1996)

36. Ronn, E.: NP-complete stable matching problems. Journal of Algorithms 11, 285–304 (1990)
37. Roth, A.E., Sotomayor, M.A.O.: Two-Sided Matching. Cambridge University Press (1990)
38. Shapley, L.S., Scarf, H.: On cores and indivisibility. Journal of Mathematical Economics 1, 23–37 (1974)
39. Subramanian, A.: A new approach to stable matching problems. SIAM Journal on Computing 23, 671–701 (1994)
40. Sung, S.-C., Dimitrov, D.: On core membership testing for hedonic coalition formation games. Operations Research Letters 35, 155–158 (2007)
41. Sung, S.-C., Dimitrov, D.: Computational complexity in additive hedonic games. European Journal of Operational Research 203, 635–639 (2010)
42. Tamura, A.: Transformation from arbitrary matchings to stable matchings. Journal of Combinatorial Theory A 62, 310–323 (1993)
43. Wagner, K.: Bounded query classes. SIAM Journal on Computing 19, 833–846 (1990)
44. Woeginger, G.J.: A hardness result for core stability in additive hedonic games. Mathematical Social Sciences (2013)

# Software Production: A New Paradigm for Software Engineering Research

Sjaak Brinkkemper

Department of Information and Computing Sciences, Utrecht University,
P.O. Box 80.089, 3508 TB, Utrecht, The Netherlands
S.Brinkkemper@uu.nl

**Abstract.** Increasingly software products are being offered in an online mode, also called software-as-a-service. Serving millions of users possibly spread over the world from a central software producing organization brings about many challenges and will require several innovations from the software engineering domain. In this keynote we will introduce the notion of software production that unifies the whole range of software development activities with the continuous operations of hosting and updating the online software products. We will present an overview of recent results in the area of software product management, software implementation, software operation knowledge, and software ecosystems that make up some of the research areas of software production. This overview is accompanied with a series of intriguing challenges for the research community.

**Keywords:** software production, software-as-a-service, software product management, software operation knowledge, software ecosystems.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, p. 51, 2013.

# A Model of the Commit Size Distribution of Open Source

Carsten Kolassa[1], Dirk Riehle[2], and Michel A. Salim[2]

[1] RWTH Aachen
Carsten@Kolassa.de
[2] Friedrich-Alexander-University Erlangen-Nürnberg
dirk@riehle.org, michel@sylvestre.me

**Abstract.** A fundamental unit of work in programming is the code contribution ("commit") that a developer makes to the code base of the project in work. We use statistical methods to derive a model of the probabilistic distribution of commit sizes in open source projects and we show that the model is applicable to different project sizes. We use both graphical as well as statistical methods to validate the goodness of fit of our model. By measuring and modeling a fundamental dimension of programming we help improve software development tools and our understanding of software development.

## 1 Introduction

Free/libre/open source software (FLOSS) has been adopted widely by industry in recent years. In 2008, 85% of all enterprises were using open source software [8]. A 2010 study estimated that 98% of all enterprises were using open source software [24].

Given the significance of open source, it is surprising that there are few representative statistical analyses of open source projects, and that there are no high quality models of the fundamental dimensions of programming in open source projects.

In this paper we present a model of one important dimension of programming in open source software development, the distribution of the sizes of code contributions made to open source projects. This so-called commit size distribution describes the probability that a given commit is of a particular size.

A commit is an individual code contribution of a developer. [14] show that lines of code are a good proxy for work spent on that code. Hence, a commit is a basic unit of work performed by a developer.

The commit size distribution is therefore a model of fundamental units of work performed in open source programming.

Understanding the work performed in open source programming is helpful for building better software development tools and understanding software development in general. Moreover, case studies suggest that open source is similar to closed source in terms of growth, project complexity or modularity [19].

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 52–66, 2013.

Thus the results of this paper are likely to apply not only to open source but to closed source as well.

The contributions of this paper are:

1. A high quality analytical model of the commit size distribution of open source.
2. An in-depth validation of the model using appropriate statistical measures.
3. A comparison of commit size distributions of different project sizes.

The rest of the paper is organized as follows. Section 2 describes the necessary terms. Section 3 defines and analyzes the commit size distribution. Section 4 discusses the potential threats to validity and Section 5 discusses prior and related work. We consider potential extensions in Section 6, and present our conclusions in Section 7.

## 2 Commit Sizes

A software project is typically developed in multiple iterations, in a series of changes to its artifacts, for instance, code, documentation, or artwork. If a project is managed using a *version control system* (also known as a *source code management* system), these changes are organized into sets known as *commits*.

In this paper we address programming, hence we are only concerned with source code commits. We measure the commit size in terms of *lines of code* (LoC). We distinguish between source code lines, comment lines, and empty lines. We use the following definitions:

1. a source code line (SLoC) is one line of program code,
2. a comment line (CL) is a line consisting only of comments,
3. an empty line contains only whitespace, and
4. a line of code (LoC) is either a source code line or a comment line.

Measuring the size of a commit is a non-trivial task. The main tool for assessing commit sizes is the "diff" tool which tells the user which lines have been added and which lines have been removed. Unfortunately, a diff tool cannot identify with certainty whether a line was changed, because a changed line is always counted as one line removed and one line added. However, a changed line should count as one line of work, while an added and a separately removed line of code should count as two lines of work.

[4] developed an algorithm for identifying changed lines of code from added and removed lines of code. They use the Levenshtein distance algorithm which is a metric for measuring the distance between two strings. While helpful this approach has one major disadvantage: it is computationally expensive and does not scale to large amounts of source code. Because our analysis covers about 30% of all of open source code at its time, we need another approach.

We use the sample data from Canfora et al. to derive a simple function for estimating diff chunk sizes where the diff chunk size is a function of two variables: lines of code added and lines of code removed. This function provides a

$$lower\_bound(a,r) = max(a,r)$$
full overlap, highest number of changed lines (1)

$$upper\_bound(a,r) = a + r$$
no overlap, no changed lines in diff chunk (2)

$$diff\_chunk\_size(a,r) = \frac{(lower\_bound(a,r) + upper\_bound(a,r))}{2}$$
mean value of lower and upper bound (3)

**Fig. 1.** Equations used to compute a commit's size from input lines added and removed

statistically valid estimate for the size of a given diff output as shown in [13]. However, our evaluation based on Canfora's sample data revealed that the regression performs only trivially better than estimating the diff chunk size by taking the mean of the minimum possible and the maximum possible sizes. Thus, a more plausible and unbiased algorithm for estimating the statistically expected value is simply to take the mean of the minimum and maximum possible values.

Figure 1 provides the necessary equations. In these equations, $a$ represents the number of lines added and $r$ represents the number of lines removed according to the diff tool.

In this paper we compute commit sizes by adding up the diff chunk sizes computed using equation 3 of figure 1. A diff chunk size is the size of the diff of one file in the commit. After calculating the size of every commit in our data set we compute the commit size distribution. The commit size distribution describes the relative likelihood that a commit has a particular size. The commit size distribution of some commit population is the distribution of the probabilities (or number of occurrences) of all possible commit sizes.

## 3 Commit Size Distribution

### 3.1 Data Source and Research Method

This paper uses the database of the Ohloh.net open source project index. Our database snapshot is dated March 2008. It contains 11,143 open source projects with a total of 8,705,118 commits. [6] estimates that there were 18,000 active open source projects in September 2007 worldwide. The total number of projects is much larger, but most open source projects are not active and by our activity definition have to be excluded. We use the same definition of "active project" as Daffara: A project is active at a given point in time if the number of commits in the preceding 12 months is at least 60 % of the number of commits in the 12 months before that. Using this definition our data set contains 5,117 active open source projects. We therefore estimate that our database contains about 30 % of all open source projects considered active in March 2008.

Our analysis is descriptive: we are discovering existing characteristics in our data rather than starting off with a hypothesis and attempting to invalidate or validate it. We provide details not only of our final findings but also of the attempted distributions that did not fit. We also split our analysis along project sizes and provide the characteristics of commit size distributions by project size.

### 3.2 Measurements

We determined the total commit size distribution of our open source sample population using the definition of Section 2. In statistics a distribution can be represented as a *probability distribution function* (PDF) or a *cumulative distribution function* (CDF). The PDF in our case describes the relative likelihood that a commit of a certain size occur at a given point. The CDF can be computed by integrating the PDF. Integrating the PDF over an interval provides the probability that a commit is of the size determined by the interval boundaries. For example, integrating over the interval [1,10] provides the probability that a commit has between 1 and 10 lines of code, 1 and 10 included.

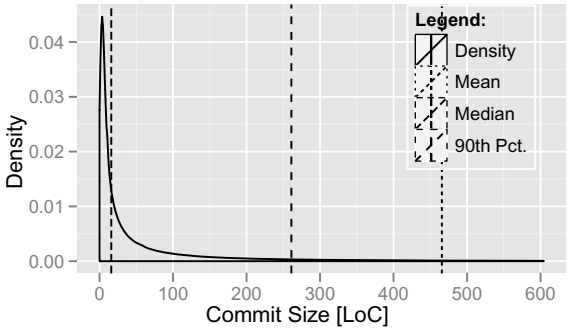



**Fig. 2.** The EPDF of the commit size distribution up to the 95th percentile of about 30% of all active open source projects (March 2008)

The empirical result of our measurements is the *empirical probability distribution function* (EPDF) as shown in Figure 2. The EPDF is a density estimation based on the observed data. It describes the probability that a certain commit has a certain commit size. The EPDF is not a closed model, it is just a representation of the observed data. The statistical key characteristics are shown in Table 1.

The form of the EPDF, in particular that it is monotonically falling suggests the following possible probability distributions:

- Biexponential
- Exponential
- Generalized Pareto

– Pareto
– Simple Power Law
– Weibull
– Zipf's

To determine the analytically closed form of the PDF and CDF we calculated the *empirical cumulative distribution function* (ECDF) and fitted the different possible distributions to the ECDF based on Newman's advice to choose the CDF for analytical purposes [17]. This approach is robust and allows us to use different regression techniques without binning. Thus we prevent the introduction of biases and information loss that comes with binning. We then fitted the different distributions (see enumeration above).

**Table 1.** Statistical key characteristics of the open source commit size distribution

| Key Parameter | Value |
|---|---|
| Mean | 465.72 |
| Median | 16 |
| 90th percentile | 261 |
| 95th percentile | 604.5 |

After reviewing the different fits and the residual plots as well as the P-P plots ("P" stands for percentile) we found that the Generalized Pareto Distribution (GPD) provides the best fit.

The GPD is broadly applicable and incorporates both exponential and Pareto distributions when certain parameters are fixed [15].

The Generalized Pareto Distribution is difficult to fit using the maximum likelihood approach, as the location parameter is unbounded (see [22] and [21]). We therefore decided to use a least square fit on the ECDF. The location parameter is chosen manually, attempting to fit the other two parameters with increasing values of location in increments of 0.5 (the granularity of commit size estimates, since they are averages of two integral values). We find that a value of 0.5 minimizes the difference between CDF and ECDF at the mode commit size of 1.

$$f(x) = \begin{cases} \frac{1}{\sigma}(1+\xi\frac{x-\theta}{\sigma})^{-1-\frac{1}{\xi}} \text{ for } \xi \neq 0 \\ \frac{1}{\sigma}exp(-\frac{(x-\theta)}{\sigma}) \text{ for } \xi = 0 \end{cases} \quad (4)$$

**Fig. 3.** PDF formula for the Generalized Pareto Distribution

The result of our fit is the CDF of the commit size distribution in closed form, which is shown next to the ECDF in Figure 5.

The parameters of the Generalized Pareto Distribution are shown in Table 2, the equations for the Generalized Pareto Distribution are shown in Figure 3

$$F(x) = \begin{cases} 1 - (1 + \frac{\xi(x-\theta)}{\sigma})^{-1/\xi} \text{ for } \xi \neq 0 \\ 1 - exp(-\frac{x-\theta}{\sigma}) \text{ for } \xi = 0 \end{cases} \quad (5)$$

**Fig. 4.** CDF formula for the Generalized Pareto Distribution

and 4. Where $\theta$ is the location parameter, it controls how much the distribution is shifted. $\sigma$ is the scale parameter it controls the dispersion of the distribution, while $\xi$ is the shape parameter which controls the shape of the generalized pareto distribution [5].

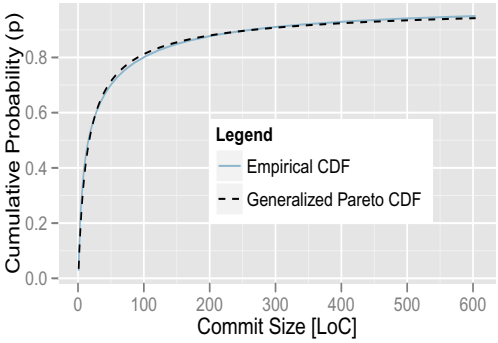



**Fig. 5.** The Generalized Pareto Distribution of the CDF up to the 95th percentile of about 30% of all active open source projects (March 2008)

**Table 2.** Model parameters of Generalized Pareto Distribution as calculated from least squares

| Parameter | Value |
|---|---|
| $\xi$ (xi) / Shape | 1.4617 |
| $\theta$ (theta) / Location | 0.5 |
| $\sigma$ (sigma) / Scale | 13.854 |

Figure 6 shows the P-P plot that compares our model to the empirical data. A P-P plot is a graphical method to compare two probability distributions by plotting their percentiles against each other; here, we compare the percentiles of our model (i.e. the CDF) to the percentiles of the empirical data (i.e. the ECDF).

As can be observed by examining the CDFs of our model and empirical data, both distributions are long-tailed; thus, per [10], the P-P plot is more appropriate than the more familiar Q-Q (quantile-quantile) plot.

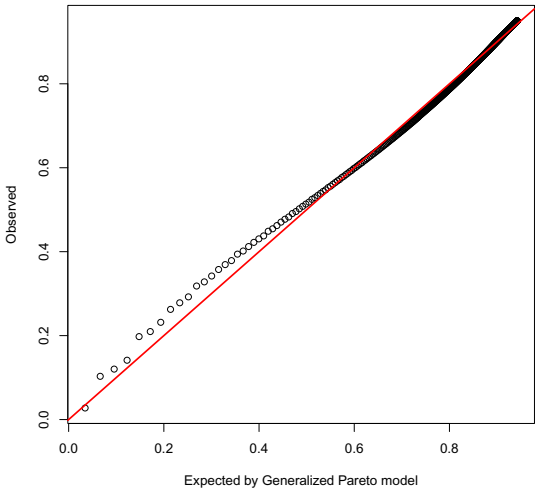



**Fig. 6.** P-P plot comparing the percentiles of the model and the empirical percentiles

To show the goodness of our fit we also compute quantitative measures such as R-square and Pearson's R as shown in Table 3.

**Table 3.** Goodness of Fit Indicators calculated up to the 95th percentile

| Parameter | Value |
|---|---|
| R-square on CDF | 0.9949 |
| Pearson's R on CDF | 0.99755 |

### 3.3 Comparison by Project Size

We also want to understand how the commit size distribution varies by project size. One might hypothesize that small projects are different from medium sized and large projects. However, we found that the GPD not only fits when analyzing all projects, it also fits to subsets of different sizes in terms of number of developers.

We classify the projects into small, medium, and large sized projects based on the number of involved developers. [3] provide an analysis of the number of developers in a random sample of projects included in the Debian GNU/Linux distribution. We use their proposed partitioning to group our projects accordingly (see Table 4).

**Table 4.** Project size boundaries

| Parameter | Minimum number of developers | Maximum number of developers |
|---|---|---|
| Small | 1 | 5 |
| Medium | 6 | 47 |
| Large | 48 | $\infty$ |

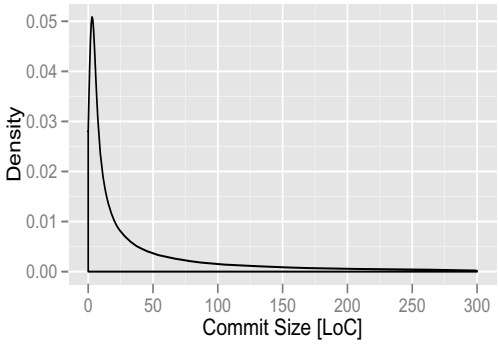



**Fig. 7.** EPDF for small projects

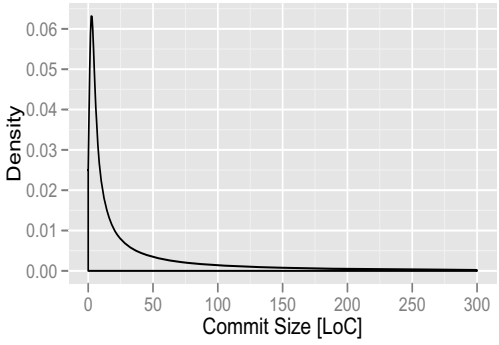



**Fig. 8.** EPDF for medium projects

We can now measure how the commit size distribution correlates with project sizes. We found that the commit size distribution of small, medium, and large projects are also best characterized as generalized Pareto distributions. Figures 10, 11, and 12 show the cumulative distribution functions respectively.

The parameters of these distributions (see Table 5) are close to the parameters of the total distribution. Figure 13 compares the EPDFs of the different subsets. After comparing the plots and the model parameters we came to the

conclusion that the location parameters is invariant to the number of developers in a project. The shape parameters have no obvious correlation with the number of developers and the differences are small, while the scale parameter falls as the number of developers increases. A possible explanation for this is that when

**Table 5.** Model parameters of the Generalized Pareto Distribution as calculated for different project sizes

| Parameter | Small projects | Medium projects | Big projects |
|---|---|---|---|
| $\xi$ (xi) / Shape | 1.5969 | 1.6008 | 1.5708 |
| $\theta$ (theta) / Location | 0.5 | 0.5 | 0.5 |
| $\sigma$ (sigma) / Scale | 14.249 | 12.199 | 10.822 |

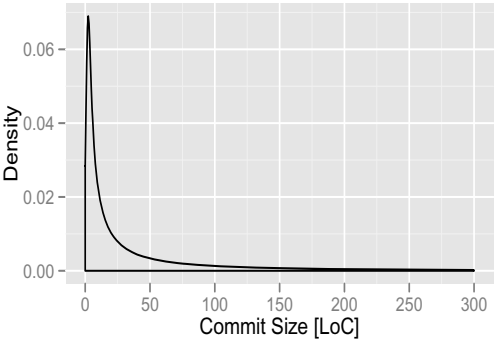



**Fig. 9.** EPDF for large projects

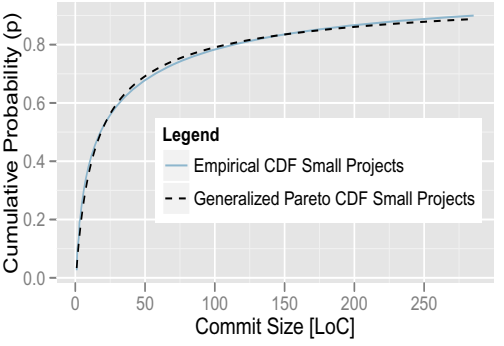



**Fig. 10.** Generalized Pareto Distribution as CDF of small open source projects (March 2008)

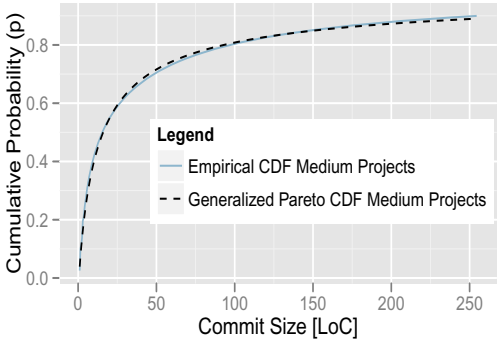



**Fig. 11.** Generalized Pareto distribution as CDF of medium open source projects (March 2008)

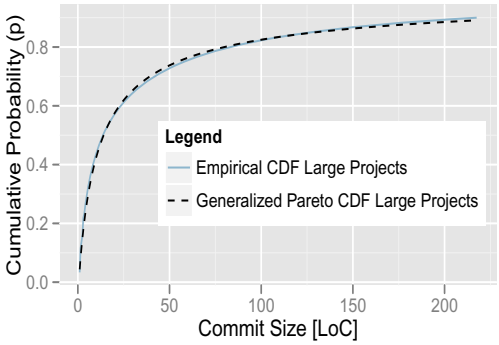



**Fig. 12.** Generalized Pareto distribution as CDF of large open source projects (March 2008)

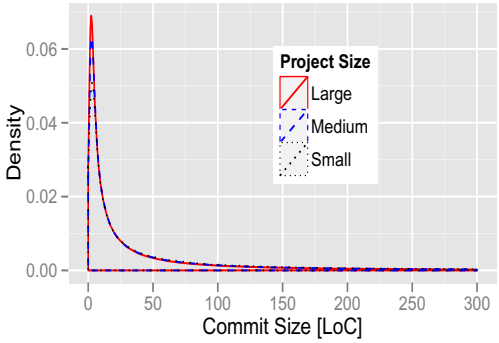



**Fig. 13.** EPDF for different project sizes for easy comparison

more and more developers join a project the average commit size goes down to prevent merge conflicts. Another explanation is the observation of [23] that small patches are more likely to be accepted then large ones; we posit that this affects larger projects more since they are more likely to have a formalized code review process.

The effect is not very big; in fact, in the region of commit sizes with the most pronounced difference (commits smaller than 13 LoC), the difference in proportion of commits in this category between large and small projects is 6.03 %. For commits in this region, both models have errors (the difference between the respective model and the empirical data) smaller than 3 %.

## 4 Threats to Validity

### 4.1 Poor Support for Multiple Enlistments

The Ohloh 2008 dataset has limited support for multiple enlistments [18]. Projects that have changed their SCM, either moving it to a new URL or transitioning to newer technology (e.g. CVS → SVN → Git or Mercurial) thus face either having their commits listed multiple times (resulting in older commits being given too much weight). But in practice this is not a problem because there is no difference in the distribution of old and new commits [7].

### 4.2 Model Errors

The slight over- and undercutting is an indication for a systematic error in our model. We cannot explain this systematic error, but the goodness of fit calculations and the P-P plot show that it is sufficiently small. That is the reason why we do not address it further. There might be a second underlying distribution that is responsible for this error but we have not been able to determine it. We also think that removing that error would come at the risk of overfitting our model.

### 4.3 Bias of the Ohloh Data

As mentioned earlier we rely on the Ohloh data for our analysis. The Ohloh data has two self-selection biases:

Projects that die very early probably never make it into the Ohloh database and are therefore underrepresented in our analysis. Projects from non-English speaking countries are also less likely to be included in Ohloh.

We think the first bias is not an issue because for such projects it would be difficult to derive a statistically significant measure as there are almost no commits yet. We also think that the language bias is unproblematic because we

do not think that there are differences in the commit size distribution whether a project is done in an English- or non-English-speaking environment.

## 5 Related Work

We previously presented a preliminary analysis of the commit size distribution [2]. Compared to this work our current analysis adds a closed model and a validation of this model as well as an analysis of the commit size distribution by project size.

[1] present an analysis of "a typical commit" using the version history of 9 open source projects. They mostly focus on the number of files changed (and how), but also provide chunk and line-size data. They compute line size changes by adding lines added and removed, thus overestimating sizes by ignoring changed lines of code. Still, they find "quite small" commit sizes without giving more details. Interestingly, they find a strong correlation between diff chunk and size. Alali et. al.'s 9 projects are large well-known open source projects. In contrast to Alali we focus solely on commit size, use a more precise measure and compute a derived function, the commit size distribution, on a more representative sample rather than 9 selected projects.

[20] analyze the impact that small changes have on various quality attributes of the software under consideration. Their data is derived from a single large closed source project. They find that one-line changes represent the majority of changes during maintenance, which is in line with our results. [12] analyze 2,000 large commits from 9 selected open source projects and they find that small commits are more corrective while large commits are more perfective. Unfortunately, the authors do not discuss as to what extent their results might be representative of open source. [23] look at the patch submission and acceptance process of two open source projects. They find that small patches are more likely to get accepted into the code base than large patches. An obvious reason may be, that smaller patches are easier to review than large patches which, if not handled quickly, get harder to review and accept with time. While not representative, Weißgerber's observation is interesting to us, as it might explain why the commit size distribution is skewed towards small commits, and why this skewness is more pronounced in larger projects.

The analysis of code repositories for various purposes is an important research area that has given birth to the annual Mining Software Repositories conference series, usually co-located with ICSE [11]. A 2009 IEEE Software special issue on Mining Software Archives [16] was followed by a symposium of the same name in 2010. Ghezzi and Gall propose not only to undertake such research but to provide a platform that allows for the distributed composition of services for such analysis work [9].

Our research has one key distinguishing feature when compared to other open source data analysis research: The size of our sample population is much larger than any other published data set and brings us close to being representative of open source.

# 6 Future Work

## 6.1 Study of Proprietary Software

We would like to extend our analysis to that of proprietary software projects. In order to do this we require access to commit statistics of proprietary software projects, and this in turn requires collaboration with software vendors to get access to their statistics.

## 6.2 Validation by Accessing Software Repositories

For the open-source projects that we analyze it would be desirable to validate our findings by picking several key projects, mining their revision history ourselves (rather than depending on the Ohloh statistics) and comparing the results to the same statistics computed over the Ohloh data (for the same projects).

## 6.3 Extended Analysis

Some analysis are not possible with the Ohloh data, and like those above require direct access to the software repositories:

Certain version control systems, like Git allow a commit to have an author, multiple signatories, and a committer. With certain others (e.g. SVN) it's not built-in, and projects have to resort to informal conventions for marking commit authorship if the author does not have commit access. Thus it is not possible to reliably reconstruct this data.

With direct access we could better characterize projects based on write access to code – whether BSD-style (a core team with commit bit can touch any part of the codebase; centralized development), Linux-style (a hierarchical system with lieutenants in charge of certain parts of the codebase; distributed development, changes can still be made to the entire tree, but commits tend to be accepted only if they are within the developer's competence), commercial open source (most development is done by paid employees; external fixes might be accepted but are committed by a paid employee)

# 7 Conclusions

This paper shows that small commits are much more likely than large commits with 50 % being 16 lines of code or less.

The actual commit size distribution of open source is best modeled by a Generalized Pareto Distribution and we have found the same kind of distribution fits for different project sizes, with the likeliness of small commits increasing with the number of developers.

The fact that it is a Pareto Distribution, which is a distribution with a long tail, also shows that large commits happen although they are less likely.

The empirical knowledge gained from actually measuring the commit size distribution is the first step to creating hypotheses for future research to improve software development tools. It can be used as a benchmark to compare projects as well.

The mathematical model presented in this paper is one step towards a more precise model of software development. It is also important for developing new software development methodologies and to develop a general model of software development.

## References


1. Alali, A., Kagdi, H., Maletic, J.I.: What's a typical commit? A characterization of open source software repositories. In: International Conference on Program Comprehension, pp. 182–191. IEEE Computer Society, Los Alamitos (2008)
2. Arafat, O., Riehle, D.: The commit size distribution of open source software. In: Hawaii International Conference on System Sciences, pp. 1–8. IEEE Computer Society, Los Alamitos (2009)
3. Beecher, K., Boldyreff, C., Capiluppi, A., Rank, S.: Evolutionary success of open source software: An investigation into exogenous drivers. Electronic Communications of the EASST 8 (2008)
4. Canfora, G., Cerulo, L., Di Penta, M.: Ldiff: An enhanced line differencing tool. In: Proceedings of the 31st International Conference on Software Engineering, ICSE 2009, pp. 595–598. IEEE Computer Society, Washington, DC (2009), http://dx.doi.org/10.1109/ICSE.2009.5070564
5. Coles, S.: An introduction to statistical modeling of extreme values. Springer, London (2001)
6. Daffara, C.: How many stable and active libre software projects? (2007), http://flossmetrics.org/news/11
7. Deshpande, A., Riehle, D.: Continuous Integration in Open Source Software Development. In: Russo, B., Damiani, E., Hissam, S., Lundell, B., Succi, G. (eds.) Open Source Development, Communities and Quality. IFIP, vol. 275, pp. 273–280. Springer, Boston (2008), http://dx.doi.org/10.1007/978-0-387-09684-1_23
8. Gartner: User Survey Analysis: Open-Source Software, Worldwide (2008), http://www.gartner.com/DisplayDocument?id=757916
9. Ghezzi, G., Gall, H.: Towards software analysis as a service. In: Proceedings of the 4th International ERCIM Workshop on Software Evolution and Evolvability, pp. 1–10. IEEE (2008)
10. Gibbons, J.D., Chakraborti, S.: Tests of Goodness of Fit. In: Nonparametric Statistical Inference, pp. 144–145. CRC Press (2003)
11. Hassan, A., Holt, R., Mockus, A. (eds.): Proceedings of the 1st International Workshop on Mining Software Repositories, MSR 2004 (2004)
12. Hindle, A., German, D.M., Holt, R.: What do large commits tell us? A taxonomical study of large commits. In: Proc. of the 2008 International Working Conference on Mining Software Repositories, MSR 2008, pp. 99–108. ACM, New York (2008), http://doi.acm.org/10.1145/1370750.1370773
13. Hofmann, P., Riehle, D.: Estimating Commit Sizes Efficiently. In: Boldyreff, C., Crowston, K., Lundell, B., Wasserman, A. (eds.) OSS 2009. IFIP AICT, vol. 299, pp. 105–115. Springer, Boston (2009), http://dx.doi.org/10.1007/978-3-642-02032-2_11
14. Lind, R., Vairavan, K.: An experimental investigation of software metrics and their relationship to software development effort. IEEE Transactions on Software Engineering 15, 649–653 (1989)

15. MathWorks: Generalized Pareto Distribution, http://www.mathworks.com/help/toolbox/stats/brn2ivz-52.html
16. Nagappan, N., Zeller, A., Zimmermann, T.: Guest editors' introduction: Mining software archives. IEEE Software 26(1), 24–25 (2009)
17. Newman, M.E.J.: Power laws, Pareto distributions and Zipf's law. Contemporary Physics 46(5), 323–351 (2005)
18. Ohloh: Forum topic: Multiple enlistments (2010), http://www.ohloh.net/forums/8/topics/4497
19. Paulson, J.W., Succi, G., Eberlein, A.: An empirical study of open-source and closed-source software products. IEEE Transactions on Software Engineering 30, 246–256 (2004)
20. Purushothaman, R., Perry, D.E.: Toward understanding the rhetoric of small source code changes. IEEE Transactions on Software Engineering 31, 511–526 (2005)
21. Ribatet, M.: A User's Guide to the POT Package, 1.4 edn. (2007)
22. Singh, V.P., Guo, H.: Parameter estimation for 3-parameter generalized pareto distribution by the principle of maximum entropy (POME). Hydrological Sciences Journal 40(2), 165–181 (1995)
23. Weißgerber, P., Neu, D., Diehl, S.: Small patches get in! In: Proceedings of the 5th Working Conference on Mining Software Repositories (MSR 2008), pp. 67–76 (2008)
24. Zenoss Inc.: 2010 Open Source Systems Management Survey (2010), http://community.zenoss.org/servlet/JiveServlet/download/38-3009/OpenSourceManagement.pdf

# ISICIL: Semantics and Social Networks for Business Intelligence

Michel Buffa, Nicolas Delaforge, Guillaume Erétéo, Fabien Gandon, Alain Giboin, and Freddy Limpens

Wimmics, Inria, I3S, University of Nice,
2004 route des Lucioles, BP 93,
06902 Sophia Antipolis
fabien.gandon@inria.fr

**Abstract.** The ISICIL initiative (Information Semantic Integration through Communities of Intelligence onLine) mixes viral new web applications with formal semantic web representations and processes to integrate them into corporate practices for technological watch, business intelligence and scientific monitoring. The resulting open source platform proposes three functionalities: (1) a semantic social bookmarking platform monitored by semantic social network analysis tools, (2) a system for semantically enriching folksonomies and linking them to corporate terminologies and (3) semantically augmented user interfaces, activity monitoring and reporting tools for business intelligence.

**Keywords:** social semantic web, business intelligence, social network, social network analysis, folksonomies, semantic wiki.

## 1 Introduction

Recently, online communities of interest have emerged and started to build di-rectories of references in their domains of interest at an impressive speed. One of the main strengths of the tools enabling these communities is their ability to turn usually passive users into active participants and producers. The diversity and the mass of users are used to tackle the diversity and the mass of information sources. Monitoring market, science and technological changes is a vital ability of today's organizations, yet the growing diversity of sources to track in each domain of interest remains a challenge for any organization. Therefore there is a growing interest in importing inside corporate information systems the tools and practices that made the success of these online communities. But, Web 2.0 tools exhibit limits when it comes to automating tasks or controlling processes. On another hand, more structured information systems often suffer from usability and knowledge capture issues. Thus a challenge of the ISICIL project is to reconcile viral new web applications with formal representations and processes and to integrate them into corporate practices. More specifically, we studied and experimented with the usage of new tools for assisting corporate intelligence tasks. These tools rely on web 2.0 advanced interfaces for

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 67–85, 2013.

interactions and on semantic web technologies for interoperability and information processing.

## 2 Overview of the ISICIL Project

Over the past four years, the goal of the ISICIL project has been to combine social networks and semantic web in a user-friendly platform supporting corporate intelligence. The project started with a multidisciplinary design methodology to specify a new form and a new platform for corporate intelligence. From the technical point of view, the main challenge of this project was to reconcile the new viral web applications with formal representations of business processes and to integrate them into communities of practice of the company. We explored new scientific developments of the notion of epistemic cooperation (human interaction oriented toward the development and transmission of knowledge) and we identified usable technological solutions. An ergonomic approach, combining impregnation of ground truth data and freer technological inspirations from bibliographic and webographic sources, was followed and evaluated. The results of this study allowed us to specify, design, experiment and evaluate new tools to support collaborative tasks in business intelligence by leveraging Web 2.0 interfaces (blog, wiki, social bookmarking) for interactions and semantic web technologies for interoperability and information processing.

All the models proposed and formalized in the project are typed graphs. These models capture structures and semantics underlying epistemic communities, their networks, their resources and their interactions. ISICIL relies on a unifying model based on RDF graphs to represent resources and community stakeholders. These models are integrated with bookmarking or "web scraping" tools. The outputs of these tools are tagged and the tags are collected to form folksonomies and analyzed to semi-automatically structure these folksonomies. The user feedback on this structure is capture when they use the search engine which offers tags related to their keywords. Users can accept, reject or adjust these suggestions and enrich or correct the structure as a side-effect of refining their queries. User profiles and links are processed by a series of operators to propose a semantic analysis of social networks e.g. centrality metrics parameterized by an ontology. Merged graphs of structured folksonomies and of social networks finally allow the detection and labeling of epistemic communities.

ISICIL is a proof of concept of the compatibility of the Semantic Web formalisms, practices and models of Web 2.0 and the philosophical framework of social epistemology. An open-source platform is available under CeCILL-C and was tested at ADEME and Orange, and all the deliverables are online[1]. In the following sections we zoom on three types of results from the project: semantic social network analysis (Section 3.1), folksonomy enrichment (Section 3.2) and community and interest detection (Section 3.3). We explain the software architecture and implementation in Section 4. We provide a guided tour of some of the users' the interfaces in Section 5. Finally we position the project within the state of the art (Section 6) and we conclude and discuss the results and evaluation (Section 7).

[1] http://isicil.inria.fr

# 3 Semantics and Social Metrics

## 3.1 Social Network Analysis (SNA)

ISICIL integrates a bookmark-centric social network and Semantic Network Analysis (SNA) tools. We used popular ontologies to model a social graph in RDF (linking people, resources, tags). We implemented the computation of the main SNA indices using Sparql and post-processing in a few cases, see [1] and [3] for details. We designed stack of tools (Figure 1) to conduct a semantic social network analysis. The goal of this stack is to provide a framework that enables us to consider not only the network structure embedded in social data, but also the schemas that are used to structure, link and exchange these data. This stack is composed of (1) tools for building, representing and exchanging social data and (2) tools for extracting social network analysis metrics and leveraging social graphs with their characteristics.

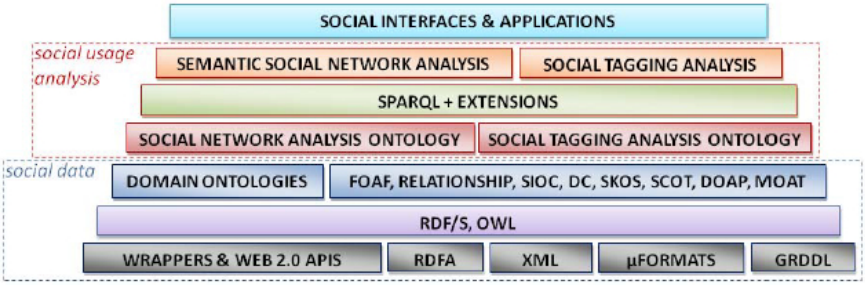



**Fig. 1.** Abstraction Stack for Semantic Social Network Analysis

We represented the social graphs in RDF, which provides a directed typed graph structure. Then we leveraged the typing of nodes and edges with the primitives of existing ontologies together with specific domain ontologies when needed. With this rich typing, semantic engines are able to perform type inferences from data schemas for automatically enriching the graph and checking its consistency.

For the analysis of the network, we designed SemSNA[2] (Figure 2) that defines different SNA metrics ranging from the annotation of strategic positions and strategic actors (like degrees or centralities), to the description of the structure of the network (diameter, etc.). With this ontology, we can abstract social network constructs from domain ontologies to apply our tools on existing schemas by having them extend our primitives. We can also enrich the social data with the SNA metrics that are computed on the network. These annotations enable us to manage more efficiently the life cycle of an analysis, by pre-calculating relevant SNA indices and updating them incrementally when the network changes over time. Moreover they can be used during the querying of social data for ordering and filtering the results and reused by other applications.

[2] http://ns.inria.fr/semsna/2009/06/21/voc.rdf

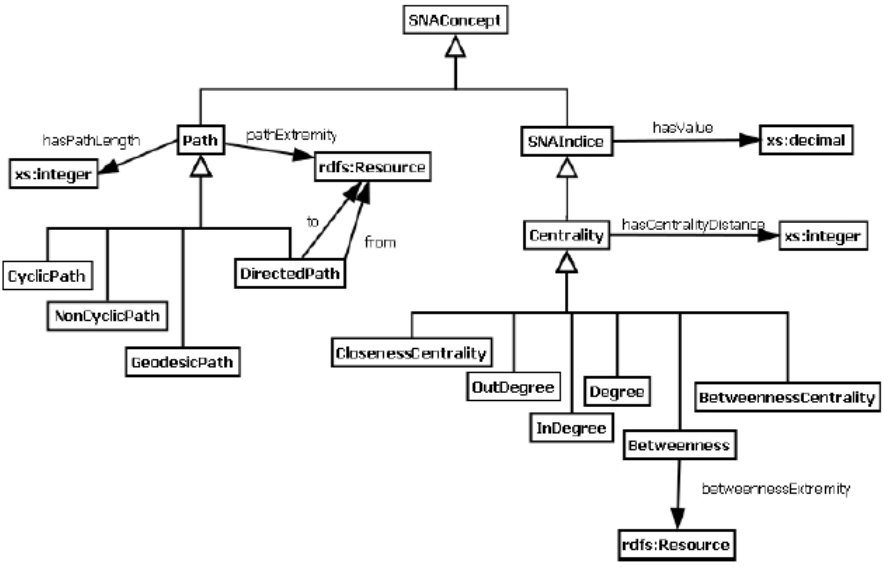



**Fig. 2.** Subset of SemSNA ontology: paths and strategic positions

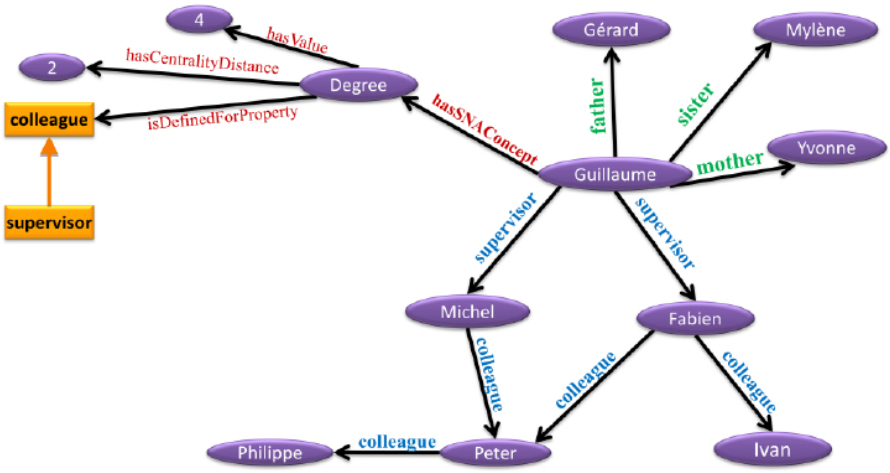



**Fig. 3.** Example of a social graph enriched with social network analysis results

Figure 3 shows an example of enrichment of a semantic social graph with SemSNA. We use the property hasSNAConcept to link Guillaume to a Degree, the property semsna:isDefinedForProperty specifies that this degree has been computed in the colleague sub-network (taking into account sub-relationships like supervisor), the property hasValue describes the value of this degree, and hasCentralityDistance defines the path length that was considered.

On top of SemSNA we proposed SPARQL formal definitions of SNA operators handling the typing of the semantic representations of social networks through parameterized queries focusing automatically on specific path patterns, involving specific resource or property types. The SPARQL queries that we designed are based on extensions of the SPARQL language that are implemented in the semantic graph engine CORESE [4]. In particular, the property path extension of CORESE [5] enables us to extract paths in RDF graphs by specifying multiple criteria such as the type of the properties involved in the path with regular expressions, or edge directions or constraints on the vertices that paths go through.

### 3.2 Semi-automated Semantic Structuration of Folksonomies

Social tagging is a successful classification means to involve users in the life cycle of the content they exchange, read or publish online. However, folksonomies resulting from this practice have shown limitations (spelling variations, lack of semantic relationships, etc.) that significantly hinder the navigation within tagged corpora. One way of tackling these limitations is to semantically structure folksonomies. This can help navigate within tagged corpora by (1) enriching tag-based search results with spelling variants and hyponyms, or (2) by suggesting related tags to extend the search, or (3) by semantically organizing tags to guide novice users in a given domain more efficiently than with flat lists of tags or occurrence-based tag clouds, or (4) by assisting disambiguation. We designed a tagging-based system that integrates collaborative and assisted semantic enrichment of the community's folksonomy. We proposed formal models and methods to support diverging points of view regarding the semantics of tags and to efficiently combine them into a coherent and semantically structured folksonomy, see [6] and [7] for details. Our approach consists in creating a synergistic combination of automatic structuring methods to bootstrap the process, and of users' contributions at the lowest possible cost through user-friendly interfaces to improve the results. The system *supports conflicting points of view* regarding the semantic organization of tags, but also helps online communities *build a consensual point of view* emerging from individual contributions.

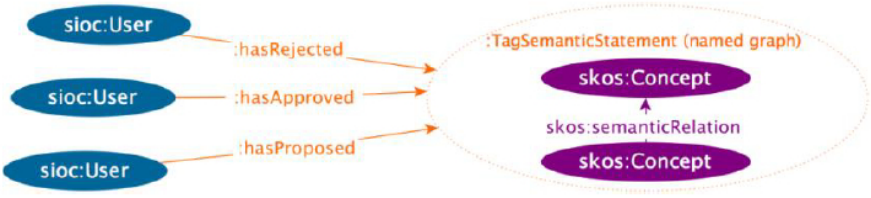



**Fig. 4.** SRTag RDF schema

We proposed an RDF schema (Figure 4), SRTag[3], which makes use of named graphs mechanisms [8] and [9] to capture statements and points of views. We encapsulate statements about tags within a named graph typed as srtag:TagSemanticStatement or more precise subclasses.

The relationships between tags can be taken from any model, but we chose to limit the number of possible relations to thesaurus-like relations as modeled in SKOS. Then we modeled a limited series of semantic actions which can be performed by users (represented using sioc:User class), namely srtag:hasApproved, srtag:hasProposed, and srtag:hasRejected. This allows us to capture and track users' opinions on the asserted relations, and thus to collect consensus and diverging points of view. We distinguish different types of *automatic* and *human* agents according to their role in the life cycle of the folksonomy. We modeled different subclasses of the class sioc:User in order to filter statements according to the users who approve it. This includes srtag:SingleUser which corresponds to regular users of the system, srtag:ReferentUser (e.g. an archivist) who is in charge of building a consensual point of view, srtag:TagStructureComputer which corresponds to the software agents performing automatic handling of tags, and srtag:ConflictSolver corresponding to software agents which propose temporary conflict resolutions for diverging points of view before referent users choose one consensual point of view.

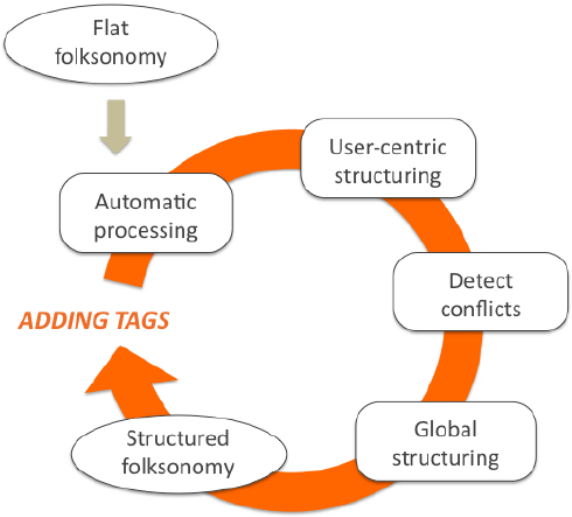



**Fig. 5.** Folksonomy enrichment lifecycle

As a result, our model allows for the factorization of individual contributions as well as the maintenance of a coherent view for each user and a consensual view linked to a referent user. Furthermore, by modeling different types of agents who propose, approve or reject tag relations, we are able to set up a complete life cycle of enriched

[3] http://ns.inria.fr/srtag/2009/01/09/srtag.html

folksonomies (Figure 5) which starts with a "flat" folksonomy (i.e. with no semantic relationships between tags) and can be decomposed as follows:

1. **Automatic processing** is performed on tags using (a) string based comparisons between pairs of tags and (b) methods on the network structure of the folksonomy (linking tags, users and resources), see [6] and [7] for details. Agents then add assertions to the triple store stating semantic relations between tags. These computations are done overnight due to their algorithmic complexity. We conducted a benchmark to evaluate the ability of string metrics to retrieve other types of semantic relations such as *related* relation, or *narrower* or *broader* relation, also called *hyponym* relation. We also implemented the algorithm described by [10] in order to extract *subsumption* relations which consists in looking at the inclusions of the sets of users associated to a tag. Co-occurrences of tags have also been used for determining missing *related* relations. The zoom in Figure 6 shows an extract of the structured folksonomy we obtain just after automatic processing.

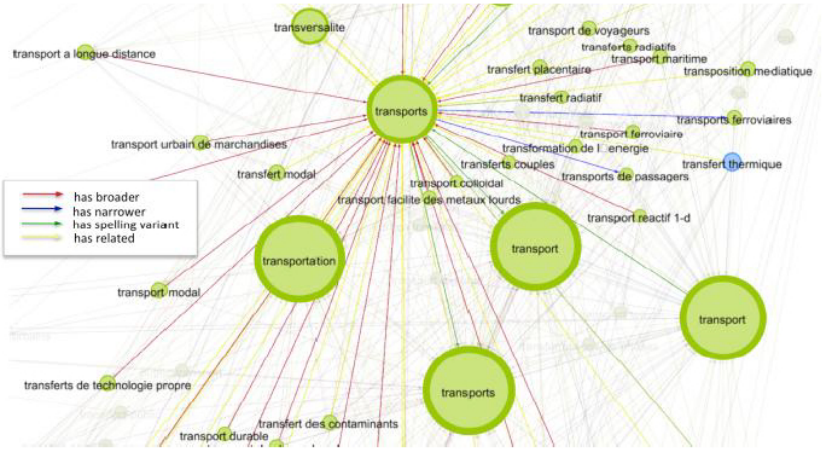



**Fig. 6.** Example of the results of automatic processing with the String Based method showing tags linked with the tag "transports". The size of the nodes indicates the in-degree.

2. **User's contributions:** people contribute through interfaces integrated into tools they use daily by suggesting, correcting or validating tag relations. Each user maintains his point of view, while benefitting from the points of view of other users.
3. **Conflict detection:** as logical inconsistencies arise between users' points of view, a software agent using production rules detects these conflicts and proposes resolutions when possible. The statements proposed are used to reduce the noise that may hinder the use of the system when, for instance, different relations are stated about the same pair of tags.
4. **Global structuring:** the statements from the conflict solver agent are also used to help *referent users* (e.g. librarians) in their tasks to maintain global and consensual

thesaurus with no conflicts. This view is used to filter the suggestions of related tags by giving priority to referent-validated relations over others suggested by computers or individual users.
5. **Structured folksonomy:** at this point of the life cycle we have a semantically structured folksonomy in which each user's point of view co-exists with the consensual point of view. Then a set of rules is applied to exploit these points of view in order to offer coherent navigation and search tools to each and every user.
6. **And again:** another cycle restarts with automatic handlings to take into account new tags added to the folksonomy.

### 3.3 Identifying Communities and Shared Interests

Building on top of our results on semantic social network analysis and folksonomy structuring, we proposed a community detection algorithm, SemTagP. This algorithm not only offers to detect but also to label communities of interest by exploiting at the same time the structure of the social graph and the semantics of the tags used. Doing so, we are able to refine the partitioning of the social graph with semantic processing and to label the activity of detected communities. We tested and evaluated this algorithm on the social network built from Ph.D. theses funded by ADEME, the French Environment and Energy Management Agency. We showed how this approach allows us to detect and label communities of interest and control the precision of the clustering and labeling.

SemTagP [2,3] is an extension of the RAK algorithm [18] in which we turned the random label propagation into a semantic propagation of tags: instead of assigning and propagating random labels, we assign to actors the tags they use and we propagate them using generalization relations between tags (e.g. skos:narrower / skos:broader) to merge over-specialized communities and generalize their labels to common hyperonyms (for instance merging football and handball communities into a sport community).

We use the directed modularity on RDF directed graphs to assess the quality of the community partition obtained after each propagation loop. When a partitioned network has a high modularity, it means that there are more connections between nodes within each community than between nodes from different communities.

SemTagP iteratively propagates the tags in the network in order to get a new partitioning: nodes that share the same tag form a community. During a propagation loop each actor chooses the most used tag among its neighbors, for a tag t we count 1 occurrence for each neighbor using t and 1 occurrence for each neighbor using a skos:narrower tag of t. We iterate until the modularity stops increasing. The penultimate partitioned network is the output of the algorithm.

In our first experimentation, we witnessed that some tags with many skos:narrower relations absorbed too many tags during the propagation phase, such as the tag "environnement" (environment), which is ubiquitous in the corpus of the ADEME agency. Such tags grouped actors in very large communities. Consequently, we added an option to refine manually the results: after the first propagation loop we present the current community partition and labeling to a user that can reject the use of skos:narrower relations of tags labeling too large communities. Then, we restart the algorithm and repeat this process until no more relation is rejected.

We formalized our algorithm in [2] and [3] and it was implemented on top of the semantic graph engine CORESE-KGRAM [4] that supports RDF/S and SPARQL 1.1 query language. We delegate all the semantic processing performed on the graph to the semantic graph engine, taking benefits of SPARQL queries to exploit semantic relations between tags. Notice that the pattern matching mechanism of KGRAM's SPARQL implementation is based on graph homomorphism that is an NP complete problem. However, many heuristic optimizations enable us to significantly cut the time calculation of the RDF graph querying.

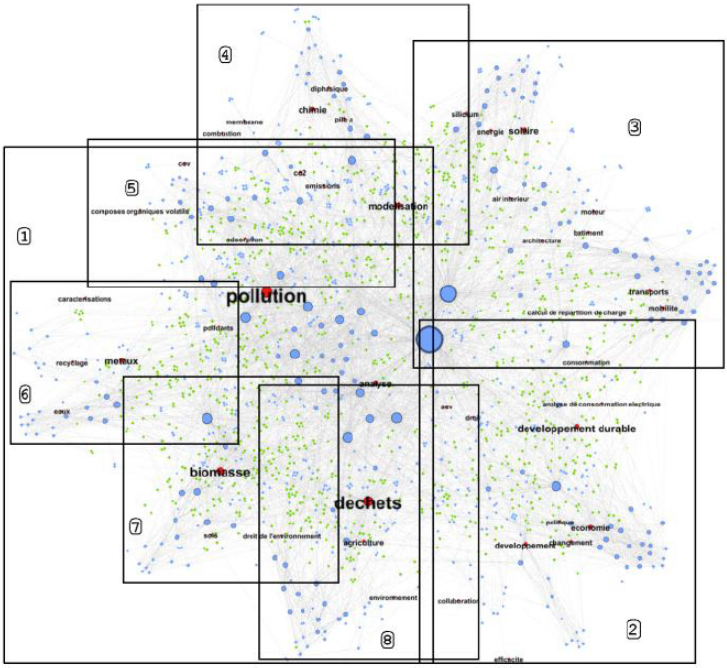



**Fig. 7.** Ph.D. social network of the ADEME with tags labeling the communities

In Figure 6 we show an overview of the ADEME social network with the labels of the communities identified by SemTagP. We used a graph visualization tool, GEPHI, with a force layout to render the results. The size of the nodes is proportional to their degrees, and the size of the tags is proportional to the size of the labeled communities. Groups of densely linked actors are gathered around few tags, which highlight the efficiency of the algorithm at partitioning the network. Moreover, communities that are labeled with tags representing related topics are close in the visualization, which enables us to build thematic area of the network using the labeling of the communities. In Figure 6, communities displayed in framed area are respectively labeled with tags related to: pollution (1), sustainable development (2), energy (3), chemistry (4), air pollution (5), metals (6), biomass (7), wastes (8). For instance, the area 3 contains tags related to

energy production and consumption with the tags *energie* (energy), *silicium*, *solaire* (solar), *moteur* (engine), *bâtiment* (building) and *transports*. This observation shows that SemTagP labeled closest communities with related labels.

## 4 Implementing ISICIL

The ISICIL platform integrates all the approaches summarized in the previous section into a web architecture deploying interconnected social semantic tools on an intranet to support business intelligence and technology monitoring activities including watch, search, notification and reporting[4].

### 4.1 General Architecture of the Platform

The platform is a typical REST API architecture, built as a JavaEE webapp, hosted in a classical servlet container. For practical reasons we split this project into 3 layers:

The core layer embeds the CORESE/KGram library as the main triple store, conceptual graph engine and SPARQL 1.1 compliant interpreter [11]. As such, we make an extensive use of new features of SPARQL 1.1 like update, named graphs, paths. This layer also implements the read/write mechanisms between CORESE/KGram and the ISICIL custom persistence system.

The business layer is dedicated to the implementation of ISICIL models and publishes services and methods required for interacting with the semantic engine. Each business object has a dedicated service for inserting/updating/deleting annotations. For mobile devices concerns, we also have pre-wired some of the main queries users can have to ask to the system. This work greatly simplifies the client-side interaction with the server and the parsing of the results. For those who need complex queries, a SPARQL endpoint service is also provided.

The RESTful API represents the HTTP translation of the business layer. Almost all of the business services have a corresponding web service. Because of the sensitive aspect of the business intelligence information, we took special care of security issues. An ontology based access control model had been designed and implemented in a prototype based on ISICIL datasets [12].

### 4.2 Clients of the RESTful API

Two software clients were also developed during the project. One is a semantic enhancement of an industrial quality open source wiki[5] whose role is to support web scraping and reporting and the second one is a Firefox add-on whose goal is to propose a semantic augmented browsing experience to the user. Users can access ISICIL information through them, but their activities within these tools also feed the ISICIL social network. For instance, the wiki tracks collaboration between contributors, comments,

[4] The platform is open-source, available on the INRIA repository, `https://gforge.inria.fr/projects/isicil`

[5] Mindtouch Dekiwiki, `http://www.mindtouch.com/`

tags and proposes some reporting features through several dashboard widgets that can be embedded into wiki pages. It also proposes corporate documentation templates to guide users in their editorial activities and to push them to bring this kind of documents out of the office.

The second tool, the Firefox add-on provides a bookmark extension, called *Webmarks*, which allows the user to share his web references through the ISICIL social network. Furthermore, users can scrap (archive) a web page that is uploaded to the wiki and use it in a wiki page to have a backup copy, to illustrate something or to initiate a discussion or a report. The Firefox add-on also embeds an RDFa parser that analyses metadata hidden in the pages that are marked. This feature for instance is used to bootstrap the Webmarking form with some Dublin core, FOAF or OGP metadata[6]. RDFa can also be injected in the PHP templates of corporate legacy tools to use their web front end as integration points and analyze what the user is currently browsing on a corporate intranet tool and start tracking his activity or suggest him some other good readings or good persons to contact, or relevant services for the data he accesses. The Figure 8 illustrates the way these tools are connected together.

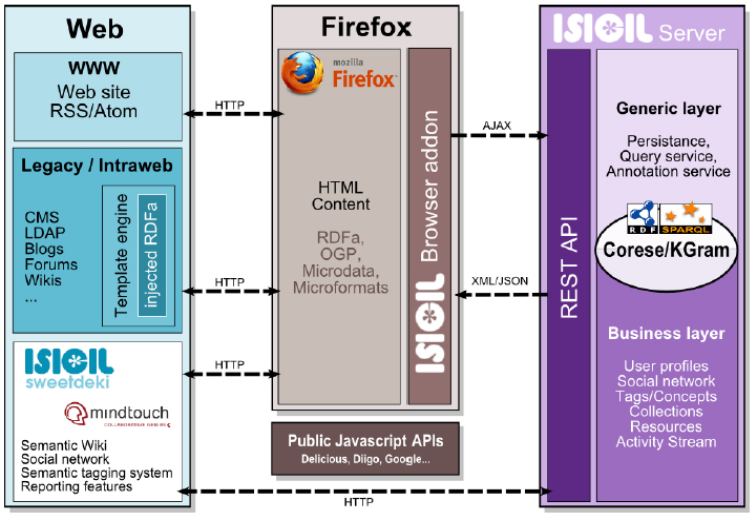



**Fig. 8.** ISICIL General Architecture

### 4.3 Data Consolidation, Metrics Computing and Connectors

One thing that the three layers architecture allowed was the building of independent, small and easy-to-maintain software agents, sharing the same business layer that the REST API. This is of primary importance as when the model evolves, automatically every dependent module benefits of the business layer update.

[6] http://developers.facebook.com/docs/opengraph/

In Section 3, we have presented the algorithms that we are using for respectively social network analysis, semantic structuration of folksonomies and community detection. Those resource-consuming processes cannot be done in real-time. For that purpose, those independent software agents can be dispatched through a local network and remotely consolidate the ISICIL database with their results. Their execution can also be scheduled as an entry in a crontab or in any other scheduling system. Our data connectors are following exactly the same design principles.

### 4.4 ISICIL Ontologies: Effective Reuse and Required Adaptations

Modeling the identity was a critical step during the design phase. As a conception guide, we relied mostly on a real use case from the French agency for sustainable development and energy mastery (ADEME). They were very interested in merging their contacts databases into a unified social network. For that task, we set up the ISICIL social network with both of their corporate directory and PHD contact databases, but to do so, many difficult aspects had to be taken into account:

- *Distinguish employees of the company and external persons*. Bootstraping the social network only with ADEME employees was not considered of interest. The most valuable information in such a corporate social network resides in the links between inner experts and industrial or academic partners.
- *Manage the turnover*. Many experts stay only a few years as members of the agency. So the system should be able to disable their account, but also keep track of what they did before and what they are doing next.
- *Bridge existing online identities under a unique URI*. Many of the staff members are contributing to the corporate knowledge under many different online identities on intranet or extranet tools, like blogs or forums. Our first work was to integrate, curate and merge all of these fragments into a single coherent entity so that it is possible to answer to (not so) simple questions such as “who wrote that?”, ”who’s currently working on that topic in the agency?”...
- *Enable online communities*. Social network users like to create online groups and communities to share information. But in our case, groups inherited from the organization chart were not considered representative of the way members wanted to collaborate. So we had to integrate in our model a second kind of group to model these parallel online communities.

The part of the ISICIL model dedicated to identity, groups and communities is illustrated by Figure 9. We made design choices using some parts of the FOAF ontologies, such as foaf:Person, foaf:Group, foaf:Organization to model information from the corporate organization chart while SIOC was used to model online activities. Though, any user of the ISICIL software has at least two identities, an official one named by a foaf:Person URI and an online account represented by a sioc:UserAccount.

At the beginning, social relations were defined as properties of the *relationship* ontology[7]. This model proposes a many foaf:knows sub-properties enabling the specification of each social relation type. But, we faced a lot of issues when we tried to calculate

[7] http://vocab.org/relationship/.html

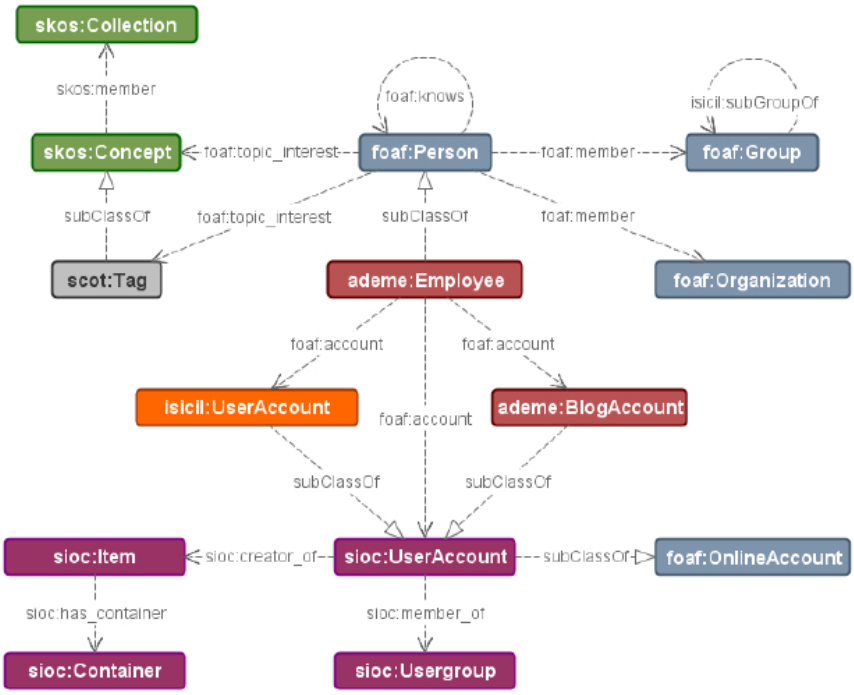



**Fig. 9.** ISICIL Global model

some basic SNA metrics. Results were false because most of these relations were defined symmetric while we needed to take into account in-degree and out-degree. For that purpose, we had to define our own non-symmetric sub-properties of foaf:knows.

Another important part of the design task was the integration of the ADEME controlled vocabulary into the system and its articulation it with other legacy folksonomies inherited from intranet CMS/Blogs/Wikis that provide free form social tagging. The idea was to bootstrap the ISICIL tagging system in such a manner that we can propose some relevant auto-completions and tag suggestions, assisting users in their tagging activities. This approach is also fostered by our semi-automated semantic structuration of folksonomies presented in Section 3.2.

To do so, we made an extensive use of the SKOS vocabulary[8], separating "official" controlled terms (skos:Concept) from tags created by users. For tags, we are using the SCOT ontology [13] that defines a scot:Tag as a subclass of a skos:Concept (Figure 9). As explained in the Section 3.2, we infer SKOS relations between these concepts. The NiceTag vocabulary [14] is used to model *tagging events* as named graphs including a creator, a date and a targeted resource. Tagging events represent a very important part of the online activity that we are tracking. We analyse the tags an expert is currently using, and tags that are related, to suggest him relevant people to contact.

[8] http://www.w3.org/TR/skos-reference/

Moreover, a macroscopic usage analysis on tagging events for a given period provides valuable information about scientific and technological trends.

Business Intelligence tasks are not always focusing on the same topics. So, an expert should be able to describe his current field of interest by a set of keywords that he can activate or deactivate according to his current activity. For that purpose, we used the foaf:topic_interest property as a link between an expert and a tag, a concepts or a person. As we can see on Figure 9, the user can organize these topics in labeled lists (skos:Collection) that he can share with others. The user who described his topics of interest this way, will be notified of incoming information produced over the ISICIL social network that is related to his active topics. These declarations of interests could also be exploited to improve the suggestions of readings and contacts the system can provide.

One of the principal concerns of experts at the ADEME is to qualify information. How to distinguish valuable information in a flow of poorly reliable data? Answering such a question requires information about authors. The ISICIL platform manages two kinds of documentary resources: legacy documents coming from corporate databases and pages produced on the ISICIL semantic wiki.

For legacy documents, most of the time, we know nothing about authors and the context where these documents were created. Thus, it is almost impossible to qualify "a priori" their informational value. But it is possible to analyze the way ISICIL users reference it. For instance, if a well-known expert in the ISICIL social network adds a Webmark on a report and tag it, there are chances that this document contains something useful. If many users do the same, this probability increases. The ISICIL platform computes ranking metrics based on Webmarks and tagging analysis that give to documentary resource a value that represents its social relevance. These values are used to sort suggestions or the results of a full-text search.

For ISICIL wiki pages, we have made an extension of the SIOC ontology for wiki pages, revisions, attached documents and comments. This model is mainly based on sioc:Item and sioc:Container and contributors on wiki pages are identified by their sioc:UserAccount and the sioc:creator_of property. Then, in addition to the previous analysis made from tagging and webmarking, we also have the social information about the author.

## 5 User Interfaces

### 5.1 Capture Online Activity

For a social network to be alive the social data need to be attractive. None of the ISCIL algorithms are efficient without good tagging, interesting wiki content, webmarks or social relations. Thus, providing intuitive tools to produce these data was a primary task. The commercial wiki (Figure 10A) we choose to be enhanced with semantic features in ISICIL already provides a friendly WYSIWYG editor and a templating system enabling the wiki pages customisation. We have replaced the original tagging system and we have implemented many hooks to capture editorial events but most of the editing system was left as is.

Finally, the webmarking extension (Figure 10B) allows a user to specify the type of the resource that he is going to mark. He can also specify if and how this resource is geo-localized (e.g. a place), temporally constrained (e.g. an event) or socially linked (e.g. a colleague). The user can also scrap the web page content to preserve and share it on an ISICIL wiki page. The webmarking tool also provides assisted tagging and sharing features.

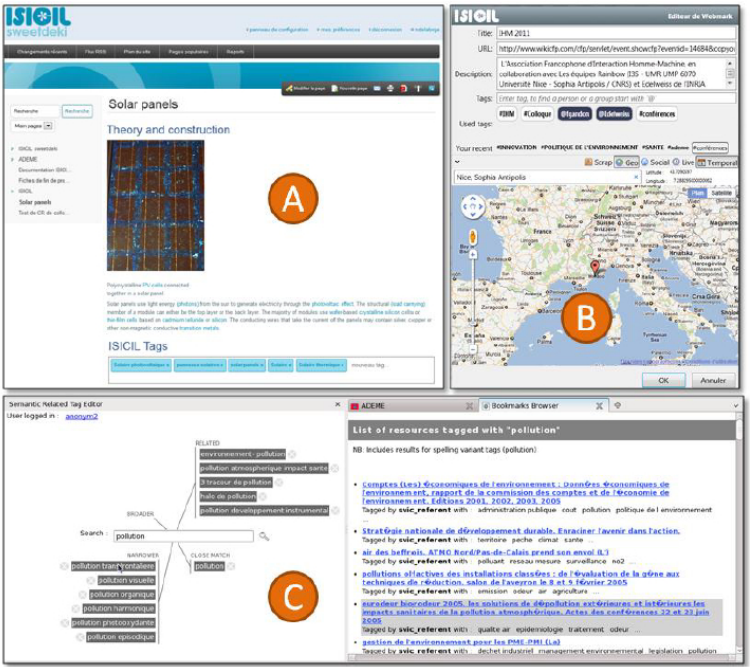



**Fig. 10.** ISICIL GUI sample : (A) the semantic wiki, (B) the webmarking dialog, (C) the "search and suggest" interface

## 5.2 Search Features

Once the social network is fed, BI tasks require global search functionalities to retrieve information. The Firefox add-on (Figure 10C) provides a "search and suggest" feature seamlessly integrating tag structuring capabilities (left part). The user was about to drag the tag "energy" towards the "spelling variant" area to state that the tag "energie" (the tag currently searched for) is a spelling variant of "energy". In this interface, as a side effect of searching and refining search results, the users make statements on the structure found for the folksonomy and contribute to its maintenance.

Recently, we have improved our full-text searching system with Fresnel based data selectors [15]. Fresnel lenses were written for each type of resources we want to show. After a full-text query, the system applies the lenses on the list of the matching URIs. Then according to the rdf:type of the resource, it automatically selects the adequate properties and output formats as defined in the corresponding lens.

### 5.3 Reporting and Data Visualization

As we said before, the ISICIL wiki (called SweetDeki) is an enhancement of a commercial wiki which integrates a custom scripting language, and powerful extension mechanisms. We developed a set of extensions that communicate with the ISICIL REST services like SNA, etc. Some inject RDFa annotations in pages, generate semantic annotations such as collaboration notifications between users for the social network, replace the original tagging system with our own, etc. We also developed a set of extensions for the WYSIWYG editor for inserting dynamic visualizations or data into wiki documents. For example, widgets from the ISICIL social networks services such as the ones presented in Figure 11 can be integrated in any document. They represent collaboration diagrams between users, users-tag-users relationships, or results from the Semantic Network Analysis tools.

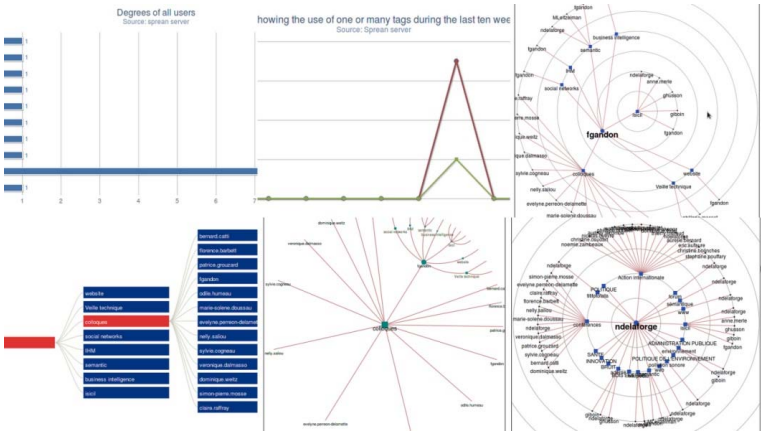



**Fig. 11.** ISICIL dashboard widgets sample

## 6 Related Works

As this article focuses on the overall ISICIL platform we will restrain this section to this topic too. The KIWI framework[9] has similarities with the ISICIL framework as both have service oriented architecture that provides a set of RESTful web services, both use a fully versioned triple store, a full text and metadata search index and services to manage content, users and tags. However, where KIWI describes itself as a "generic framework for Semantic Social Media" and is a natural evolution of the semantic wiki IkeWiki into a framework (KIWI means "Knowledge in a Wiki"), ISICIL is a framework dedicated to semantic social networks that provides a social network analysis stack and a system for structuring semi automatically the tags. Moreover, the tools developed on top of this framework have been targeted to business intelligence

[9] http://kiwi-project.eu

users: a semantic wiki based on the Mindtouch Core open source wiki that interacts with the social network and acts as a reporting tool, FireFox extensions dedicated to searching and exploring the network in order to locate experts on particular domains or collaborations between members on particular subjects, social bookmarking tools, etc. Other works related to semantic enhancement of content management systems (CMS) share common points with our work (many propose to use ontologies to link resources, people and tags), see [16] for a survey, but are not focused on the social network part or on the semantic enrichment of tags as much as ISICIL. For the years to come, works conducted by members of the IKS European project[10], that address the problem of interaction with semantic contents in CMS and in the Linked Open Data [17], open new perspective for framework like ISICIL.

## 7 Discussion and Conclusion

To conclude the project, an experiment was conducted over several months at ADEME. Its aim was to evaluate the usefulness, usability and collaborative practices allowed by the platform for a representative sample of members of the ADEME. This sample was composed of engineers, librarians and PhD students.

The experiment consisted of a series of activities carried out in a spirit of co-design with users: a) a questionnaire on business intelligence practices b) individual sessions of exploration and discovery of the platform c) guided cognitive tracking sessions in the platform d) collaborative online sessions supported by the communication features of the platform (chat and comments), e) focus group.

The data collected during these activities were analyzed in terms of:

- Purpose: to assess whether the platform is useful for achieving business objectives, to accomplish the tasks of business intelligence.
- Usability: evaluating ergonomic three main criteria: effectiveness, efficiency and satisfaction when interacting with the platform.
- Collaborative practices: identify the practices and forms of cooperation induced by the platform.

In general, the ISICIL platform has been validated regarding the usefulness and collaborative practices: users clearly see the collaborative potential of ISICIL. However, the usability of the platform is the critical point. The main point is the lack of unity among the features, it was difficult for the user to understand that the building blocks of ISICIL (browser, wiki and webmarks) form a whole; interfaces are isolated from each other, and users lack a main interface that centralizes and guides them; this made the understanding and learning of the platform too heavy for the user. Despite these negative aspects of usability, multiple users ADEME expressed the wish to continue experimenting with ISICIL, in addition, discussions are underway with ADEME on possible actions for further developments of ISICIL within the agency. To improve the usability of the platform, ergonomic recommendations have been proposed. Many of these recommendations have already been taken into account.

[10] http://www.iks-project.eu

As we have seen, the ISICIL platform is innovative in many aspects of the features it proposes. This truly experimental characteristic has a cost. Some of these features are not optimized enough and can involve sometimes performance issues.

In this paper, we have presented the ISICIL platform as the result of a four years research project. We have seen the core algorithms and the software architecture. In this project, we can say that we have fully embraced the latest technologies of the semantic web and the entire architecture of the software suite relies on semantic web standards. The main difficulty here resided in the articulation of many lightweight ontologies in a read and write context. Sometimes we had to make some arbitrary choices in the design in order to solve inconsistency issues implied by this aggregation. But, we believe that this experiment proves the effectiveness of semantic web technologies in a "real world" scenario of business intelligence.

**Acknowledgements.** French Research National Agency (ANR) and the ISICIL project team (contract No. ANR-08-CORD-011-05).

## References

1. Erétéo, G., Buffa, M., Gandon, F., Corby, O.: Analysis of a Real Online Social Network Using Semantic Web Frameworks. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 180–195. Springer, Heidelberg (2009)
2. Erétéo, G., Buffa, M., Gandon, F.: SemTagP: Semantic Community Detection in Folksonomies. In: IEEE/WIC/ACM International Conference on Web Intelligence, Lyon (2011)
3. Erétéo, G.: Semantic Social Network Analysis, PhD Thesis Telecom ParisTech Inria Orange Labs (2011)
4. Corby, O., Dieng-Kuntz, R., Faron-Zucker, C.: Querying the semantic web with the corese search engine (2004)
5. Corby, O.: Web, Graphs and Semantics. In: Eklund, P., Haemmerlé, O. (eds.) ICCS 2008. LNCS (LNAI), vol. 5113, pp. 43–61. Springer, Heidelberg (2008)
6. Limpens, Gandon, F., Buffa, M.: Helping online communities to semantically enrich folksonomies. In: Proceedings of Web Science 2010 (2010)
7. Limpens, F.: Multi-points of View Semantic Enrichment of Folksonomies, PhD thesis, University of Nice (2010), http://hal.archives-ouvertes.fr/hal-00530714/fr
8. Carroll, J., Bizer, C., Hayes, P., Stickler, P.: Named graphs, provenance and trust. In: WWW 2005: Proc. of the 14th International WWW Conference, pp. 613–622 (2005)
9. Gandon, F., Bottolier, V., Corby, O., Durville, P.: Rdf/xml source declaration, w3c member submission (2007), http://www.w3.org/Submission/rdfsource/
10. Mika, P.: Ontologies Are Us: A Unified Model of Social Networks and Semantics. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 522–536. Springer, Heidelberg (2005)
11. Corby, O., Faron-Zucker, C.: The KGRAM Abstract Machine for Knowledge Graph Querying. In: IEEE/WIC/ACM International Conference (2010)
12. Villata, S., Delaforge, N., Gandon, F., Gyrard, A.: An Access Control Model for Linked Data. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2011 Workshops. LNCS, vol. 7046, pp. 454–463. Springer, Heidelberg (2011)

13. Kim, H.-L., Breslin, J.G., Yang, S.-K., Kim, H.-G.: Social Semantic Cloud of Tag: Semantic Model for Social Tagging. In: Nguyen, N.T., Jo, G.-S., Howlett, R.J., Jain, L.C. (eds.) KES-AMSTA 2008. LNCS (LNAI), vol. 4953, pp. 83–92. Springer, Heidelberg (2008)
14. Monnin, A., Limpens, F., Gandon, F., Laniado, D.: Speech acts meet tagging: NiceTag ontology. In: I-SEMANTICS (2010)
15. Pietriga, E., Bizer, C., Karger, D., Lee, R.: Fresnel: A Browser-Independent Presentation Vocabulary for RDF. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 158–171. Springer, Heidelberg (2006)
16. Laleci, G.B., Aluc, G., Dogac, A., Sinaci, A., Kilic, O., Tuncer, F.: A semantic backend for content management systems. Knowledge-Based Systems 23(8), 832–843 (2010)
17. Damjanovic, V., Kurz, T., Westenthaler, R., Behrendt, W., Gruber, A., Schaffert, S.: Semantic Enhancement: The Key to Massive and Heterogeneous Data Pools. In: Proc. of the 20th International IEEE ERK Conference (2011)
18. Raghavan, R.N., Albert, R., Kumara, S.: Near Linear Time Algorithm to Detect Community Structures in Large Scale Network. Phys. Rev. E 76, 036106 (2007)

# A Multi-dimensional Comparison of Ontology Design Patterns for Representing *n*-ary Relations

Aldo Gangemi and Valentina Presutti

STLab, ISTC-CNR, Rome, Italy

**Abstract.** Within the broad area of *knowledge pattern science*, an important topic is the discovery, description, and evaluation of modeling patterns for a certain task. One of the most controversial problem is constituted by modeling relations with large or variable (polymorphic) arity. There is indeed a large literature on representing n-ary relations in logical languages with expressivity limited to unary and binary relations, e.g. when time, space, roles and other knowledge should be used as indexes to binary relations. In this paper we provide a comparison of several design patterns, based on their respective (dis)advantages, as well as on their axiomatic complexity. Data on actual processing time for queries and DL reasoning from an in-vitro study is also provided.

**Keywords:** n-ary relations, ontology design patterns.

## 1 A Knowledge Pattern Science

Semantics is finally taking off in the web context, impacting on the way data and software applications are being built both at academic and industrial levels. The Semantic Web, after some years of lagging behind, just while the social web was substantially changing the relations between humans by creating an unprecedented world of socio-technical entities, has now started providing flexible, yet precise solutions that scale well beyond the artificial-intelligence-oriented past. The growth of the Linked Open Data Cloud and the widespread adoption of the Open Data paradigm in public administrations and e-Science is accompanied by the adoption of natural language processing techniques in content management systems and search engines ("semantic search"), and by novel paradigms of data consumption: visual analytics and exploratory search. Ontologies are being used in the large for data integration and interoperability, and in layman tools like vCard, RSS, etc. Big graphs, like Google Knowledge Graph and Facebook Social Graph reuse and extend semantic web data or technologies, and are clear indications of the paradigm shift that Jim Hendler and Tim Berners-Lee envisioned in the late nineties of the past century, i.e. to put semantics at the core of the Web and the socio-technical system that is growing from it. This paradigm shift is fully endorsed by a research area called *knowledge pattern science* [16], which tries to discover, represent, reuse, extract, and reason with, structures, called

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 86–105, 2013.

knowledge patterns, which emerge as *relevant invariances* from either structured or unstructured content: ontologies, schemas, databases, web links, social graphs, natural language, etc. In the vein of Christopher Alexander's work on design patterns [1], we call *relevant invariances* the structures that are key for solving problems, and therefore are closely associated to requirements, centrality in graphs and linguistic parse trees, user-orientation in interaction, etc.

Some research carried out by our laboratory in this area include e.g. modelling and experimenting with ontology design patterns [6,7,34], knowledge patterns extracted from full text [33], crowd-sourced patterns discovered in Wikipedia link structure [32]. In this paper, we show an example of such knowledge pattern science by studying modeling solutions for arbitrary relationships in web ontologies and linked data.

## 2 Problem Addressed

The problem of logically representing facts that involve more than two entities, usually called *n-ary relations*,[1] is a known issue in formal languages that only have unary or binary predicates, as it is the case in semantic web languages (RDF, OWL, RIF), many frame languages, most description logics, etc. Restricted expressivity is well known to require an approximation of the intended semantics of a model.

In the very spirit of knowledge pattern science, behind this modeling requirement, there is the big problem of *recognizing and describing the boundaries* of a fact, situation, event, from an arbitrary set of data, or from a graph (a description of this problem in the Semantic Web context is [16]). The problem is therefore related to cognitive, philosophical, logical, and computational issues that go well beyond the pragmatics of approximating the intended semantics of the predicates.

As an example of the related issues, on one hand, Gestalt psychology described some principles that seem to affect the way humans organize their perception, such as *emergence*, *reification*, *invariance*, and *pithiness*, all pointing to our "synthetic" attitude to perceiving and describing the world and our knowledge. These principles sometimes are found in actual modeling practices, sometimes not. For example, extended axiomatization of ontologies goes deeper in terms of analytic specification, and in principle enhances interoperability if semantic data are structured according to those axioms, but goes also against gestaltic principles. However, current web ontology engineering practices show that interoperability can be reached more easily and pragmatically by following gestaltic pithiness. In this case, we see the emergence of conflicting requirements with reference to e.g. intuitiveness vs. complexity of modeling, and technical views on data integration and interoperability.

In this paper we do not concentrate on the cognitive motivations why n-ary relations are difficult to handle in many logical languages, or why ontology

[1] *n-ary* would actually include also unary and binary relations, but a convention has emerged to use the term for relations with arity higher than 2.

designers have so many different opinions on the pros and cons of the different modeling solutions adopted. On the contrary, we get empirical evidence of those pros and cons.

In [14] we have started an empirical study on comparing logical patterns for representing extensional and intensional n-ary relations[2] in semantic web languages. In this paper, we focus on one of the problems addressed in [14], by deepening the analysis of how extensional n-ary relationships are represented, and adding new patterns and data.

We firstly (Section 3) summarize the (huge) literature on the topic. In Section 4 we describe the general characterization of the problem in [18], and motivate the need for an empirical analysis of design solutions beyond first-order logic ones. In Section 5 we compare seven different designs in OWL2, with realistic examples, on a multi-dimensional design space. Section 6 presents a discussion of the results of this work, which come from an "in vitro" study, and relates them to quantitative evidence of the presence of some design patterns on the linked open data cloud, as recently published in [35].

## 3 Related Work

N-ary relations are everywhere in human interaction. Underlying natural language verbal structures, in relational database tables, when recognizing or talking about events, when interpreting news, when making medical diagnoses, interpreting legal norms, controlling situations, or simply preparing a coffee. Most of the times, more than two entities are bound together, and we cannot easily represent that boundary with just unary and binary predicates.

Ontology engineering has focused on some special cases of n-ary relations, such as time-indexing of relations, role-indexing of people and objects in actions, projection or slicing of objects through space, time, roles (called sometimes *quaentities*), aggregation of selected entities to make an observation emerge, etc. A notable problem emerged in the last years is to represent, discover, reconstruct, or integrate event descriptions out of linked data (cf. [36]). It is then needed to attempt a general overview and empirical comparison of the solutions that have been devised to represent n-ary relations.

The amount of related work pairs the relevance of real world use cases presented above; there is no room in this paper to make a decent survey of them, hence we only recap them according to the different communities and topics. There is a long history of solutions to deal with relations that range over more than two arguments, or have polymorphic arities. (Generic) use cases and solution proposals come from AI (e.g. *situation calculus* [27], *relation reification* [11,12,20,26], Minsky's *frames* [28], *description logics* [2,10,30], etc.), from philosophy (e.g. *Gestalt theory* [24], Davidson's *reified events* [9], and the debate on the nature of events vs. propositions [29]), from linguistics (e.g. Bach's *eventualities* [3], *Discourse Representation Theory* [23], FrameNet *frames* [4,32], cognitive

[2] The difference is between actually occurring *relationships*, and *relation* types.

linguistics *schemata* [13]), from conceptual modeling (e.g. [17]), from the Semantic Web (e.g. *n-ary relation reification* [14,15,31], *fluents* [5,37], *named graphs* [8], *linked data integration* [21], *RDF data structure extensions* [25]), etc. We have taken inspiration from the literature, and have tried to single out all the patterns that are used for n-ary relation representation.

## 4 Three Logical Patterns to Time Indexing of Sentences

Based on [19], Pat Hayes recently stimulated the discussion [18] with respect to how to attach temporal indexing to sentences of the form $R(a, b)$, and suggested three possible *logical patterns*, summarized as follows (we quote him literally here):

I. *Attach it to the sentence*, meaning that the sentence $R(a, b)$ is true at the time $t$. [...] Call this 3D.
II. *Attach it to the relation as an extra argument*, and call the relation a 'fluent': $R(a, b, t)$ [...] Call this 3D+1.
III. *Attach it to the object terms*, (using a suitable function, written here as an infix @): $R(a@t, b@t)$ [...] @ operation [...] means the t-slice of the thing named [...] Call this 4D.

Hayes shows that the different patterns are just syntactic variants, which (in principle) could be unified by some algorithm. Indeed the algorithm indicated for unification by [19] is purely syntactic, and works on a parse tree representation of the three solutions, by showing that the time indexing parameter "trickles down" from 3D to 4D, i.e. from sentences to relational constants to individual constants. Given the three propositions 1,2,3,

```
(R(a,b))+t      [sentence indexing]      (1)
R(a,b) + t      [relation indexing]      (2)
R(a+t,b+t)      [individual indexing]    (3)
```

the parse tree (Figure 1) shows the trickling down of the time parameter from the sentence level to the individual level through the relation level.

The unification suggested by [19] however works well with a formal language that has enough expressivity to encode the topology of the parse tree. For example, this is straightforward in first-order logic with some meta-level sugar[3] (propositions 4,5,6), but not in a description logic.

```
holdsAt(R(a,b)),t)      (4)
R(a,b,t)                (5)
R(h(a,t),h(b,t))        (6)
```

[3] [18] also adds a sort for time-indexed terms of the vocabulary, so requiring a sorted FOL.

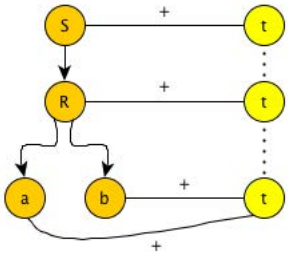



**Fig. 1.** The parse tree of Hayes' time-indexing solutions, with the "trickling-down" time parameter

Notice that in FOL we already need some design touches to represent the solutions effectively: the `holdsAt` predicate is meta-level sugar for a proposition holding at a certain time, while the dyadic function `h` (standing for *history*) encodes the "time slices" of the individuals, e.g. the time slice of `a` at `t` is the history `h(a,t)`.

But besides those touches, FOL design preserves the basic topology of the relation, i.e. a sub-tree remains invariant across transformations, as Figure 1 shows: the parse tree from statements to relations to individuals still exists untouched in the three solutions. Moreover, the different designs do not affect the constants of the vocabulary, which stay the same {`R, a, b, t`} (and the `holdsAt` relational constant is meta-level sugar).

On the contrary, when attempting to represent those solutions in formal languages with restricted expressivity, we do usually affect relation topology (e.g. because of relation reification in OWL, which changes the parse tree) and/or vocabulary constants (e.g. because of individual time slice reification in OWL, which adds new individual constants). This means that for languages with restricted expressivity we need to understand the design consequences of each solution, which can eventually bring representation and reasoning constraints.

For this reason, instead of attempting a unification algorithm for restricted expressivity languages, we want to compare, on a multi-dimensional design space, the respective solutions. Only once the design choices are compared, we can also figure out to what extent the different solutions can be unified, e.g. by providing a "lingua franca" approach that is able to reconcile the design choices implemented for each solution.

Consider that the proposal of Hayes' came in the middle of a discussion about 3D vs. 4D ontologies, and the approaches were tied to time indexing, but the problem is more general, and touches any attempt to include a new dimension in order to perspectivize facts or entities, e.g. with space, roles, etc., as exemplified in Section 3.

As a matter of fact, often one cannot know in advance the arity that is intended when a(n intensional) relation is used. The classic example by Davidson [9] is the predicate *preparing a coffee*: how many arguments are needed: the agent preparing the coffee, the amount of coffee, its quality, the coffee-pot, the heating, the kind of water, the time and place of preparing it, etc.?

## 5 Seven RDF-OWL Patterns for *n-ary* Relation Representation

We have looked for representations of the three Hayes' logical patterns in RDF and OWL2 knowledge bases, and we have singled out seven RDF and OWL2 logical patterns. Then we have compared the resulting models against a multi-dimensional design space, including the following dimensions:

(a) amount of axioms needed for the representation of each design pattern, calculated on a normalized set of 10000 individuals per ontology
(b) expressivity of the resulting model, in terms of description logic varieties
(c) time needed to check the consistency of the model
(d) time needed to classify the model
(e) amount of newly generated constants needed
(f) ability to support DL reasoning with reference to the full n-ary model (cf. Section 4)
(g) ability to support polymorphism (possibility to add new arguments when needed)
(h) preservation of FOL relation topology, which we call *relation footprint*, i.e. the vability to fully navigate the graph structure of n-ary relations instances, as described in Section 4
(i) intuitiveness of representation and usability: these have been evaluated anecdotically and subjectively, but an ongoing study is addressing users. Indirect usability data come also from an empirical study on RDF temporal data patterns for linked data [34], which we discuss in Section 6.2.

Dimensions (a) to (e) are quantitative and objective, dimensions (f) to (h) are objective, but qualitative, and dimension (i) is subjective and qualitative, except for combined results presented in Section 6.2. We represent qualitative dimensions on a three-value scale (good, limited, none).

In order to represent the approaches in a computable language, the patterns are exemplified with reference to a leading example of time indexing: the fact that Garibaldi landed in Sicily in 1860 (during the so-called *Expedition of the Thousand*). We restricted this in-vitro study to a simple case, but we remark again that larger arities are just more complex variations of the same design patterns.

In order to create a critical mass of individuals and axioms for ontologies that sample the patterns, we have used SyGENiA (Synthetic GENerator of instance Axioms)[4], which, contrarily to many random axiom generators, allows to specify a query that is used to generate ABox axioms from an ontology. We have used Pellet 2.3[5] that produces accurate accounts of time used by the reasoner, running on a MacBookPro 3.06 GHz Intel Core Duo processor, with 8GB of RAM and MacOSX 10.7.3.We have not attempted to balance axioms with specific constructs, hoping that SyGENiA would think about it by using the

[4] http://code.google.com/p/sygenia/
[5] http://clarkparsia.com/pellet/

queries provided. This is not completely true however, and the results, iterated 10 times to neutralize random effects of concurrent processes, reflect the lack of a real balance. Since we were interested in a first assessment of these dimensions, we have decided to live with some imbalance derived from the synthetic ABox, deferring a more grounded assessment to in vivo studies.

In this section, the patterns are presented and exemplified quite schematically for space reasons, and their pros and cons with respect to the seven dimensions are briefly explained. Section 6 will recap on these summaries, and will present tables that compare the measures and the objective or subjective assessments.

### 5.1 StatementContextualization

This pattern is one of the alternative OWL2 logical patterns for the FOL *sentence indexing* Hayes' logical pattern, à la context/modal logic or 3D ontologies. In RDF (*referring to a specific implementation by quad store using named graph*) or OWL2 a possible representation is depicted in Figure 2. This solution requires the contextualization of the triples using a certain index (1860 in the example). The contextualization can be implemented in different ways. In RDF, the triple data structure is extended by adding a fourth element to each triple, which is intended to express the *context* (a label), or the *named graph* (i.e. a document URI) of a set of triples. In OWL2, the same solution can be obtained by indicating an *ontology URI* instead of a named graph, either in a quad store for RDF serialization, or by using a `holdsAt` annotation property that asserts the time index to the named graph or the ontology URI. This solution has the advantage of being able to talk about assertions, but the disadvantages of needing a lot of contexts (i.e. documents) that could make a model very sparse.

With respect to *polymorphism*, this solution cannot do much, because the possible design depends on the ability to create appropriate contexts that handle a partitioning of the triples that can prevent ambiguities, e.g. for different landing events of Garibaldi in Sicily in different years or with different modalities.

This solution disrupts the *relation footprint* depicted in Figure 1, because no connection is maintained between $t$ and either the relation or the individuals. The sentence is actually transformed into a context (an ontology in this rendering),

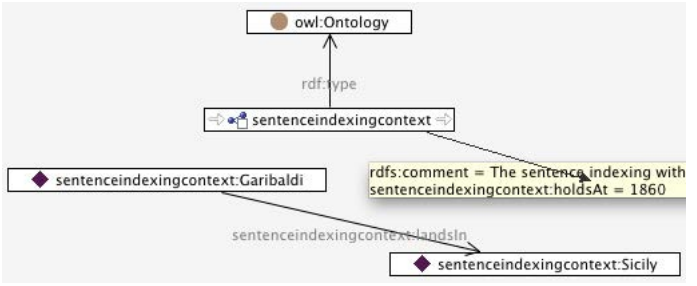



**Fig. 2.** Example OWL graph for *StatementContextualization* pattern

which is connected to $t$, but it is not available to DL reasoning. The footprint could be reconstructed in RDF by means of a SPARQL query, but only if we assume that each named graph only contains one occurrence of the `landsIn` property, and no other properties in the graph need to be indexed. Of course, we can think of more hacking, but here we want to stay close to typical modeling practices.

A radical proposal to extend the RDF data structure to quintuples ("quins") to deal with starting and ending times is [25], in which there is also a proof of the computational advantages of quin-stores over reified relation-based models.

### 5.2 StatementReification

This pattern is a second alternative for the *sentence indexing* Hayes' logical pattern. In OWL2 (*OWL2 axiom reification/annotation*), or in RDF (*statement reification/annotation*) a possible solution is depicted in Figure 3. This solution requires reification of either a RDF statement or an OWL2 axiom, and its annotation. Syntactically, RDF and OWL2 differ, because the latter puts axiom reification and annotation in one construct.

This solution has the advantage of being able to talk about assertions, but the disadvantages of needing a lot of reification axioms, to introduce a primary binary relation to be used as a pivot for axiom annotations, and that in OWL2(DL) reasoning is not supported for axiom annotations.

With respect to *polymorphism*, this solution needs more statement/axiom annotations, e.g. for the sentence *Garibaldi landed in Sicily in 1860 with 1000 soldiers*:

```
_:[Garibaldi landsIn Sicily] with 1000Soldiers          (7)
```

However, it is not possible to ensure that different events of Garibaldi landing in Sicily in the same year will be associated with the correct modality: if he landed twice, one with 500 soldiers, and another with 1000, both will be true for both landings.

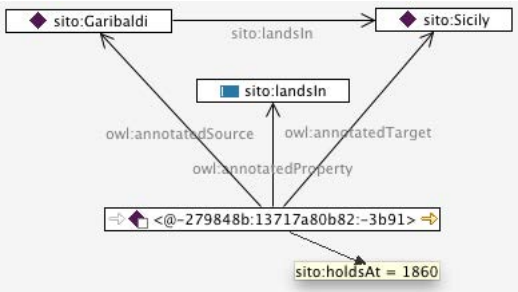



**Fig. 3.** Example OWL graph for *StatementReification* pattern

This solution loosely maintains the *relation footprint*, because the connection between $t$ and either the relation or the individuals is only granted by means of an annotation, or statement reification, which are not used for DL reasoning. Anyway, the footprint can be reconstructed in RDF by means of a SPARQL query, e.g.:

```
SELECT ?r ?x ?y ?t
WHERE {?r ?x ?y . ?a owl:annotatedProperty ?r .
           ?a owl:annotatedSource ?x .
        ?a owl:annotatedTarget ?y . ?a holdsAt ?t }
```

### 5.3 StatementAbstraction

This pattern is another alternative OWL2 logical pattern for the FOL *sentence indexing* logical pattern. A sample model on the running example is depicted in Figure 4. This solution requires a *singleton domain and range* for an object property, which is also punned as an individual in order to assert time indexing.

With respect to *polymorphism*, i.e. the need to add more arguments to the relation "on-the-go", this solution needs more punned singleton-domain properties, e.g. for the sentence *Garibaldi landed in Sicily in 1861 with 1000 soldiers*:

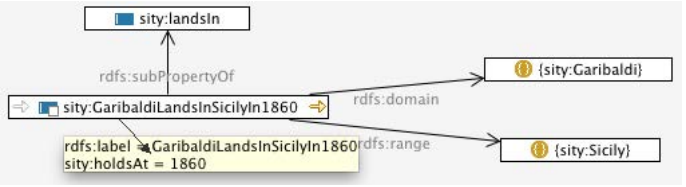



**Fig. 4.** Example OWL graph for *StatementAbstraction* pattern.

```
GaribaldiLandsInSicilyWith1000Soldiers rdfs:subPropertyOf landsIn   (8)
GaribaldiLandsInSicilyWith1000Soldiers rdfs:domain {Garibaldi}      (9)
GaribaldiLandsInSicilyWith1000Soldiers rdfs:range {Sicily}         (10)
GaribaldiLandsInSicilyWith1000Soldiers with 1000Soldiers           (11)
```

which looks quite goofy in terms of naming.

This solution loosely maintains the *relation footprint*, because the connection between $t$ and either the relation or the individuals is only granted by means of the punning of the sub-property of `landsIn`, but punning does not enable DL reasoning across object and metalevel. Anyway, the footprint can still be reconstructed in RDF by means of a SPARQL query that traverses all the indirections created by this solution (domain and range to nominals to individuals, time to punned sub-property to property).

### 5.4 Situation

This pattern implements the *relation indexing* logical pattern, à la frame logic, situation calculus, or 3D+1 ontologies. A sample model on the running example is depicted in Figure 5. This solution requires to put `owl:hasKey` axioms for preserving identification constraints (i.e. that any two same occurrences of the relation would be actually inferred to be the same). Ordering can be put in the name of properties as an index when ambiguity arises. Punning is not needed.

This solution has the advantage of being able to talk about assertions as (reifying) individuals, but the disadvantage of not being able to use them as properties as well (this could be accommodated by punning the reified relation individuals as object or data properties).

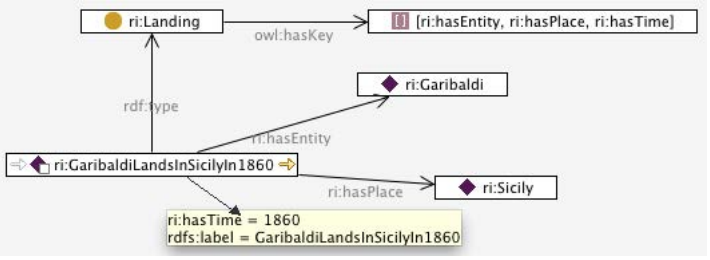



**Fig. 5.** Example OWL graph for *Situation* pattern

Another advantage consists in representing temporally heterogeneous relations, at the cost of additional entities and axioms (one more property in the TBox, $n$ more data declarations), e.g.:

```
Liking hasAgent John                    (12)
Liking hasTime 2011                     (13)
Liking hasTarget NewYork                (14)
Liking hasTargetTime 1899               (15)
```

For a complete representation, there should be one main time (the time of the main fact expressed in the sentence), and other times related to situations referred by the elements of the sentence. E.g. in axiom 15 NewYork is considered at a different time from the time of the liking situation (as from axiom 14). The time of the sentence can even be different from the times reported in the content of that sentence, e.g. as from this multiple time reference: *From 2007 to 2008, John used to like New York in 1899*.

With respect to *polymorphism*, this solution only needs more property assertions, e.g. for the sentence *Garibaldi landed in Sicily in 1861 with 1000 soldiers*:

```
GaribaldiLandsInSicilyIn1860 with 1000Soldiers   (16)
```

This solution maintains the *relation footprint*, because the connection between $t$ and the relationship is provided by a reification, which preserves direct connections to the entities, and ensures uniqueness of tuples via an `owl:hasKey` statement. This preserves DL reasoning and closely approximates FOL semantics. The footprint can be reconstructed in RDF by means of a SPARQL1.1 query with property paths[6], e.g.:

```
SELECT ?r ?x ?y ?t
WHERE {?r1 a ?r . ?r1 ?b1 ?x . ?r1 ?b2 ?y . ?r1 ?b3 ?t .
?r owl:hasKey/rdf:first ?b1 . ?r owl:hasKey/rdf:rest/rdf:first ?b2 .
?r owl:hasKey/rdf:rest/rdf:rest/rdf:first ?b3 }
```

The Situation pattern has been proposed in various forms. [31] describes a purely logical pattern; [15] proposes a `Situation` knowledge pattern[7], with a basic OWL vocabulary, that has been extended in domain-oriented use cases.

### 5.5 Simple4D

This pattern implements a basic version of the *individual indexing* logical pattern, à la 4D ontology. As a sentence in FOL, an example is (cf. axiom schema 6, from which axiom 17 departs, because the binary function $h$ cannot be expressed in OWL, and only the reified value of the function is therefore expressed). Figure 6 depicts a sample model on the running example for this pattern. It requires to introduce new entities: the time slices of the individuals referenced in the sentence.

```
landsIn(Garibaldi@1860,Sicily@1860)                    (17)
```

An advantage of this solution is that it allows to compactly (no additional axioms required) represent temporally heterogeneous relations, e.g. `John@2011 likes NewYork@1899`. However, the time of the sentence as different from the times of the elements cannot be represented.

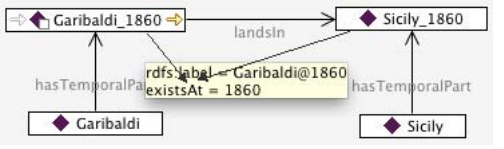



**Fig. 6.** Example OWL graph for *Simple 4D* pattern

A general objection to 4D approaches is that we could have *nD* entities, e.g. for place, orientation, state of matter, numerosity, etc. In other words, we can perspectivize things within any dimension, not just time (cf. [13]). The solution 5.6 is an extension that allows to create more general *contextual slices*.

[6] Property paths are not strictly needed, but the syntax is much more readable.
[7] http://www.ontologydesignpatterns.org/cp/owl/situation.owl

With respect to *polymorphism*, this solution needs more kinds of perspectivized individuals, e.g. for the sentence *Garibaldi landed in Sicily in 1861 with 1000 soldiers*:

```
GaribaldiWith1000Soldiers with 1000Soldiers                    (18)
Garibaldi hasCollectivePerspective GaribaldiWith1000Soldiers   (19)
```

In this respect, 5.6 appears more elegant because it does not require additional names for each type of index, but just one for a context slice, while the context can bear the indices.

This solution maintains the *relation footprint*, because the connection between $t$ and the relationship is a direct trickledown like in FOL. Also DL reasoning is preserved, and closely approximates FOL semantics. The footprint can be reconstructed in RDF by means of a SPARQL query, e.g.:

```
SELECT ?r ?x ?y ?t
WHERE { ?xt ?r ?yt . ?x hasTemporalPart ?xt .
      ?y hasTemporalPart ?yt . ?xt existsAt ?t . ?yt existsAt ?t }
```

### 5.6 ContextSlices

This pattern implements a more sophisticated variety of the *individual indexing* logical pattern, by actually mixing 5.4 and 5.5. As a sentence in FOL an example is:

```
landsIn(Garibaldi@1860,Sicily@1860,ExpeditionOfTheThousand)    (20)
```

that actually inserts new information by giving a name to a context parameter, i.e. the implicit knowledge of the context, in which Garibaldi landed in Sicily in 1860. In OWL2 (*contextualized individual logical pattern*), a possible solution is depicted in Figure 7. This solution requires to introduce new entities: the contextual projections and assignment of the individuals referred to in the sentence, as well as a context index that takes the indexes. An advantage of this solution is that it allows to get the freedom of the pattern 5.4, which allows to attach all

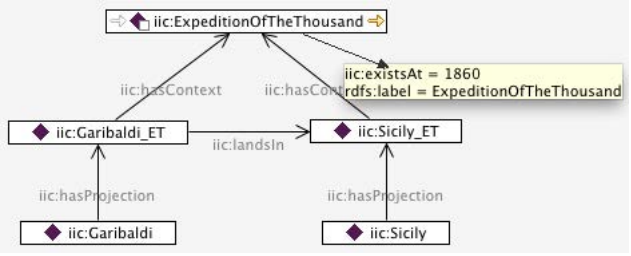



**Fig. 7.** Example OWL graph for *Context slices* pattern

kinds of indices to the reified relation (or context in this case), which overcomes e.g. the problem of dealing with multiple times, and the general perspectivization problem objected to 4D patterns.

With respect to *polymorphism*, this solution has the same simplicity as the 5.4 pattern: only one more axiom for each index is needed, attached to the context parameter. E.g. for the sentence *Garibaldi landed in Sicily in 1861 with 1000 soldiers*:

```
ExpeditionOfTheThousand with 1000Soldiers                (21)
```

However, when an index is bound to *parts* of a context, this solution needs to build a partonomy of contexts in order to deal with it. Such a partonomy may require a possibly elaborated reasoning in order to find out what is indexed in what part of a context, since some parts will be accessible by other parts, but others not. For example, `ExpeditionOfTheThousand` maximal context is a military campaign, in which Garibaldi's army numerosity has changed substantially through time, thus a `GaribaldiLanding` context that is part of `ExpeditionOfTheThousand` should be created, etc.

Similarly to pattern 5.5, this solution maintains the *relation footprint*, because the connection between $t$ and the relationship is a direct trickledown. DL reasoning is preserved, and closely approximates FOL semantics. The footprint can be reconstructed in RDF by means of a SPARQL query, e.g.:

```
SELECT ?r ?x ?y ?t
WHERE { ?xt ?r ?yt . ?x hasProjection ?xt . ?c existsAt ?t .
    ?y hasProjection ?yt . ?xt hasContext ?c . ?yt hasContext ?c }
```

Context slices has been proposed e.g. by Chris Welty's[8], based on the fluent representation described in [37]. A solution focused on temporal reasoning for OWL is presented in [5].

### 5.7 NameNesting

This solution implements the *name nesting* logical pattern, which results to be a common semantic web practice. As a sentence in FOL an example is:

```
landsInSicily(Garibaldi,1860)                            (22)
```

while in OWL2 (*name nesting logical pattern*), a sample model on the running example is shown in Figure 8.

This solution requires to nest the name of one of the arguments into the name of the property. Therefore, it works well when nested names are not essential to the knowledge to be represented.

An advantage of this solution is clearly the minimal amount of axioms needed. However, the expressivity is very limited, and seems recommendable only in contexts when one argument is much more prominent than others that can be

[8] http://ontologydesignpatterns.org/wiki/Submissions:Context_Slices

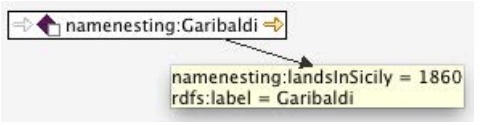



**Fig. 8.** Example OWL graph for *Name nesting* pattern

made implicit with name nesting, a form of "property localization". A realistic example is the property `blogEntryDate`, which nests the *blog* argument in the property name: as far as we only talk of entry dates in a specific blog, it works. Disadvantages with respect to the other solutions include not being able to talk about the situation itself, and not being able to add further arguments.

With respect to *polymorphism*, this solution needs more localized properties, e.g. for the sentence *Garibaldi landed in Sicily in 1861 with 1000 soldiers*:

```
Garibaldi landsInSicilyIn1860With 1000Soldiers          (23)
```

Name nesting disrupts the *relation footprint*, because the connection between $t$ and either the relationship or individuals is broken. DL reasoning is also very limited. The footprint cannot be reconstructed in RDF either.

## 6 Results and Discussion

In the presentation of the patterns, we have summarized data about the four qualitative dimensions in the design space: DL reasoning support for full n-ary sentences (f), polymorphism support (g), relation footprint (h), and intuitiveness (i): those data are also summarized in Table 1. The amount of axioms (a) dimension is reported in Table 2. The three purely computational dimensions: expressivity (b), consistency checking time (c), and classification time (d) are reported in Table 3. The last dimension: the amount of base and new constants (e) in the signature of each pattern, is reported in Table 4.

**Table 1.** Qualitative evaluation of OWL patterns on a three-value scale (good, limited, none)

| OWL pattern | DL reasoning | Polymorphism | Footprint | Intuitiveness |
|---|---|---|---|---|
| StatementContextualization | limited | none | none | good |
| StatementReification | limited | limited | limited | good |
| StatementAbstraction | limited | limited | limited | limited |
| Situation | good | good | good | good |
| Simple4D | good | limited | good | limited |
| ContextSlices | good | good | good | limited |
| NameNesting | limited/none | limited | none | good |

**Table 2.** Amount of OWL axioms needed for the representation of each pattern in function of the number $n$ of sentences represented in the ontology

| FOL pattern | OWL pattern | TBox axioms | ABox axioms | Annotation axioms | Total axioms |
|---|---|---|---|---|---|
| Sentence indexing (SI) | StatementContextualization | $c$ | $n$ | $n$ | $2n + c$ |
| Sentence indexing (SI) | StatementReification | $c$ | $4n$ | $n$ | $5n + c$ |
| Sentence indexing (SI) | StatementAbstraction | $4n$ | $n$ | 0 | $5n$ |
| Relation indexing | Situation | $c$ | $4n$ | 0 | $4n + c$ |
| Individual indexing | Simple4D | 0 | $7n$ | 0 | $7n$ |
| Individual indexing | ContextSlices | 0 | $9n$ | 0 | $9n$ |
| n/a | NameNesting | $n$ | 0 | 0 | $n$ |

**Table 3.** Expressivity of each OWL pattern and time taken for consistency checking and classification go the in-vitro knowledge base. Times are in *milliseconds*

| OWL Pattern | Expressivity | Cons Check Time Base | Classific. Time Base | Cons Check Time Artif | Classific. Time Artif | Artif Indiv | Artif Axioms |
|---|---|---|---|---|---|---|---|
| StatementContextualization | ELH | 18 | 11 | 68 | 11 | 876 | 1012 |
| StatementReification | ELH | 18 | 14 | 64 | 10 | 617 | 1017 |
| StatementAbstraction | ALHO(D) | 18 | 7 | n/a | n/a | 823 | 1019 |
| Situation | ALH(D) | 54 | 21 | 1824 | 48 | 650 | 1024 |
| Simple4D | ALH(D) | 18 | 7 | 54 | 7 | 528 | 1027 |
| ContextSlices | ALH(D) | 19 | 26 | 57 | 23 | 551 | 1035 |
| NameNesting | ALH(D) | 17 | 7 | 75 | 6 | 1001 | 1007 |

**Table 4.** Data about the constants needed for each OWL pattern. Reference FOL data is in the last row

| OWL Pattern | Class#base | Prop#base | Ind#base | NewIndividuals | NewProperties | NewClasses |
|---|---|---|---|---|---|---|
| StatementContextualization | 0 | 2 | 2 | 0 | 0 | 0 |
| StatementReification | 1 | 5 | 2 | 0 | 0 | 0 |
| StatementAbstraction | 1 | 3 | 3 | $n$ | $n$ | $2n$ |
| Situation | 2 | 3 | 3 | $n$ | 2 | 1 |
| Simple4D | 1 | 2 | 4 | $2n$ | 1 | 0 |
| ContextSlices | 2 | 4 | 5 | $3n$ | 2 | 2 |
| NameNesting | 1 | 1 | 1 | $-1n$ | 0 | 0 |
| *FOL* | 0 | 1 | 3 | 0 | 0 | 0 |

Among the nine design dimensions, the first five ((a) to (f)) (cf. data reported in Tables 2 and 3) are used to assess the *efficiency* and *expressivity* of a solution; (c) and (i) (cf. Tables 4 and 1) are used to assess its *usability*; (g) (cf. Table 1)is used to assess its *robustness and flexibility*; finally, (h) (cf. Table 1) is used to assess its potential for *interoperability*.

Concerning efficiency and expressivity, there is an uneven distribution of axioms, which e.g. in case of 1000 n-ary FOL sentences, would result in OWL models figures as different as 1000 for NameNesting to 9000 for ContextSlices pattern. StatementAbstraction is the only one to grow its TBox proportionally to the number of sentences, which we can expect to be computationally cumbersome, but actually that is not the case, at least on the base model[9]. The

[9] The artificial model cannot be generated for StatementAbstraction with SyGENiA because of the TBox generativity.

theoretical expressivity remains low on all solutions, which is basically confirmed by empirical results of consistency checking and classification on the base model, where the only clear increase (about 300 %) seems on the consistency checking time for the Situation pattern, which becomes worse with the artificial knowledge base, which has created 1000 artificial axioms and a variable number of individuals for each ontology implementing an example of a pattern. In terms of enabling DL reasoning over the entire n-ary sentence elements (cf. Table 1), three patterns are outstanding: Situation, ContextSlices, and Simple4D. This feature could actually correlate with longer times for reasoning tasks, although in this in vitro study such correlation is not completely proved.

Concerning usability, on one hand intuitiveness is highly debated. A common assumption is that individual-indexing-based patterns are less intuitive, because humans tend to reason by attributing some "endurant" qualities to objects, rather than thinking about them in terms of "slices". As [19] correctly notices, this could be due to cultural effects, since many scientists and manufacturers reason exactly in terms of slices, let alone the possibility that some features of noun-centric grammars of Western languages may induce an endurant effect. However, the ContextSlices pattern seems to be a good compromise, by minimizing the slices to the situations or contexts where objects are considered. On the other hand, the need to generate new terms in the signature of either a TBox or ABox is usually considered problematic for intuitiveness. Humans are accustomed to consider a conventionally finite amount of entities, and making them grow quite artificially with reification, abstraction, or slicing reduces usability. Also, semantic web toolkits are not quite proficient in smoothing the process of creating additional entities. Finally, linked data and semantic web developers are quite suspicious of entities that are created for the sake of design, rather than actually referenced in databases, texts, or social networks. The best patterns from this respect are StatementContextualization, StatementReification, and NameNesting, which seem actually the first choice in lightweight vocabularies and linked data, and typically preferred in the loose debates from the semantic technology blogosphere.

Concerning robustness/flexibility, polymorphism is poorly supported by 5 patterns out of 7. Only Situation and ContextSlices are flexible enough to make their knowledge bases grow linearly when new arguments are added to an n-ary relation. Situation is probably ideal here, because it is isomorphic to linguistic frame semantics, which is assumed to be closest to the cognitive structures that humans use in organizing their knowledge (cf. e.g. [13]).

Concerning interoperability, the relation footprint from the FOL sentences is still there in 5 patterns, which is good news for future unification algorithms: we have shown some simple SPARQL queries to extract the footprint from some patterns, and we can generate them from the footprint as well, so enabling a lightweight unification. Unfortunately, relation footprint tends to disappear when expressivity decreases, as is the case with StatementContextualization and NameNesting.

### 6.1 Comparison to LOD Data Patterns Evidence

Additional dimensions to the comparison between n-ary modeling patterns need to be added from empirical analysis of actual usage and subjective preference or intuitivity of the patterns. Leaving the second to ongoing experiments, for the first, we have some recent results that can be combined to our results.

The survey presented in [35] contains data about the usage of temporal information in Linked Open Data. They present 5 design patterns, represented over an abstraction of the RDF data model. There is an interesting and useful overlap with the patterns presented here: the authors distinguish between document-centric and fact-centric temporal perspectives on representing temporal information in RDF, the second being the one addressed in this paper as well. Among fact-centric patterns, they identify two "sentence-centric" patterns: *Reification* and *Applied temporal RDF*, and two "relationship-centric" patterns: *N-ary relationship*, and *4D-fluents*.

Their *Reification* corresponds to our *StatementReification* pattern, their *Applied temporal RDF* corresponds to our *StatementContextualization* pattern, their *N-ary relationship* corresponds to our *Situation* pattern, and their *4D-fluents* corresponds to our *Simple4D*, although the authors attribute their *4D-fluents* explicitly to [37], which is actually our *ContextSlices* pattern, but their example does not provide a context index, then making it a case of *Simple4D* instead. They do not search data patterns for *StatementAbstraction*, *ContextSlices*, and *NameNesting*.

The quantitative results of [35] indicate a neat preference of LOD for two patterns: the document-centric pattern called *Metadata-based representation*, which looks quite close to our *StatementContextualization* pattern (as well as to their *Applied temporal RDF* pattern), and the *N-ary relationship* (our *Situation*) pattern. The authors of [35] do not give data for *NameNesting*, which seems anecdotically well attested in LOD.

### 6.2 General Assessment

As a general assessment, more sophisticated patterns stand out on most criteria, however potential (anecdotically based) usability aspects and avoidance of newly introduced entities may work for the "unsophisticated" solutions such as StatementContextualization and NameNesting, which are however also the least interoperable patterns. Usage data from [35] are also ambivalent, since by far the most used data patterns are *Situation* and and *StatementContextualization*.

It is therefore easy to recognize, on objective grounds, the gap (cf. [22]) that has emerged when the linked data wave has hit the neat island of logically sound and well-designed ontologies. We can only hope for a reduction of the gap, partly with automatic reconciliation techniques, partly with the good practice of listening to the reasons and practices of the different parties.

## 7 Conclusion

We have compared seven design patterns for representing sentences with n-ary relations. We have used a multi-dimensional design space covering expressivity, efficiency, usability, flexibility, and interoperability aspects. An in-vitro study has been performed to assess the respective pros and cons in a fully comparable scenario. We have also combined our results with those of [35] obtained from empirical observation of linked data patterns. However, future work intends to bring this analysis and the methods emerging from it to in-vivo studies on human subjects.

Results basically confirm expectations about the divide between expressive, sophisticated patterns that enable deep reasoning, interoperability and smart knowledge engineering practices, and lightweight patterns that are limited in flexibility, but can be handy and usable to most developers of semantic technologies. A reduction of the gap can only come from the mutual understanding of the reasons why certain patterns tend to establish, or to live in a niche.

## References

1. Alexander, C.: The Timeless Way of Building. Oxford Press (1979)
2. Baader, F. (ed.): The Description Logic Handbook: theory, implementation, and applications. Cambridge University Press, Cambridge (2003)
3. Bach, E.: On Time, Tense, and Aspect: An Essay in English Metaphysics. In: Radical Pragmatics (1981)
4. Baker, C., Fillmore, C., Lowe, J.: The Berkeley FrameNet Project. In: Proceedings of the 17th International Conference on Computational Linguistics, Morristown, NJ, USA, pp. 86–90 (1998)
5. Batsakis, S., Petrakis, E.G.M.: SOWL: A Framework for Handling Spatio-Temporal Information in OWL 2.0. In: Bassiliades, N., Governatori, G., Paschke, A. (eds.) RuleML 2011 - Europe. LNCS, vol. 6826, pp. 242–249. Springer, Heidelberg (2011)
6. Blomqvist, E., Gangemi, A., Presutti, V.: Experiments on Pattern-Based Ontology Design. In: K-CAP. ACM (2009)
7. Blomqvist, E., Presutti, V., Daga, E., Gangemi, A.: Experimenting with eXtreme Design. In: Cimiano, P., Pinto, H.S. (eds.) EKAW 2010. LNCS, vol. 6317, pp. 120–134. Springer, Heidelberg (2010)
8. Carroll, J., Bizer, C., Hayes, P., Stickler, P.: Named Graphs, Provenance and Trust. In: WWW 2005, Chiba, Japan, May 10-14. ACM (2005)
9. Davidson, D.: The Logical Form of Action Sentences. In: The Logic of Decision and Action, 2nd edn. University of Pittsburgh Press, Pittsburgh (1967)
10. De Giacomo, G., Lenzerini, M., Rosati, R.: Towards Higher-Order DL-Lite. In: Proc. of DL 2008, Dresden, Germany (2008)
11. Galton, A.: Time and Change for AI. In: Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 4, pp. 175–240. Clarendon Press, Oxford (1995)
12. Gangemi, A.: Norms and Plans as Unification Criteria for Social Collectives. Journal of Autonomous Agents and Multi-Agent Systems 16(3) (2008)
13. Gangemi, A.: What's in a schema? A formal metamodel for ecg and framenet. In: Huang, C.-R., Calzolari, N., Gangemi, A., Lenci, A., Oltramari, A., Prévot, L. (eds.) Ontology and the Lexicon, Studies in Natural Language Processing. Cambridge University Press (2010)

14. Gangemi, A.: Super-duper schema: an owl2+rif dns pattern. In: Chaudry, V. (ed.) Proceedings of DeepKR Challenge Workshop at KCAP 2011 (2011)
15. Gangemi, A., Mika, P.: Understanding the Semantic Web through Descriptions and Situations. In: Meersman, R., Schmidt, D.C. (eds.) CoopIS/DOA/ODBASE 2003. LNCS, vol. 2888, pp. 689–706. Springer, Heidelberg (2003)
16. Gangemi, A., Presutti, V.: Towards a pattern science for the semantic web. Semantic Web 1(1-2), 61–68 (2010)
17. Guizzardi, G.: Ontological foundations for structural conceptual models. PhD thesis, University of Twente, Enschede, The Netherlands (2005)
18. Hayes, P.: 3D+1 (email message), http://ontolog.cim3.net/forum/ontologforum/2011-02/msg00009.html
19. Hayes, P.: Formal unifying standards for the representation of spatiotemporal knowledge. Technical report, IHMC (2004), http://www.ihmc.us/users/phayes/Trickledown2004.pdf
20. Hobbs, J.R.: Ontological Promiscuity. In: Proceedings of 23rd Annual Meeting of the Association for Computational Linguistics (1985)
21. Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: Yago2: A spatially and temporally enhanced knowledge base from wikipedia. Artificial Intelligence (to appear)
22. Jain, P., Hitzler, P., Yeh, P.Z., Vermay, K., Sheth, A.P.: Linked Data is Merely more Data. In: Proceedings of the AAAI Symposium Linked Data Meets Artificial Intelligence, pp. 82–86 (2010)
23. Kamp, H., Reyle, U.: From Discourse to Logic: Introduction to Model-theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory. In: Link, S., Prade, H. (eds.) Foundations of Information and Knowledge Systems. Studies in Linguistics and Philosophy, vol. 42. Kluwer, Dordrecht (1993)
24. Köhler, W.: Gestalt Psychology. Liveright, New York (1947)
25. Krieger, H.-U.: A Temporal Extension of the Hayes/ter Horst Entailment Rules and an Alternative to W3C's N-ary Relations. In: International Conference on Formal Ontology and Information Systems, FOIS 2012 (2012)
26. Masolo, C., Vieu, L., Bottazzi, E., Catenacci, C., Ferrario, R., Gangemi, A., Guarino, N.: Social roles and their descriptions. In: Welty, C., Dubois, D. (eds.) Proc. of Knowledge Representation and Reasoning (KR), pp. 267–277 (2004)
27. McCarthy, J.: Actions and other events in situation calculus. In: Proceedings of KR 2002 (2002)
28. Minsky, M.: A Framework for Representing Knowledge. In: Winston, P. (ed.) The Psychology of Computer Vision. McGraw-Hill (1975)
29. Moore, M.S.: Legal Reality: A Naturalist Approach to Legal Ontology. Law and Philosophy 21 (2002)
30. Motik, B.: On the properties of metamodeling in OWL. J. Log. Comput., 617–637 (2007)
31. Noy, N., Rector, A.: Defining N-ary Relations on the Semantic Web: Use With Individuals. Technical report, W3C (2005), http://www.w3.org/TR/swbp-naryRelations/
32. Nuzzolese, A.G., Gangemi, A., Presutti, V.: Gathering Lexical Linked Data and Knowledge Patterns from FrameNet. In: Proc. of the 6th International Conference on Knowledge Capture (K-CAP), Banff, Alberta, Canada, pp. 41–48 (2011)

33. Presutti, V., Draicchio, F., Gangemi, A.: Knowledge Extraction Based on Discourse Representation Theory and Linguistic Frames. In: ten Teije, A., Völker, J., Handschuh, S., Stuckenschmidt, H., d'Acquin, M., Nikolov, A., Aussenac-Gilles, N., Hernandez, N. (eds.) EKAW 2012. LNCS, vol. 7603, pp. 114–129. Springer, Heidelberg (2012)
34. Presutti, V., Gangemi, A.: Content Ontology Design Patterns as Practical Building Blocks for Web Ontologies. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 128–141. Springer, Heidelberg (2008)
35. Rula, A., Palmonari, M., Harth, A., Stadtmüller, S., Maurino, A.: On the Diversity and Availability of Temporal Information in Linked Open Data. In: Cudré-Mauroux, P., Heflin, J., Sirin, E., Tudorache, T., Euzenat, J., Hauswirth, M., Parreira, J.X., Hendler, J., Schreiber, G., Bernstein, A., Blomqvist, E. (eds.) ISWC 2012, Part I. LNCS, vol. 7649, pp. 492–507. Springer, Heidelberg (2012)
36. Shaw, R., Troncy, R., Hardman, L.: LODE: Linking Open Descriptions of Events. In: Gómez-Pérez, A., Yu, Y., Ding, Y. (eds.) ASWC 2009. LNCS, vol. 5926, pp. 153–167. Springer, Heidelberg (2009)
37. Welty, C., Fikes, R.: A Reusable Ontology of Fluents in OWL. In: Int. Conference on Formal Ontology and Information Systems. IOS Press (2006)

# Cognition-Enabled Autonomous Robot Control for the Realization of Home Chore Task Intelligence

Michael Beetz

University of Bremen, Germany

**Abstract.** This talk gives an overview of cognition-enabled robot control, a computational model for controlling autonomous service robots to achieve home chore task intelligence. For the realization of task intelligence, this computational model puts forth three core principles, which essentially involve the combination of reactive behavior specifications represented as semantically interpretable plans with inference mechanisms that enable flexible decision making. The representation of behavior specifications as plans enables the robot to not only execute the behavior specifications but also to reason about them and alter them during execution. I will provide a description of a complete system for cognition-enabled robot control that implements the three core principles, demonstrating the feasibility of our approach.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, p. 106, 2013.

# Searching Things in Large Sets of Images

Arnold Smeulders

University of Amsterdam, The Netherlands

**Abstract.** In this presentation we discuss the challenges of computer vision in general search through images and videos. The point of departure is the digital content of the images, a set of 50 to 100 examples, and smart features, machine learning and computational tools to find the images best answering a one-word specification. This problem is far from being solved, yet the progress as recorded in the yearly TRECvid competitions for image search engines is impressive. We will discuss the difference between the "where" and "what" in images, discuss the need for invariance, fast computational approaches, and the state of the art in image search engines.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, p. 107, 2013.

# SOS Rule Formats for Idempotent Terms and Idempotent Unary Operators*

Luca Aceto[1,2], Eugen-Ioan Goriac[1], and Anna Ingólfsdóttir[1]

[1] ICE-TCS, School of Computer Science, Reykjavik University, Iceland
[2] SysMA, IMT Lucca Institute for Advanced Studies, Lucca 55100, Italy

**Abstract.** A unary operator $f$ is idempotent if the equation $f(x) = f(f(x))$ holds. On the other end, an element $a$ of an algebra is said to be an idempotent for a binary operator $\odot$ if $a = a \odot a$. This paper presents a rule format for Structural Operational Semantics that guarantees that a unary operator be idempotent modulo bisimilarity. The proposed rule format relies on a companion one ensuring that certain terms are idempotent with respect to some binary operator.

## 1 Introduction

Over the last three decades, Structural Operational Semantics (SOS) [22] has proven to be a flexible and powerful way to specify the semantics of programming and specification languages. In this approach to semantics, the behaviour of syntactically correct language expressions is given in terms of a collection of state transitions that is specified by means of a set of syntax-driven inference rules. This behavioural description of the semantics of a language essentially tells one how the expressions in the language under definition behave when run on an idealized abstract machine.

Language designers often have expected algebraic properties of language constructs in mind when defining a language. For example, in the field of process algebras such as ACP [8], CCS [18] and CSP [15], operators such as nondeterministic and parallel composition are often meant to be commutative and associative with respect to bisimilarity [21]. Once the semantics of a language has been given in terms of state transitions, a natural question to ask is whether the intended algebraic properties do hold modulo the notion of behavioural semantics of interest. The typical approach to answer this question is to perform an *a posteriori verification*: based on the semantics in terms of state transitions, one proves the validity of the desired algebraic laws, which describe the expected semantic properties of the various operators in the language. An alternative approach is to ensure the validity of algebraic properties *by design*, using the so

* The authors have been partially supported by the project 'Meta-theory of Algebraic Process Theories' (No. 100014021) of the Icelandic Research Fund. Eugen-Ioan Goriac is also funded by the project 'Extending and Axiomatizing Structural Operational Semantics: Theory and Tools' (No. 1102940061) of the Icelandic Research Fund.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 108–120, 2013.

called *SOS rule formats* [6]. In this approach, one gives *syntactic templates* for the inference rules used in defining the operational semantics for certain operators that guarantee the validity of the desired laws, thus obviating the need for an a posteriori verification. (See [1,2,3,6,10,19] for examples of rule formats for algebraic properties in the literature on SOS.) The definition of SOS rule formats is based on finding a reasonably good trade-off between generality and ease of application. On the one hand, one strives to define a rule format that can capture as many examples from the literature as possible, including ones that may arise in the future. On the other, the rule format should be as easy to apply, and as syntactic, as possible.

The main advantage of the approach based on the development of rule formats is that one is able to verify the desired algebraic properties by syntactic checks that can be mechanized. Moreover, it is interesting to use rule formats for establishing semantic properties since the results so obtained apply to a broad class of languages. Last, but not least, these formats provide one with an understanding of the semantic nature of algebraic properties and of their connection with the syntax of SOS rules. This insight may serve as a guideline for language designers who want to ensure, a priori, that the constructs of a language under design enjoy certain basic algebraic properties.

*Contribution.* The main aim of this paper is to present a format of SOS rules that guarantees that some unary operation $f$ is *idempotent* with respect to any notion of behavioural equivalence that includes bisimilarity. A unary operator $f$ is idempotent if the equation $f(x) = f(f(x))$ holds. Examples of idempotent unary operators from the fields of language theory and process calculi are the unary Kleene star operator [16], the delay operator from SCCS [17], the replication operator from the $\pi$-calculus [24] and the priority operator from [7].

It turns out that, in order to develop a rule format for unary idempotent operations that can deal with operations such as Kleene star and replication, one needs a companion rule format ensuring that terms of a certain form are idempotent for some binary operator. We recall that an element $a$ of an algebra is said to be an *idempotent* for a binary operator $\odot$ if $a = a \odot a$. For example, the term $x^*$, where $^*$ denotes the Kleene star operation, is an idempotent for the sequential composition operation '$\cdot$' because the equation $x^* = x^* \cdot x^*$ holds. As a second contribution of this paper, we therefore offer an SOS rule format ensuring that certain terms are idempotent with respect to some binary operator. Both the rule formats we present in this paper make an essential use of previously developed formats for algebraic properties such as associativity and commutativity [10,19].

In [5], we provide a variety of examples showing that our rule formats can be used to establish the validity of several laws from the literature on process algebras dealing with idempotent unary operators and idempotent terms.

*Roadmap of the Paper.* The paper is organized as follows. Section 2 reviews some standard definitions from the theory of SOS that will be used in the remainder of this study. We present our rule format for idempotent terms in Section 3.

That rule format plays an important role in the definition of the rule format for idempotent unary operators that we give in Section 4. We discuss the results of the paper and hint at directions for future work in Section 5.

## 2 Preliminaries

In this section we review, for the sake of completeness, some standard definitions from process theory and the meta-theory of SOS that will be used in the remainder of the paper. We refer the interested reader to [4,20] for further details.

*Transition System Specifications in GSOS Format*

**Definition 1 (Signature, Terms and Substitutions).** *We let $V$ denote an infinite set of variables with typical members $x, x', x_i, y, y', y_i, \ldots$. A* signature *$\Sigma$ is a set of function symbols, each with a fixed arity. We call these symbols* operators *and usually represent them by $f, g, \ldots$. An operator with arity zero is called a* constant. *We define the set $\mathbb{T}(\Sigma)$ of* terms *over $\Sigma$ (sometimes referred to as $\Sigma$-terms) as the smallest set satisfying the following constraints.*

- *A variable $x \in V$ is a term.*
- *If $f \in \Sigma$ has arity $n$ and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.*

*We use $t, t', t_i, u, \ldots$ to range over terms. We write $t_1 \equiv t_2$ if $t_1$ and $t_2$ are syntactically equal. The function $vars : \mathbb{T}(\Sigma) \to 2^V$ gives the set of variables appearing in a term. The set $\mathbb{C}(\Sigma) \subseteq \mathbb{T}(\Sigma)$ is the set of* closed terms, *i.e., the set of all terms $t$ such that $vars(t) = \emptyset$. We use $p, p', p_i, q, \ldots$ to range over closed terms. A* context *is a term with an occurrence of a hole [ ] in it.*

*A* substitution *$\sigma$ is a function of type $V \to \mathbb{T}(\Sigma)$. We extend the domain of substitutions to terms homomorphically. If the range of a substitution is included in $\mathbb{C}(\Sigma)$, we say that it is a* closed substitution. *For a substitution $\sigma$ and sequences $x_1, \ldots, x_n$ and $t_1, \ldots, t_n$ of distinct variables and of terms, respectively, we write $\sigma[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ for the substitution that maps each variable $x_i$ to $t_i$ $(1 \leq i \leq n)$ and agrees with $\sigma$ on all of the other variables. When $\sigma$ is the identity function over variables, we abbreviate $\sigma[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$ to $[x_1 \mapsto t_1, \ldots, x_n \mapsto t_n]$.*

The GSOS format is a widely studied format of deduction rules in transition system specifications proposed by Bloom, Istrail and Meyer [9], which we now proceed to define.

**Definition 2 (GSOS Format [9]).** *A deduction rule for an operator $f$ of arity $n$ is in the* GSOS format *if and only if it has the following form:*

$$\frac{\{x_i \xrightarrow{l_{ij}} y_{ij} \mid 1 \leq i \leq n, 1 \leq j \leq m_i\} \cup \{x_i \xrightarrow{l_{ik}} \not\rightarrow \mid 1 \leq i \leq n, 1 \leq k \leq n_i\}}{f(\boldsymbol{x}) \xrightarrow{l} C[\boldsymbol{x}, \boldsymbol{y}]} \quad (1)$$

*where the $x_i$'s and the $y_{ij}$'s $(1 \leq i \leq n$ and $1 \leq j \leq m_i)$ are all distinct variables, $m_i$ and $n_i$ are natural numbers, $C[\boldsymbol{x}, \boldsymbol{y}]$ is a $\Sigma$-term with variables including at*

*most the $x_i$'s and $y_{ij}$'s, and $l_{ij}$'s and $l$ are labels. The above rule is said to be* $f$-defining *and* $l$-emitting.

*A* transition system specification (TSS) in the GSOS format $\mathcal{T}$ *is a triple* $(\Sigma, L, D)$ *where* $\Sigma$ *is a finite signature,* $L$ *is a finite set of labels, and* $D$ *is a finite set of deduction rules in the GSOS format. The collection of* $f$*-defining and* $l$*-emitting rules in a set* $D$ *of GSOS rules is denoted by* $D(f, l)$.

*Example 1.* An example of a TSS in the GSOS format is the one describing the semantics of BCCSP [14]. The signature for this TSS contains the operators $\mathbf{0}$ (of arity zero), $a.\_$ ($a \in L$) and $\_+\_$. The standard deduction rules for these operators are listed below, where $a$ ranges over $L$.

$$\frac{}{a.x_1 \xrightarrow{a} x_1} \qquad \frac{x_1 \xrightarrow{a} x_1'}{x_1 + x_2 \xrightarrow{a} x_1'} \qquad \frac{x_2 \xrightarrow{a} x_2'}{x_1 + x_2 \xrightarrow{a} x_1'}$$

Informally, the intent of a GSOS rule of the form (1) is as follows. Suppose that we are wondering whether $f(\boldsymbol{p})$ is capable of taking an $l$-step. We look at each $f$-defining and $l$-emitting rule in turn. We inspect each positive premise $x_i \xrightarrow{l_{ij}} y_{ij}$, checking if $p_i$ is capable of taking an $l_{ij}$-step for each $j$ and if so calling the $l_{ij}$-children $q_{ij}$. We also check the negative premises: if $p_i$ is *in*capable of taking an $l_{ik}$-step for each $k$. If so, then the rule *fires* and $f(\boldsymbol{p}) \xrightarrow{l} C[\boldsymbol{p}, \boldsymbol{q}]$. This means that the transition relation $\longrightarrow$ associated with a TSS in the GSOS format is the one defined by the rules using structural induction over closed $\Sigma$-terms. This transition relation is the unique sound and supported transition relation. Here *sound* means that whenever a closed substitution $\sigma$ 'satisfies' the premises of a rule of the form (1), then $\sigma(f(\boldsymbol{x})) \xrightarrow{l} \sigma(C[\boldsymbol{x}, \boldsymbol{y}])$. On the other hand, *supported* means that any transition $p \xrightarrow{l} q$ can be obtained by instantiating the conclusion of a rule of the form (1) with a substitution that satisfies its premises. We refer the interested reader to [9] for the precise definition of $\longrightarrow$ and much more information on GSOS languages. The above informal description of the transition relation associated with a TSS in GSOS format suffices to follow the technical developments in the remainder of the paper.

*Bisimilarity.* Terms built using operators from the signature of a TSS are usually considered modulo some notion of behavioural equivalence, which is used to indicate when two terms describe 'essentially the same behaviour'. The notion of behavioural equivalence that we will use in this paper is the following, classic notion of bisimilarity [18,21].

**Definition 3 (Bisimilarity).** *Let* $\mathcal{T}$ *be a TSS in GSOS format with signature* $\Sigma$*. A relation* $R \subseteq \mathbb{C}(\Sigma) \times \mathbb{C}(\Sigma)$ *is a* bisimulation *if and only if* $R$ *is symmetric and, for all* $p_0, p_1, p_0' \in \mathbb{C}(\Sigma)$ *and* $l \in L$,

$$(p_0 \ R \ p_1 \wedge p_0 \xrightarrow{l} p_0') \Rightarrow \exists p_1' \in \mathbb{C}(\Sigma).\ (p_1 \xrightarrow{l} p_1' \wedge p_0' \ R \ p_1').$$

*Two terms $p_0, p_1 \in \mathbb{C}(\Sigma)$ are called* bisimilar*, denoted by $\mathcal{T} \vdash p_0 \underline{\leftrightarrow} p_1$ (or simply by $p_0 \underline{\leftrightarrow} p_1$ when $\mathcal{T}$ is clear from the context), when there exists a bisimulation $R$ such that $p_0\ R\ p_1$. We refer to the relation $\underline{\leftrightarrow}$ as* bisimilarity*.*

It is well known that $\underline{\leftrightarrow}$ is an equivalence relation over $\mathbb{C}(\Sigma)$. Any equivalence relation $\sim$ over closed terms in a TSS $\mathcal{T}$ is extended to open terms in the standard fashion, i.e., for all $t_0, t_1 \in \mathbb{T}(\Sigma)$, the equation $t_0 = t_1$ holds over $\mathcal{T}$ modulo $\sim$ (sometimes abbreviated to $t_0 \sim t_1$) if, and only if, $\mathcal{T} \vdash \sigma(t_0) \sim \sigma(t_1)$ for each closed substitution $\sigma$.

**Definition 4.** *Let $\Sigma$ be a signature. An equivalence relation $\sim$ over $\Sigma$-terms is a* congruence *if, for all $f \in \Sigma$ and closed terms $p_1, \ldots, p_n, q_1, \ldots, q_n$, where $n$ is the arity of $f$, if $p_i \sim q_i$ for each $i \in \{1, \ldots, n\}$ then $f(p_1, \ldots, p_n) \sim f(q_1, \ldots, q_n)$.*

*Remark 1.* Let $\Sigma$ be a signature and let $\sim$ be a congruence. It is easy to see that, for all $f \in \Sigma$ and terms $t_1, \ldots, t_n, u_1, \ldots, u_n$, where $n$ is the arity of $f$, if $t_i \sim u_i$ for each $i \in \{1, \ldots, n\}$ then $f(t_1, \ldots, t_n) \sim f(u_1, \ldots, u_n)$.

The following result is well known [9].

**Proposition 1.** *$\underline{\leftrightarrow}$ is a congruence for any TSS in GSOS format.*

The above proposition is a typical example of a result in the meta-theory of SOS: it states that if the rules in a TSS satisfy some syntactic constraint, then some semantic result is guaranteed to hold. In the remainder of this paper, following the work presented in, e.g., [1,2,3,6,10,19], we shall present a rule format ensuring that certain unary operations are idempotent. This rule format will rely on one yielding that terms of a certain form are idempotent for some binary operator. For this reason, we present first the latter rule format in the subsequent section.

## 3 A Rule Format for Idempotent Terms

**Definition 5 (Idempotent Term).** *Let $\Sigma$ be a signature. Let $f$ and $\odot$ be, respectively, a unary and a binary operator in $\Sigma$. We say that $f(x)$ is an* idempotent term *for $\odot$ with respect to an equivalence relation $\sim$ over $\mathbb{T}(\Sigma)$ if the following equation holds:*

$$f(x) \sim f(x) \odot f(x). \tag{2}$$

In what follows, we shall present some syntactic requirements on the SOS rules defining the operators $f$ and $\odot$ that guarantee the validity of equation (2). In order to motivate the syntactic constraints of the rule format, let us consider the unary replication operator '!', which is familiar from the theory of the $\pi$-calculus (see, e.g., [24]), and the binary interleaving parallel composition '||', which appears in, amongst others, ACP [8], CCS [18], and CSP [15]. The rules for these operators are given below, where $a$ ranges over the set of action labels $L$.

$$\frac{x \xrightarrow{a} x'}{!x \xrightarrow{a} x' \,||\, !x} \qquad \frac{x \xrightarrow{a} x'}{x \,||\, y \xrightarrow{a} x' \,||\, y} \qquad \frac{y \xrightarrow{a} y'}{x \,||\, y \xrightarrow{a} x \,||\, y'} \tag{3}$$

It is well known that $!x$ is an idempotent term for $||$ modulo any notion of equivalence that includes bisimilarity. Indeed, the equation

$$!x = (!x) \,||(!x)$$

is one of the laws for the structural congruence over the $\pi$-calculus with replication considered in, e.g., [11].

It is instructive to try and find out why the above law holds by considering the interplay between the transition rules for '!' and those for '||', rather than considering the transitions that are possible for all the closed instantiations of the terms $!x$ and $(!x) \,||(!x)$. To this end, consider the rule for replication for some action $a$. The effect of this rule can be mimicked by the term $(!x) \,||(!x)$ by means of a combination of the instance of the first rule for $||$ in (3) for action $a$ and of the rule for replication. When we do so, the appropriate instantiation of the target of the conclusion of the first rule for $||$ is the term

$$(x' \,||\, y)[x' \mapsto x' \,||!x,\ y \mapsto !x] = (x' \,||!x) \,||!x.$$

Note that the target of the conclusion of the rule for replication, namely $x' \,||!x$, and the above term can be proved equal using associativity of $||$, which is well known, and the version of axiom (2) for replication and parallel composition, as follows:

$$(x' \,||!x) \,||!x = x' \,||(!x \,||!x) = x' \,||!x.$$

The validity of the associativity law for $||$ is guaranteed by the rule format for associativity given in [10, Definition 8]. On the other hand, the soundness of the use of equation (2) can be justified using coinduction [23].

Consider instead the combination of the instance of the second rule for $||$ in (3) for action $a$ and of the rule for replication. When we do so, the appropriate instantiation of the target of the conclusion of the second rule for $||$ is the term

$$(x \,||\, y')[x \mapsto !x,\ y \mapsto x' \,||!x] = !x \,||(x' \,||!x).$$

Note that the target of the conclusion of the rule for replication, namely $x' \,||!x$, and the above term can be proved equal using commutativity and associativity of $||$, which are well known, and the version of axiom (2) for replication and parallel composition, as follows:

$$!x \,||(x' \,||!x) = (x' \,||!x) \,||!x = x' \,||(!x \,||!x) = x' \,||!x.$$

The validity of the commutativity law for $||$ is guaranteed by the rule format for commutativity given in [19].

The above discussion hints at the possibility of defining an SOS rule format guaranteeing the validity of equation (2) building on SOS rule formats for algebraic properties like associativity and commutativity of operators [6], and on a coinductive use of equation (2) itself. The technical developments to follow will offer a formalization of this observation.

Our definition of the rule format is based on a syntactically defined equivalence relation over terms that is sufficient to handle the examples from the literature we have met so far.

**Definition 6 (The Relation $\leftrightsquigarrow$).** *Let $\mathcal{T} = (\Sigma, L, D)$ be a TSS in GSOS format.*

1. *The relation $\leftrightsquigarrow$ is the least equivalence relation over $\mathbb{T}(\Sigma)$ that satisfies the following clauses:*
   - *$f(t,u) \leftrightsquigarrow f(u,t)$, if $f$ is a binary operator in $\Sigma$ and the commutativity rule format from [19] applies to $f$,*
   - *$f(t, f(t',u)) \leftrightsquigarrow f(f(t,t'),u)$, if $f$ is a binary operator in $\Sigma$ and one of the associativity rule formats from [10] applies to $f$, and*
   - *$C[t] \leftrightsquigarrow C[t']$, if $t \leftrightsquigarrow t'$, for each context $C[\ ]$.*
2. *Let $f$ and $\odot$ be, respectively, a unary and a binary operator in $\Sigma$. We write $t \downarrow_{f,\odot} u$ if, and only if, there are some $t'$ and $u'$ such that $t \leftrightsquigarrow t'$, $u \leftrightsquigarrow u'$, and $t' = u'$ can be proved by possibly using one application of an instantiation of axiom (2) in a context—that is, either $t' \equiv u'$, or $t' = C[f(t'')]$ and $u' = C[f(t'') \odot f(t'')]$, for some context $C[\ ]$ and term $t''$, or vice versa.*

*Example 2.* Consider the terms $!x \,||(x' \,||!x)$ and $x' \,||!x$. Then

$$x' \,||!x \ \downarrow_{!,||} \ !x \,||(x' \,||!x).$$

Indeed, $!x \,||(x' \,||!x) \leftrightsquigarrow x' \,||(!x \,||!x)$, because the rules for $||$ are in the associativity and commutativity rule formats from [10,19], and $x' \,||!x = x' \,||(!x \,||!x)$ can be proved using one application of the relevant instance of axiom (2) in the context $x' \,||[\ ]$.

We are now ready to present an SOS rule format guaranteeing the validity of equation (2).

**Definition 7 (Rule Format for Idempotent Terms).** *Let $\mathcal{T} = (\Sigma, L, D)$ be a TSS in the GSOS format. Let $f$ and $\odot$ be, respectively, a unary and a binary operator in $\Sigma$. We say that the rules for $f$ and $\odot$ in $\mathcal{T}$ are in the* rule format for idempotent terms *if either $\odot$ is in the rule format for idempotence from [1] or the following conditions are met:*

1. *For each $f$-defining rule in $D$, say $\dfrac{H}{f(x) \xrightarrow{a} t}$, there is some $\odot$-defining rule $\dfrac{H'}{x_1 \odot x_2 \xrightarrow{a} u}$, such that*
   *(a) $H' \subseteq \{x_1 \xrightarrow{a} y_1,\ x_2 \xrightarrow{a} y_2\}$, and*
   *(b) $t \downarrow_{f,\odot} u[x_1 \mapsto f(x), x_2 \mapsto f(x), y_1 \mapsto t, y_2 \mapsto t]$.*
2. *Each $\odot$-defining rule has the form*

$$\frac{\{x_i \xrightarrow{a} y_i\} \cup \{x_1 \xrightarrow{a_j} y_j \mid j \in J\} \cup \{x_2 \xrightarrow{b_k} z_k \mid k \in K\}}{x_1 \odot x_2 \xrightarrow{a} u}, \tag{4}$$

*where $i \in \{1,2\}$, $J$ and $K$ are index sets, and $vars(u) \subseteq \{x_1, x_2, y_i\}$.*

3. *Let $r$ be an $\odot$-defining rule of the form (4) such that $D(f, a_j) \neq \emptyset$, for each $j \in J$, and $D(f, b_k) \neq \emptyset$, for each $k \in K$. Let $\dfrac{H}{f(x) \xrightarrow{a} t}$ be a rule for $f$. Then*

$$t \downarrow_{f,\odot} u[x_1 \mapsto f(x), x_2 \mapsto f(x), y_i \mapsto t].$$

**Theorem 1.** *Let $\mathcal{T} = (\Sigma, L, D)$ be a TSS in the GSOS format. Let $f$ and $\odot$ be, respectively, a unary and a binary operator in $\Sigma$. Assume that the rules for $f$ and $\odot$ in $\mathcal{T}$ are in the rule format for idempotent terms. Then equation (2) holds over $\mathcal{T}$ modulo bisimilarity.*

The proof of this result may be found in [5].

*Example 3 (Replication and Parallel Composition).* The unary replication operator and the parallel composition operator, whose rules we presented in (3), are in the rule format for idempotent terms. Indeed, we essentially carried out the verification of the conditions in Definition 7 when motivating the constraints of the rule format and the relation $\leftrightsquigarrow$. Therefore, Theorem 1 yields the soundness, modulo bisimilarity, of the well-known equation

$$!x = (!x) \,||(!x).$$

We give further examples of application of the rule format in [5].

## 4 A Rule Format for Idempotent Unary Operators

**Definition 8 (Idempotent Unary Operator).** *Let $\Sigma$ be a signature. Let $f$ be a unary operator in $\Sigma$. We say that $f(x)$ is* idempotent *with respect to an equivalence relation $\sim$ over $\mathbb{T}(\Sigma)$ if the following equation holds:*

$$f(x) \sim f(f(x)). \tag{5}$$

In what follows, we shall present some syntactic requirements on the SOS rules defining a unary operator $f$ that guarantee the validity of equation (5). The rule format for idempotent unary operators will rely on the one for idempotent terms given in Definition 7.

In order to motivate the use of the rule format for idempotent terms in the definition of the one for idempotent unary operators, consider the replication operator whose rules were introduced in (3). As is well known, the equation $!x =!(!x)$ holds modulo bisimilarity. The validity of this equation can be 'justified' using the transition rules for '!' as follows. Consider the rule for replication for some action $a$, namely $\dfrac{x \xrightarrow{a} x'}{!x \xrightarrow{a} x' \,||!x}$. The effect of this rule can be mimicked by the term $!(!x)$ by using the same rule twice. When we do so, the appropriate instantiation of the target of the conclusion of the rule for '!' is the term

$$(x' \,||!x)[x' \mapsto x' \,||!x,\ x \mapsto !x] = (x' \,||!x) \,||!(!x).$$

Note that the target of the conclusion of the rule for replication, namely $x' \,||\, !x$, and the above term can be proved equal using associativity of $||$, the version of axiom (2) for replication and parallel composition, and the version of axiom (5) for replication, as follows:

$$(x' \,||\, !x) \,||\, !(!x) = (x' \,||\, !x) \,||\, !x = x' \,||\, (!x \,||\, !x) = x' \,||\, !x.$$

As mentioned in Example 3, the validity of the version of axiom (2) for replication and parallel composition is guaranteed by Theorem 1. On the other hand, the soundness of the use of equation (5) can be justified using coinduction [23].

As we did in the definition of the rule format for idempotent terms presented in Definition 7, in stating the requirements of the rule format for idempotent unary operators, we shall employ a syntactically defined equivalence relation over terms that is sufficient to handle the examples from the literature we have met so far.

**Definition 9 (The Relation $\leftrightarrow$).** *Let $\mathcal{T} = (\Sigma, L, D)$ be a TSS in the GSOS format.*

1. *The relation $\leftrightarrow$ is the least equivalence relation over $\mathbb{T}(\Sigma)$ that satisfies the following clauses:*
   - *$f(t) \leftrightarrow f(t) \odot f(t)$, if the rules for $f$ and $\odot$ in $\mathcal{T}$ are in the rule format for idempotent terms from Definition 7,*
   - *$f(t,u) \leftrightarrow f(u,t)$, if $f$ is a binary operator in $\Sigma$ and the commutativity rule format from [19] applies to $f$,*
   - *$f(t, f(t',u)) \leftrightarrow f(f(t,t'),u)$, if $f$ is a binary operator in $\Sigma$ and one of the associativity rule formats from [10] applies to $f$, and*
   - *$C[t] \leftrightarrow C[t']$, if $t \leftrightarrow t'$, for each context $C[\,]$.*
2. *Let $f$ be a unary operator in $\Sigma$. We write $t \Downarrow_f u$ if, and only if, there are some $t'$ and $u'$ such that $t \leftrightarrow t'$, $u \leftrightarrow u'$, and $t' = u'$ can be proved by possibly using one application of an instantiation of axiom (5) in a context—that is, either $t' \equiv u'$, or $t' = C[f(t'')]$ and $u' = C[f(f(t''))]$, for some context $C[\,]$ and term $t''$, or vice versa.*

*Example 4.* Consider the terms $(x' \,||\, !x) \,||\, !(!x)$ and $x' \,||\, !x$. Then

$$x' \,||\, !x \Downarrow_! (x' \,||\, !x) \,||\, !(!x).$$

Indeed, $x' \,||\, !x \leftrightarrow (x' \,||\, !x) \,||\, !x$, using the relevant instance of axiom (2) and associativity of $||$, and $(x' \,||\, !x) \,||\, !x = (x' \,||\, !x) \,||\, !(!x)$ can be proved using one application of the relevant instance of axiom (5) in the context $(x' \,||\, !x) \,||\, [\,]$.

**Definition 10 (Rule Format for Idempotent Unary Operators).** *Let $\mathcal{T} = (\Sigma, L, D)$ be a TSS in the GSOS format. Let $f$ be a unary operator in $\Sigma$. We say that the rules for $f$ are in the rule format for* idempotent unary operators *if the following conditions are met:*

1. *Each rule for $f$ in $D$ has the form*

$$\frac{H \cup \{x \overset{b_j}{\not\longrightarrow} \mid j \in J\}}{f(x) \xrightarrow{a} t}, \tag{6}$$

   *where*
   (a) $H \subseteq \{x \xrightarrow{a} x'\}$ *and*
   (b) $H = \{x \xrightarrow{a} x'\}$ *if $J$ is non-empty.*
2. *If some rule for $f$ of the form (6) has a premise of the form $x \overset{b}{\not\longrightarrow}$, then each $b$-emitting and $f$-defining rule has a positive premise of the form $x \xrightarrow{b} x'$.*
3. *Consider a rule for $f$ of the form (6). Then*
   (a) *either $H$ is empty and $t \Downarrow_f t[x \mapsto f(x)]$*
   (b) *or $H = \{x \xrightarrow{a} x'\}$ and, for each $a$-emitting rule for $f$ $\dfrac{H'}{f(x) \xrightarrow{a} t'}$, we have that $t' \Downarrow_f t[x \mapsto f(x), x' \mapsto t']$.*

**Theorem 2.** *Let $\mathcal{T} = (\Sigma, L, D)$ be a TSS in the GSOS format. Let $f$ be a unary operator in $\Sigma$. Assume that the rules for $f$ in $\mathcal{T}$ are in the rule format for idempotent unary operators. Then equation (5) holds over $\mathcal{T}$ modulo bisimilarity.*

The proof of this result may be found in [5].

*Remark 2.* Condition 1b in Definition 10 requires that, in rules of the form (6), $H = \{x \xrightarrow{a} x'\}$ if $J$ is non-empty. This requirement is necessary for the validity of Theorem 2. To see this, consider the unary operator $f$ with rules

$$\frac{x \overset{b}{\not\longrightarrow}}{f(x) \xrightarrow{a} \mathbf{0}} \qquad \frac{x \xrightarrow{b} x',\ x \overset{c}{\not\longrightarrow}}{f(x) \xrightarrow{b} \mathbf{0}} \qquad \frac{x \xrightarrow{c} x',\ x \overset{b}{\not\longrightarrow}}{f(x) \xrightarrow{c} \mathbf{0}}.$$

The rules for $f$ satisfy all the conditions in Definition 10 apart from the requirement in condition 1b that all rules have positive premises when they have negative ones.

It is not hard to see that $f(b.\mathbf{0} + c.\mathbf{0})$ has no outgoing transitions. On the other hand, using the $a$-emitting rule for $f$, we have that $f(f(b.\mathbf{0} + c.\mathbf{0})) \xrightarrow{a} \mathbf{0}$. Therefore, $f(b.\mathbf{0}+c.\mathbf{0}) \underline{\leftrightarrow} f(f(b.\mathbf{0}+c.\mathbf{0}))$, and the equation $f(x) \underline{\leftrightarrow} f(f(x))$ does not hold.

*Example 5 (Priority).* Assume that $<$ is an irreflexive partial ordering over $L$. The priority operator $\theta$ from [7] has rules

$$\frac{x \xrightarrow{a} x',\ (x \overset{b}{\not\longrightarrow} \text{ for each } b \text{ such that } a < b)}{\theta(x) \xrightarrow{a} \theta(x')} \qquad (a \in L).$$

It is not hard to see that the rules for $\theta$ satisfy the conditions in Definition 10. In particular, for each $a \in L$, the only $a$-emitting rule for $\theta$ has a positive premise of the form $x \xrightarrow{a} x'$. Hence, condition 1b in Definition 10 is met.

Therefore, Theorem 2 yields the soundness, modulo bisimilarity, of the well-known equation $\theta(x) = \theta(\theta(x))$.

We give further examples of application of the rule format in [5].

## 5 Conclusions

In this study, we have presented an SOS rule format that guarantees that a unary operator is idempotent modulo bisimilarity. In order to achieve a sufficient degree of generality, that rule format relies on a companion one ensuring that certain terms are idempotent with respect to some binary operator. In addition, both rule formats make use of existing formats for other algebraic properties such as associativity [10], commutativity [19] and idempotence for binary operators [1]. In this paper, we have restricted ourselves to TSSs in GSOS format [9] for the sake of simplicity. The rule formats we offered in this study can be extended to arbitrary TSSs in standard fashion, provided one gives the semantics of such TSSs in terms of three-valued stable models.

The auxiliary rule format ensuring that certain terms are idempotent with respect to some binary operator may be seen as a refinement of the one from [1]. That paper offered a rule format guaranteeing that certain binary operators are idempotent. We recall that a binary operator $\odot$ is idempotent if the equation $x \odot x = x$ holds. Of course, if a binary operation is idempotent, then any term is an idempotent for it. However, the sequential composition operator '$\cdot$' is *not* idempotent, but the term $x^*$ is an idempotent for it. Similarly, the parallel composition operator '$||$' is *not* idempotent, but the term $!x$ is an idempotent for $||$. Since the laws $x^* \cdot x^* = x^*$ and $!x \,||\, !x = !x$ play an important role in establishing, via syntactic means, that the unary Kleene star and replication operators are idempotent, we needed to develop a novel rule format for idempotent terms in order to obtain a powerful rule format for idempotent unary operations.

To our mind, idempotence of unary operators is the last 'typical' algebraic law for which it is worth developing a specialized rule format. An interesting, long-term research goal is to develop a general approach for synthesizing rule formats for algebraic properties from the algebraic law itself and some assumption on the format of the rules used to give the semantics for the language constructs in the style of SOS. We believe that this is a hard research problem. Indeed, the development of the formats for algebraic properties surveyed in [6] has so far relied on ad-hoc ingenuity and it is hard to discern some common principles that could guide the algorithmic synthesis of such formats.

## References


1. Aceto, L., Birgisson, A., Ingólfsdóttir, A., Mousavi, M.R., Reniers, M.A.: Rule formats for determinism and idempotence. Science of Computer Programming 77(7-8), 889–907 (2012)
2. Aceto, L., Cimini, M., Ingólfsdóttir, A., Mousavi, M.R., Reniers, M.A.: SOS rule formats for zero and unit elements. Theoretical Computer Science 412(28), 3045–3071 (2011)
3. Aceto, L., Cimini, M., Ingólfsdóttir, A., Mousavi, M.R., Reniers, M.A.: Rule formats for distributivity. Theoretical Computer Science 458, 1–28 (2012)
4. Aceto, L., Fokkink, W., Verhoef, C.: Structural operational semantics. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, ch. 3, pp. 197–292. Elsevier Science, Dordrecht (2001)
5. Aceto, L., Goriac, E.I., Ingólfsdóttir, A.: SOS rule formats for idempotent terms and idempotent unary operators (2012), http://www.ru.is/faculty/luca/PAPERS/idemop.pdf
6. Aceto, L., Ingólfsdóttir, A., Mousavi, M.R., Reniers, M.A.: Algebraic properties for free! Bulletin of the European Association for Theoretical Computer Science 99, 81–104 (2009)
7. Baeten, J., Bergstra, J., Klop, J.W.: Syntax and defining equations for an interrupt mechanism in process algebra. Fundamenta Informaticae IX(2), 127–168 (1986)
8. Baeten, J., Basten, T., Reniers, M.A.: Process Algebra: Equational Theories of Communicating Processes. Cambridge Tracts in Theoretical Computer Science, vol. 50. Cambridge University Press (2009)
9. Bloom, B., Istrail, S., Meyer, A.R.: Bisimulation can't be traced. Journal of the ACM 42(1), 232–268 (1995)
10. Cranen, S., Mousavi, M.R., Reniers, M.A.: A Rule Format for Associativity. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 447–461. Springer, Heidelberg (2008)
11. Engelfriet, J., Gelsema, T.: Multisets and structural congruence of the pi-calculus with replication. Theoretical Computer Science 211(1-2), 311–337 (1999)
12. Fokkink, W.: A complete equational axiomatization for prefix iteration. Information Processing Letters 52(6), 333–337 (1994)
13. Fokkink, W.: Axiomatizations for the Perpetual Loop in Process Algebra. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 571–581. Springer, Heidelberg (1997)
14. van Glabbeek, R.J.: The linear time - branching time spectrum I. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, ch. 1, pp. 3–100. Elsevier Science, Dordrecht (2001)
15. Hoare, C.A.R.: Communicating Sequential Processes. Prentice Hall (1985)
16. Kleene, S.: Representation of events in nerve nets and finite automata. In: Shannon, C., McCarthy, J. (eds.) Automata Studies, pp. 3–41. Princeton University Press (1956)
17. Milner, R.: Calculi for synchrony and asynchrony. Theoretical Computer Science 25, 267–310 (1983)
18. Milner, R.: Communication and Concurrency. Prentice Hall (1989)
19. Mousavi, M.R., Reniers, M.A., Groote, J.F.: A syntactic commutativity format for SOS. Information Processing Letters 93, 217–223 (2005)
20. Mousavi, M.R., Reniers, M.A., Groote, J.F.: SOS formats and meta-theory: 20 years after. Theoretical Computer Science 373, 238–272 (2007)

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, 17–139 (2004)
23. Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge University Press (2011)
24. Sangiorgi, D., Walker, D.: The $\pi$-Calculus: A Theory of Mobile Processes. Cambridge University Press (2001)

# Worst Case Analysis of Non-local Games*

Andris Ambainis[1], Artūrs Bačkurs[2], Kaspars Balodis[1], Agnis Škuškovniks[1], Juris Smotrovs[1], and Madars Virza[2]

[1] Faculty of Computing, University of Latvia, Raina bulv. 19, Riga, LV-1586, Latvia
{andris.ambainis,Juris.Smotrovs}@lu.lv,
{kbalodis,agnis.skuskovniks}@gmail.com
[2] Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 32 Vassar Street, Cambridge, MA 02139, USA
{abackurs,madars}@gmail.com

**Abstract.** Non-local games are studied in quantum information because they provide a simple way for proving the difference between the classical world and the quantum world. A non-local game is a cooperative game played by 2 or more players against a referee. The players cannot communicate but may share common random bits or a common quantum state. A referee sends an input $x_i$ to the $i^{\text{th}}$ player who then responds by sending an answer $a_i$ to the referee. The players win if the answers $a_i$ satisfy a condition that may depend on the inputs $x_i$.

Typically, non-local games are studied in a framework where the referee picks the inputs from a known probability distribution. We initiate the study of non-local games in a worst-case scenario when the referee's probability distribution is unknown and study several non-local games in this scenario.

## 1 Overview

Quantum mechanics is strikingly different from classical physics. In the area of information processing, this difference can be seen through quantum algorithms which can be exponentially faster than conventional algorithms [18,16] and through quantum cryptography which offers degree of security that is impossible classically [7].

Another way of seeing the difference between quantum mechanics and the classical world is through non-local games. An example of non-local game is the CHSH (Clauser-Horne-Shimonyi-Holt) game [11]. This is a game played by two parties against a referee. The two parties cannot communicate but can share common randomness or common quantum state that is prepared before the beginning of the game. The referee prepares two uniformly random bits $x, y$ and gives one of them to each of two parties. The parties reply by sending bits $a$ and $b$ to the referee. They win if $a \oplus b = x \wedge y$. The maximum winning probability that can be achieved is 0.75 classically and $\frac{1}{2} + \frac{1}{2\sqrt{2}} = 0.85...$ quantumly.

* Supported by ESF project 2009/0216/1DP/1.1.1.2.0/09/APIA/VIAA/044 and FP7 FET-Open project QCS.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 121–132, 2013.

Other non-local games can be obtained by changing the winning conditions, replacing bits $x, y$ with $x, y \in \{1, \ldots, m\}$ or changing the number of parties. The common feature is that all non-local games involve parties that cannot communicate but can share common random bits or a common quantum state.

There are several reasons why non-local games are interesting. First, CHSH game provides a very simple example to test validity of quantum mechanics. If we have implemented the referee and the two players by devices so that there is no communication possible between $A$ and $B$ and we observe the winning probability of 0.85..., there is no classical explanation possible. Second, non-local games have been used in device-independent cryptography [1,17].

Non-local games are typically analyzed with the referee acting according to some probability distribution. E. g., for the CHSH game, the referee chooses each of possible pairs of bits $(0, 0)$, $(0, 1)$, $(1, 0)$, $(1, 1)$ as $(x, y)$ with equal probabilities $1/4$. This is quite natural if we think of the CHSH game as an experiment for testing the validity of quantum mechanics. Then, we can implement the device for the referee so that it uses the appropriate probability distribution.

On the other hand, most of theoretical computer science is based on the worst-case analysis of algorithms, including areas such as quantum communication complexity [20] and distributed computing [13] (which are both related to non-local games). Because of that, we think that it is also interesting to study non-local games in a worst case setting, when the players have to achieve winning probability at least $p$ for every possible combination of input data $(x, y)$[1].

In this paper, we start a study of non-local games in the worst-case framework. We start with several simple observations (section 3). First, the maximum gap between quantum and classical winning probability in the worst-case scenario is at most the maximum gap for a fixed probability distribution. Second, many of the non-local games that achieve the biggest gaps for a fixed probability distribution (such as CHSH game for 2-player XOR games or Mermin-Ardehali game [15,5] for $n$-player XOR games) also achieve the same gap in the worst-case scenario, due to natural symmetries present in those games.

Then, in section 4, we look at examples of non-local games for which the worst case is different from the average case under the most natural probability distribution, with two goals. First, we show natural examples of non-local games for which the worst-case behaviour is not a straightforward consequence of the average-case behaviour under the uniform distribution.

Second, at the same time, we develop methods for analyzing non-local games in the worst case. For non-local games under a fixed probability distribution, computing the best winning probability is at least NP-hard [14] in the general case but there are efficient algorithms for fairly broad special cases (such as 2-player XOR games [12]). Those algorithms crucially rely on the fact that non-local games are studied under a fixed probability distribution on inputs $(x, y)$. This allows to reduce the maximum winning probability to a simple expression

[1] Also, when we give talks about non-local games to computer scientists who are not familiar with quantum computing, we often get a question: why don't you consider the worst case setting?

whose maximum can be computed by a polynomial time algorithm. (For example, for 2-player XOR games, this is done via semidefinite programming [12].)

These methods no longer work in the worst case scenario, where we have to develop new methods on case-by-case basis - for games that would have been easy to analyze with previous methods if the probability distribution was fixed.

## 2 Technical Preliminaries

We will study non-local games of the following kind [12] in both classical and quantum settings. There are $n$ cooperating players $A_1$, $A_2$, ..., $A_n$ trying to maximize the game value (see below), and there is a referee. Before the game the players may share a common source of correlated random data: in the classical case, a common random variable $R$ taking values in a finite set $\mathcal{R}$, and in the quantum case, an entangled $n$-part quantum state $|\psi\rangle \in \mathcal{A}_1 \otimes \ldots \otimes \mathcal{A}_n$ (where $\mathcal{A}_i$ is a finite-dimensional subspace corresponding to the part of the state available to $A_i$). During the game the players cannot communicate between themselves.

Each of the players ($A_i$) has a finite set of possible input data: $X_i$. At the start of the game the referee randomly picks values $(x_1, \ldots, x_n) = \mathbf{x} \in X_1 \times \ldots \times X_n$ according to some probability distribution $\pi$, and sends each of the players his input (i. e. $A_i$ receives $x_i$).

Each of the players then must send the referee a response $a_i$ which may depend on the input and the common random data source. In this paper we will consider only *binary games*, that is games where the responses are simply bits: $a_i \in \{0, 1\}$. We denote $(a_1, \ldots, a_n)$ by $\mathbf{a}$.

The referee checks whether the players have won by some predicate (known to all parties) depending on the players' inputs and outputs: $V(\mathbf{a} \mid \mathbf{x})$. For convenience in formulas, we will suppose that $V$ takes value 1 when it is *true* and $-1$ when it is *false*. A binary game whose outcome depends only on the $XOR$ of the players' responses: $V(\mathbf{a} \mid \mathbf{x}) = V'(\bigoplus_{i=1}^{n} a_i \mid \mathbf{x})$, is called an *XOR game*. A game whose outcome does not change after any permutation $\gamma$ of the players (i. e. $V(\gamma(\mathbf{a}) \mid \gamma(\mathbf{x})) = V(\mathbf{a} \mid \mathbf{x})$ for any $\gamma$) is called a *symmetric game*.

The value $\omega$ of a non-local game $G$ for given players' strategies is the difference between the probability that the players win and the probability that they lose:

$$\omega(G) = \Pr[V(\mathbf{a} \mid \mathbf{x}) = 1] - \Pr[V(\mathbf{a} \mid \mathbf{x}) = -1] \in [-1, 1].$$

The probability that the players win can then be expressed by the game value in this way: $\Pr[V(\mathbf{a} \mid \mathbf{x}) = 1] = \frac{1}{2} + \frac{1}{2}\omega(G)$.

In the classical case, the players' strategy is the random variable $R$ and a set of functions $a_i : X_i \times \mathcal{R} \to \{0, 1\}$ determining the responses. The maximal classical game value achievable by the players for a given distribution $\pi$ is thus:

$$\omega_c^{\pi}(G) = \sup_{R,\mathbf{a}} \sum_{r,\mathbf{x}} \pi(\mathbf{x}) \Pr[R = r] V(a_1(x_1, r), \ldots, a_n(x_n, r) \mid \mathbf{x}).$$

However, the use of random variable here is redundant, since in the expression it provides a convex combination of deterministic strategy game values, thus the maximum is achieved by some deterministic strategy (with $a_i : X_i \to \{0, 1\}$):

$$\omega_c^\pi(G) = \max_{\mathbf{a}} \sum_{\mathbf{x}} \pi(\mathbf{x}) V(a_1(x_1), \ldots, a_n(x_n) \mid \mathbf{x}).$$

In this paper we investigate the case when the players do not know the probability distribution $\pi$ used by the referee, and must maximize the game value for the worst distribution the referee could choose, given the strategy picked by the players. We will call it the worst-case game value. The maximal classical worst-case game value $\omega_c$ achievable by the players is given by the formula

$$\omega_c(G) = \sup_{R,\mathbf{a}} \min_{\pi} \sum_{r,\mathbf{x}} \pi(\mathbf{x}) \Pr[R = r] V(a_1(x_1, r), \ldots, a_n(x_n, r) \mid \mathbf{x}).$$

Note that in the worst-case approach the optimal strategy cannot be a deterministic one, unless there is a deterministic strategy winning on *all* inputs: if there is an input on which the strategy loses, then the referee can supply it with certainty, and the players always lose. Clearly, $\omega_c(G) \leq \omega_c^\pi(G)$ for any $\pi$.

In the most of the studied examples $\pi$ has been the uniform distribution. We will call it the average case and denote its maximum game value by $\omega_c^{\mathrm{uni}}(G)$.

In the quantum case, the players' strategy is the state $|\psi\rangle$ and the measurements that the players pick depending on the received inputs and perform on their parts of $|\psi\rangle$ to determine their responses. Mathematically, the measurement performed by $A_i$ after receiving input $x_i$ is a pair of positive semidefinite $\dim \mathcal{A}_i$-dimensional matrices $M_i^{0|x_i}$, $M_i^{1|x_i}$ with $M_i^{0|x_i} + M_i^{1|x_i} = I$ where $I$ is the identity matrix. We denote the collection of all measurements by $\mathbf{M}$.

The maximum quantum game value for a fixed distribution $\pi$ is

$$\omega_q^\pi(G) = \sup_{|\psi\rangle,\mathbf{M}} \sum_{\mathbf{x},\mathbf{a}} \pi(\mathbf{x}) \langle\psi| \bigotimes_{i=1}^{n} M_i^{a_i|x_i} |\psi\rangle V(\mathbf{a} \mid \mathbf{x}),$$

and the maximum quantum worst-case game value is

$$\omega_q(G) = \sup_{|\psi\rangle,\mathbf{M}} \min_{\pi} \sum_{\mathbf{x},\mathbf{a}} \pi(\mathbf{x}) \langle\psi| \bigotimes_{i=1}^{n} M_i^{a_i|x_i} |\psi\rangle V(\mathbf{a} \mid \mathbf{x}).$$

Since the shared entangled state can be used to simulate a random variable, $\omega_q(G) \geq \omega_c(G)$ and for any $\pi$: $\omega_q^\pi(G) \geq \omega_c^\pi(G)$.

In the case of two player games ($n = 2$) we will use notation $A, B$ for the players, $X, Y$ for the input sets, $x, y$ for the inputs, $a, b$ for the responses.

## 3 Games With Worst Case Equivalent to Average Case

### 3.1 Maximum Quantum-Classical Gap

The advantage of quantum strategies is usually measured by the ratio $\frac{\omega_q^\pi(G)}{\omega_c^\pi(G)}$ (or $\frac{\omega_q(G)}{\omega_c(G)}$ in the worst-case setting) between the quantum value $\omega_q^\pi(G)$ and the

classical value $\omega_c^\pi(G)$. Finding non-local games with maximum $\frac{\omega_q^\pi(G)}{\omega_c^\pi(G)}$ has been an object of much research (e.g. [9,8]).

We show that the maximum advantage in the worst-case scenario is never bigger than for the best choice of a fixed probability distribution.

**Theorem 1.** *For any game $G$,*

$$\frac{\omega_q(G)}{\omega_c(G)} \leq \max_\pi \frac{\omega_q^\pi(G)}{\omega_c^\pi(G)}.$$

*Proof.* By Yao's principle [21], $\omega_c(G)$ is equal to the minimum of $\omega_c^\pi(G)$ over all probability distributions $\pi$. Let $\pi$ be the probability distribution that achieves this minimum. Then, $\omega_q^\pi(G) \geq \omega_q(G)$ (since knowing $\pi$ can only make it easier to win in a non-local game) and, hence, $\frac{\omega_q^\pi(G)}{\omega_c^\pi(G)} \geq \frac{\omega_q(G)}{\omega_c(G)}$. □

For many natural games, $\max_\pi \frac{\omega_q^\pi(G)}{\omega_c^\pi(G)}$ is achieved by $\pi = uni$ and, often, there is a straightforward symmetry argument that shows that $\omega_c^{uni} = \omega_c$ or $\omega_q^{uni} = \omega_q$. Then, the uniform distribution on inputs is equivalent to the worst case. We show two examples of that in the next two subsections.

### 3.2 CHSH Game

The CHSH game [11,12] is a canonical example of a 2-player non-local game with a quantum advantage. It is a two player XOR game with $X = Y = \{0, 1\}$, $V(a, b \mid x, y) = a \oplus b \equiv x \wedge y$, and $\pi$ the uniform distribution. It is easy to check that no deterministic strategy can win on all inputs, but the strategy $a(x) = 0$, $b(y) = 0$ wins on 3 inputs out of 4, so [12]: $\omega_c^{\text{uni}}(CHSH) = 0.75 - 0.25 = 0.5$.

Moreover, since out of the four strategies $S_1$: $a(x) = 0$, $b(y) = 0$; $S_2$: $a(x) = x$, $b(y) = 0$; $S_3$: $a(x) = 0$, $b(y) = y$; $S_4$: $a(x) = x$, $b(y) = \neg y$ each one loses on a different input, and wins on the 3 other ones, we have for any predetermined $\pi$: $\omega_c^\pi(CHSH) = 1 - 2\min_{x,y} \pi(x, y) \geq 0.5$. Indeed, one can pick the strategy losing on the input with the minimal value of $\pi$.

**Theorem 2.** $\omega_c(CHSH) = 0.5$*;* $\omega_q(CHSH) = 1/\sqrt{2}$.

*Proof.* If the players use a random variable $R$ to pick one of the strategies $S_1$, $S_2$, $S_3$, $S_4$ mentioned above with equal probability (i. e. 0.25), then for any input $x, y$ they will have a winning strategy with probability 0.75. Thus $\omega_c(CHSH) \geq 0.5$. On the other hand, $\omega_c(CHSH) \leq \omega_c^{\text{uni}}(CHSH) = 0.5$.

[12] shows that the winning probability in the quantum case is $\frac{1}{2} + \frac{1}{2\sqrt{2}}$ giving $\omega_q^{\text{uni}}(CHSH) = 1/\sqrt{2}$. Moreover, the used strategy achieves this value *on every input* $x, y$, therefore it gives also the worst-case value: $\omega_q(CHSH) = 1/\sqrt{2}$. □

### 3.3 Mermin-Ardehali Game

Mermin-Ardehali (MA) game is an $n$-player XOR game that achieves the biggest quantum advantage among XOR games with 2 questions to each player ($X_1 = \ldots = X_n = \{0,1\}$). This game corresponds to Mermin-Ardehali $n$-partite Bell inequality [15,5].

The winning condition for MA game is: $a_1 \oplus \ldots \oplus a_n = 0$ if $(x_1 + \ldots + x_n) \bmod 4 \in \{0,1\}$ and $a_1 \oplus \ldots \oplus a_n = 1$ if $(x_1 + \ldots + x_n) \bmod 4 \in \{2,3\}$.

For the uniform distribution on the inputs, we have $\omega_q^{uni}(MA) = \frac{1}{\sqrt{2}}$ and $\omega_c^{uni}(MA) = \frac{1}{2^{\lceil \frac{n}{2}-1 \rceil}}$ classically [15,5,4]. As shown by Werner and Wolf [19], no XOR game has a bigger quantum advantage.

**Theorem 3.** *[19] No $n$-party XOR game $G$ with binary inputs $x_i$ (with any input distribution $\pi$) achieves $\frac{\omega_q^\pi(G)}{\omega_c^\pi(G)} > 2^{\frac{n-1}{2}}$.*

This makes the worst-case analysis of Mermin-Ardehali game for even $n$ quite straightforward. For the quantum case, the maximal game value $\frac{1}{\sqrt{2}}$ is given by a quantum strategy which achieves the corresponding winning probability $\frac{1}{2} + \frac{1}{2\sqrt{2}}$ *on every input* [15,5,4], thus

**Theorem 4.** *For all $n$: $\omega_q(MA) = 1/\sqrt{2}$.*

For the classical case, Theorems 1 and 3 together imply $\frac{\omega_q(MA)}{\omega_c(MA)} \leq 2^{\frac{n-1}{2}}$ and $\omega_c(MA) \geq \frac{1}{2^{n/2}}$. Thus,

$$\frac{1}{2^{n/2}} \leq \omega_c(MA) \leq \omega_c^{uni}(MA) \leq \frac{1}{2^{\lceil \frac{n-1}{2} \rceil}}.$$

For even $n$, the upper and lower bounds coincide, implying

**Theorem 5.** *For even $n$: $\omega_c(MA) = 2^{-\frac{n}{2}}$.*

**Other Examples.** Other examples of well known non-local games with $\omega_c^{uni} = \omega_c$ and $\omega_q^{uni} = \omega_q$ are the Odd Cycle game of [12] and the Magic Square game of [6,12]. Again, the natural symmetries present in these games which make the worst case equivalent to the average case.

## 4 Games with Worst Case Different from Average Case

The goals of this section are:

- to present natural examples of non-local games for which the worst-case scenario is different from the average case;
- to develop methods for analyzing quantum games in the worst-case scenario (which is substantially more difficult than in the average case).

### 4.1 EQUAL-EQUAL Game

We define EQUAL-EQUAL ($EE_m$) as a two-player XOR game with $X = Y = \{1, \ldots, m\}$ and $V(a, b \mid x, y) \stackrel{def}{=} (x = y) \equiv (a = b)$.

This is a natural variation of the Odd-Cycle game of [12]. For $m = 3$, the Odd-Cycle game can be viewed as a game in which the players try to prove to the referee that they have 3 bits $a_1, a_2, a_3 \in \{0, 1\}$ which all have different values.

This can be generalized to larger $m$ in two ways. The first generalization is the Odd-Cycle game [12] in which the players attempt to prove to the referee that an $m$-cycle (for $m$ odd) is 2-colorable. The second generalization is a game in which the players attempt to prove that they have $m$ bits $a_1, \ldots, a_m \in \{0, 1\}$ which all have different values. This is our EQUAL-EQUAL game.

**Theorem 6.** *For even $m$: $\omega_c(EE_m) = \frac{m}{3m-4}$, and for odd $m$: $\omega_c(EE_m) = \frac{m+1}{3m-1}$.*

*Proof.* In the full version of paper [3]. □

**Theorem 7.** *For even $m$: $\omega_q(EE_m) = \frac{m}{3m-4}$, and for odd $m$: $\frac{m+1}{3m-1} \leq \omega_q(EE_m) \leq \frac{m^2+1}{(3m-1)(m-1)}$.*

*Proof.* The lower bounds follow from $\omega_q(EE_m) \geq \omega_c(EE_m)$. For the upper bounds, let $\pi_{\alpha,\beta}$ denote the probability distribution defined by $\pi_{\alpha,\beta}(i, i) = \alpha$ for any $i$ and $\pi_{\alpha,\beta}(i, j) = \beta$ for any distinct $i$, $j$. Then, $\omega_q(EE_m) \leq \omega_q^{\pi_{\alpha,\beta}}(EE_m)$.

For the two-player XOR games where on every input exactly one of the cases $a \oplus b = 0$ and $a \oplus b = 1$ is winning, it is useful to introduce the matrix $V$ with $V_{xy} = V(0, 0 \mid x, y)$, and to observe that $V(a, b \mid x, y) = (-1)^a(-1)^b V_{xy}$. Thus, for any distribution $\pi$

$$\omega_q^{\pi}(EE_m) = \sup_{|\psi\rangle, \mathbf{M}} \sum_{x,y,a,b} \pi(x, y) \langle\psi| M_1^{a|x} \otimes M_2^{b|y} |\psi\rangle (-1)^a (-1)^b V_{xy},$$

and by the Tsirelson's theorem [10] this game value is equal to

$$\sup_d \max_{u_i : \|u_i\|=1} \max_{v_j : \|v_j\|=1} \sum_{i=1}^{m} \sum_{j=1}^{m} \pi(i, j) V_{ij} (u_i, v_j)$$

where $u_1, \ldots, u_m, v_1, \ldots, v_m \in \mathbb{R}^d$ and $(u_i, v_j)$ is the scalar product.

The part of the sum containing $u_i$ is

$$\sum_{j=1}^{m} \pi(i, j) V_{ij} (u_i, v_j) = \left( u_i, \sum_{j=1}^{m} \pi(i, j) V_{ij} v_j \right).$$

To maximize the scalar product, $u_i$ must be the unit vector in the direction of $\sum_{j=1}^{m} \pi(i, j) V_{ij} v_j$.

For the EQUAL-EQUAL game and the distribution $\pi_{\alpha,\beta}$ we have $V_{ij} = 1$ and $\pi_{\alpha,\beta}(i,j) = \alpha$ if $i = j$, $V_{ij} = -1$ and $\pi_{\alpha,\beta}(i,j) = \beta$ if $i \neq j$. So we have to maximize the sum

$$S = \sum_{i=1}^{m} \left\| \sum_{j=1}^{m} \pi_{\alpha,\beta}(i,j) V_{ij} v_j \right\| = \sum_{i=1}^{m} \left\| \alpha v_i - \beta \sum_{j=1, j\neq i}^{m} v_j \right\|.$$

Let us denote $s = \sum_{j=1}^{m} v_j$ and apply the inequality between the arithmetic and quadratic means (and use the fact that $\|v_i\| = 1$):

$$\begin{aligned} S^2 &\leq m \sum_{i=1}^{m} \|\alpha v_i - \beta(s - v_i)\|^2 = m \sum_{i=1}^{m} \|(\alpha+\beta)v_i - \beta s\|^2 \\ &= m \left( \sum_{i=1}^{m} (\alpha+\beta)^2 \|v_i\|^2 - \sum_{i=1}^{m} 2(\alpha+\beta)\beta(v_i, s) + \sum_{i=1}^{m} \beta^2 \|s\|^2 \right) \\ &= m \left( (\alpha+\beta)^2 m - 2(\alpha+\beta)\beta \left( \sum_{i=1}^{m} v_i, s \right) + m\beta^2 \|s\|^2 \right) \\ &= m((\alpha+\beta)^2 m - 2(\alpha+\beta)\beta \|s\|^2 + m\beta^2 \|s\|^2) \\ &= m^2(\alpha+\beta)^2 + \|s\|^2 m\beta(m\beta - 2(\alpha+\beta)). \end{aligned}$$

With values of $\alpha$ and $\beta$

$$\alpha = \begin{cases} \frac{m-1}{m(3m-1)} & \text{if } m \text{ is odd,} \\ \frac{m-2}{m(3m-4)} & \text{if } m \text{ is even,} \end{cases} \quad \beta = \begin{cases} \frac{2}{(m-1)(3m-1)} & \text{if } m \text{ is odd,} \\ \frac{2}{m(3m-4)} & \text{if } m \text{ is even.} \end{cases} \tag{1}$$

one can calculate that the coefficient at $\|s\|^2$ is 0 for even $m$ and $-\frac{4}{(m-1)^2(3m-1)^2}$ (negative) for odd $m$, so dropping this summand and extracting the square root we get $S \leq m(\alpha+\beta)$. Substituting the values of $\alpha$ and $\beta$ according to equation (1) we get the desired estimations. □

Thus, for any even $m$ the quantum strategy cannot achieve any advantage over the classical strategies (and for odd $m$ there is no difference asymptotically). It was quite surprising for us. In the Appendix of full paper [3], we show a similar result for any of the symmetric distributions $\pi_{\alpha,\beta}$.

**Theorem 8.** *If $m$ is even, then $\omega_q^{\pi_{\alpha,\beta}}(EE_m) = \omega_c^{\pi_{\alpha,\beta}}(EE_m)$. If $m$ is odd, then*

$$0 \leq \omega_q^{\pi_{\alpha,\beta}}(EE_m) - \omega_c^{\pi_{\alpha,\beta}}(EE_m) \leq \frac{2}{m(3m-4)}.$$

While games with quantum advantage are common, there are only a few examples of games with no quantum advantage for an entire class of probability distributions. "Guess your neighbour's input" of [2] is one such example, with quantum strategies having no advantage for any probability distribution on the input. Our EQUAL-EQUAL game provides another natural example where quantum strategies have no advantage for a class of distributions.

Also in the Appendix of full paper [3], we show that quantum and classical values are the same for the uniform distribution.

**Corollary 1.** *For $m \geq 4$:* $\omega_q^{\text{uni}}(EE_m) = \omega_c^{\text{uni}}(EE_m) = \frac{m-2}{m}$.

### 4.2 n-Party AND Game

$n$-party AND game ($nAND$) is a symmetric XOR game with binary inputs $X_1 = \ldots = X_n = \{0,1\}$ and $V(\mathbf{a} \mid \mathbf{x}) = (\bigoplus_{i=1}^n a_i = \bigwedge_{i=1}^n x_i)$.

Although this is a natural generalization of the CHSH game (compare the winning conditions), it appears that this game has not been studied before. Possibly, this is due to the fact that in the average case the game can be won classically with a probability that is very close to 1 by a trivial strategy: all players always outputting $a_i = 0$. If this game is studied in the worst-case scenario, it becomes more interesting. The following theorem implies that $\lim_{n\to\infty} \omega_c(nAND) = 1/3$.

**Theorem 9.** $\omega_c(nAND) = 2^{n-2}/(3 \cdot 2^{n-2} - 1)$.

*Proof.* In the full version of paper [3]. □

In the quantum case, since the game is symmetric with binary inputs, we can introduce parameters $c_i$ being equal to the value of $V((0,\ldots,0) \mid \mathbf{x})$ on any input $\mathbf{x}$ containing $i$ ones and $n-i$ zeroes, and $p_i$ being equal to the probability (determined by $\pi$) of such kind of input. According to [4], for such game $G$:

$$\omega_q^{\pi}(G) = \max_{z:|z|=1} \left| \sum_{i=0}^{n} p_i c_i z^i \right|$$

where $z$ is a complex number. By Yao's principle,

$$\omega_q(G) = \min_{p_0,\ldots,p_n:\sum p_i=1} \max_{z:|z|=1} \left| \sum_{i=0}^{n} p_i c_i z^i \right|. \tag{2}$$

We have for the $nAND$ game: $c_0 = \ldots = c_{n-1} = 1$ and $c_n = -1$.

**Theorem 10.** $\lim_{n\to\infty} \omega_q(nAND) = 1/3$.

*Proof.* Since $\omega_q(nAND) \geq \omega_c(nAND) > 1/3$, it is sufficient to prove that $\omega_q(nAND) \leq 1/3 + o(1)$ by picking particular values of $p_i$ and showing that with them the limit of the expression (2) does not exceed $1/3$. Such values are: $p_n = 1/3$, $p_i = pq^{n-i}$ for $i = 0,\ldots,n-1$ where $q = e^{-\frac{1}{\sqrt{n}}}$ and $p$ is chosen so that $p\sum_{i=1}^{n} q^i = \frac{2}{3}$, i.e. $p = \frac{2}{3}\frac{1-q}{q(1-q^n)}$. The inequality to prove is

$$\lim_{n\to\infty} \max_{z:|z|=1} \left| p \sum_{i=0}^{n-1} q^{n-i} z^i - \frac{1}{3} z^n \right| \leq \frac{1}{3}.$$

Since $|z| = 1$, we can divide the expression within modulus by $z^n$ and use the substitution $w = 1/z$. We obtain

$$\lim_{n\to\infty} \max_{w:|w|=1} \left| p \sum_{i=1}^{n} (qw)^i - \frac{1}{3} \right| = \lim_{n\to\infty} \max_{w:|w|=1} \left| \frac{2}{3} \frac{1-q}{1-q^n} \frac{w(1-q^n w^n)}{1-qw} - \frac{1}{3} \right|. \tag{3}$$

From $\lim_{n\to\infty} q^n = \lim_{n\to\infty} e^{-\sqrt{n}} = 0$ we get $\lim_{n\to\infty}(1-q^n) = 1$ and, since $|w| = 1$, $\lim_{n\to\infty}(1-q^n w^n) = 1$. Thus (3) is equal to

$$\lim_{n\to\infty} \max_{w:|w|=1} \left| \frac{2}{3}\frac{(1-q)w}{1-qw} - \frac{1}{3} \right|. \tag{4}$$

**Claim 1.** *For each $\epsilon > 0$ there exists $\delta_0$ such that the inequality*

$$\left| \left| \frac{2\delta w}{1-(1-\delta)w} - 1 \right| - 1 \right| < \epsilon \tag{5}$$

*holds where $0 < \delta < \delta_0$ and $z \in C$, and $|w| = 1$.*

Now Claim 1 gives that (4) is equal to $1/3$. We used the fact that $\lim_{n\to\infty} e^{-\frac{1}{\sqrt{n}}} = 1$ and the substitution $1 - q = \delta$. □

*Proof (of Claim 1).*
The inequality (5) requires that there exists some number with absolute value 1 that is sufficiently close to $\frac{2\delta w}{1-(1-\delta)w} - 1$ or, equivalently, that there exists some number on a circle in the complex plane with its center at $1/2$ and a radius of $1/2$ that is sufficiently close to $\frac{\delta w}{1-(1-\delta)w} = \frac{1}{1+((1/w)-1)/\delta}$.

The numbers $\left\{ \frac{1}{1+((1/w)-1)/\delta} | w \in C \text{ and } |w| = 1 \right\}$ form a circle in the complex plane with its center on the real axis that has common points with the real axis at 1 and $\frac{1}{1-2/\delta} = \frac{\delta}{\delta-2}$. The latter circle is sufficiently close to the circle with its center at $1/2$ and radius of $1/2$ if we choose $\delta_0 > 0$ sufficiently small so that the value of $\frac{\delta}{\delta-2}$ is sufficiently close to 0. □

### 4.3 n-Party MAJORITY Game

By replacing the AND function with the MAJORITY function in the definition of the $n$-party AND game, we obtain the $n$-party MAJORITY game.

More formally, $n$-party MAJORITY game ($nMAJ$) is a symmetric XOR game with $X_1 = \ldots = X_n = \{0,1\}$ and $V(\mathbf{a} \mid \mathbf{x})$ demanding that $\bigoplus_{i=1}^n a_i$ is *true* if at least half of $x_i$ is *true*, and *false* otherwise. Similarly as in the previous section, we introduce parameters $c_i$ and $p_i$ and use the expression for game value given in [4]. This time $c_0 = \ldots = c_{\lceil n/2 \rceil - 1} = 1$, $c_{\lceil n/2 \rceil} = \ldots = c_n = -1$.

We have

**Theorem 11.** $\lim_{n\to\infty} \omega_c(nAND) = \lim_{n\to\infty} \omega_q(nAND) = 0$.

*Proof.* Since $0 \leq \omega_c(nMAJ) \leq \omega_q(nMAJ)$, it suffices to prove $\lim_{n\to\infty} \omega_q(nAND) = 0$. Similarly as above, we can do it by picking particular values of $p_i$ for which the limit of (2) is 0. Such values are as follows. If $n$ is even, let $n = 2k$ and $p_{2k} = 0$, otherwise let $n = 2k-1$. Let $p_i = r_i/s$ where $r_i = r_{2k-1-i}$ and $r_i = 1/(2k-1-2i)$ for $0 \leq i \leq k-1$, and $s = 2\sum_{i=1}^k 1/(2i-1)$. We have to prove that

$$\lim_{k\to\infty} \max_{z:|z|=1} \left| \sum_{i=0}^{k-1} p_i z^i - \sum_{i=k}^{2k-1} p_i z^i \right| = 0.$$

Since $|z| = 1$, we can multiply the polynomial within the modulus by $z^{1/2-k}$ and use the substitution $w = z^{-1/2}$ obtaining:

$$\max_{z:|z|=1} \left| \sum_{i=0}^{k-1} p_i z^i - \sum_{i=k}^{2k-1} p_i z^i \right| = \max_{w:|w|=1} \left| \sum_{i=0}^{k-1} p_i w^{2k-1-2i} - \sum_{i=k}^{2k-1} p_i w^{2k-1-2i} \right|$$

$$= \frac{2}{s} \max_{w:|w|=1} \left| \mathrm{Im} \left( \sum_{i=0}^{k-1} r_i w^{2k-1-2i} \right) \right| = \frac{2}{s} \max_{\theta} \left| \sum_{i=1}^{k} \frac{\sin(2i-1)\theta}{2i-1} \right|$$

where $\mathrm{Im}(z)$ is the imaginary part of $z$ and $w = e^{i\theta}$.

Since the function $\sum_{i=1}^{k} (\sin(2i-1)\theta)/(2i-1)$ is a partial sum of the Fourier series of a square wave function, we have

$$\max_{\theta} \left| \sum_{i=1}^{k} \frac{\sin(2i-1)\theta}{2i-1} \right| = O(1).$$

Also, $2/s = o(1)$ because $\lim_{k\to\infty} s = \infty$. The result follows. □

## 5 Games without Common Data

What happens if the players are not allowed to share neither common randomness nor common quantum state?

If the probability distribution on the inputs is fixed, this scenario is equivalent to two players with common randomness because common random bits can be always fixed to the value that achieves the best result for the two players. For this reason, the question above has never been studied.

In the worst-case setting, the situation changes. Players with no common randomness are no longer equivalent to players with shared randomness. For many games, not allowing shared randomness results in the players being unable to win the game with any probability $p > 1/2$.

Let $\omega_n(G)$ denote the value of a game $G$ if no shared randomness is allowed. We have

**Theorem 12.** *Suppose $G$ is a two-player XOR game (with sets of inputs $X, Y$ of arbitrary size) where on every input $(x, y)$ exactly one of the two possible values of $a \oplus b$ wins. If $\omega_n(G) > 0$ then $\omega_n(G) = 1$, i. e. then $G$ can be won deterministically.*

If we do not restrict to XOR games, it becomes possible to have a game which can be won with probability more than $\frac{1}{2}$ but not with probability 1.

**Theorem 13.** *There is a two-player game $G$ (with binary sets of inputs and outputs $X = Y = A = B = \{0, 1\}$) with $0 < \omega_n(G) = \left(\sqrt{5} - 2\right) < 1$.*

We give the proofs of both theorems in the full version of this paper [3].

## References


1. Acin, A., Brunner, N., Gisin, N., Massar, S., Pironio, S., Scarani, V.: Device-independent security of quantum cryptography against collective attacks. Physical Review Letters 98, 230501 (2007)
2. Almeida, M.L., Bancal, J.-D., Brunner, N., Acin, A., Gisin, N., Pironio, S.: Guess your neighbour's input: a multipartite non-local game with no quantum advantage. Physical Review Letters 104, 230404 (2010)
3. Ambainis, A., Backurs, A., Balodis, K., Skuskovniks, A., Smotrovs, J., Virza, M.: Worst case analysis of non-local games
4. Ambainis, A., Kravchenko, D., Nahimovs, N., Rivosh, A.: Nonlocal Quantum XOR Games for Large Number of Players. In: Kratochvíl, J., Li, A., Fiala, J., Kolman, P. (eds.) TAMC 2010. LNCS, vol. 6108, pp. 72–83. Springer, Heidelberg (2010)
5. Ardehali, M.: Bell inequalities with a magnitude of violation that grows exponentially with the number of particles. Physical Review A 46, 5375–5378 (1992)
6. Aravind, P.K.: The magic squares and Bell's theorem (2002) (manuscript)
7. Bennett, C.H., Brassard, G.: Quantum Cryptography: Public key distribution and coin tossing. In: Proceedings of the IEEE International Conference on Computers, Systems, and Signal Processing, Bangalore, p. 175 (1984)
8. Briet, J., Vidick, T.: Explicit lower and upper bounds on the entangled value of multiplayer XOR games
9. Buhrman, H., Regev, O., Scarpa, G., de Wolf, R.: Near-Optimal and Explicit Bell Inequality Violations. In: Proceedings of CCC 2011, pp. 157–166 (2011)
10. Cirelson, B. (Tsirelson): Quantum generalizations of Bell's inequality. Letters in Mathematical Physics 4, 93–100 (1980)
11. Clauser, J., Horne, M., Shimony, A., Holt, R.: Proposed experiment to test local hidden-variable theories. Physical Review Letters 23, 880 (1969)
12. Cleve, R., Høyer, P., Toner, B., Watrous, J.: Consequences and limits of nonlocal strategies. In: Proceedings of CCC 2004, pp. 236–249 (2004)
13. Gavoille, C., Kosowski, A., Markiewicz, M.: What Can Be Observed Locally? In: Keidar, I. (ed.) DISC 2009. LNCS, vol. 5805, pp. 243–257. Springer, Heidelberg (2009)
14. Kempe, J., Kobayashi, H., Matsumoto, K., Toner, B., Vidick, T.: Entangled Games are Hard to Approximate. In: Proceedings of FOCS 2008, pp. 447–456 (2008)
15. Merminm, D.: Extreme Quantum Entanglement in a Superposition of Macroscopically Distinct States. Physical Review Letters 65, 15 (1990)
16. Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: FOCS 1994, pp. 124–134 (1994)
17. Silman, J., Chailloux, A., Aharon, N., Kerenidis, I., Pironio, S., Massar, S.: Fully distrustful quantum cryptography. Physical Review Letters 106, 220501 (2011)
18. Simon, D.R.: On the power of quantum computation. In: Proceedings of FOCS 1994, pp. 116–123. IEEE (1994)
19. Werner, R.F., Wolf, M.M.: Bell inequalities and Entanglement. Quantum Information and Computation 1(3), 1–25 (2001)
20. de Wolf, R.: Quantum Communication and Complexity. Theoretical Computer Science 287(1), 337–353 (2002)
21. Yao, A.: Probabilistic computations: Toward a unified measure of complexity. In: Proceedings of the 18th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 222–227 (1977)

# Two-Dimensional Rational Automata: A Bridge Unifying One- and Two-Dimensional Language Theory*

Marcella Anselmo[1], Dora Giammarresi[2], and and Maria Madonia[3]

[1] Dipartimento di Informatica, Università di Salerno I-84084 Fisciano (SA) Italy
anselmo@dia.unisa.it
[2] Dipartimento di Matematica. Università di Roma "Tor Vergata",
via della Ricerca Scientifica, 00133 Roma, Italy
giammarr@mat.uniroma2.it
[3] Dip. Matematica e Informatica, Università di Catania,
Viale Andrea Doria 6/a, 95125 Catania, Italy
madonia@dmi.unict.it

**Abstract.** We define *two-dimensional rational automata* for pictures as an extension of classical *finite automata* for strings. They are obtained replacing the finite relation computed by the transition function with a rational relation computed by a transducer. The model provides a uniform setting for the most important notions, techniques and results presented in the last decades for recognizable two-dimensional languages, and establishes new connections between one- and two- dimensional language theory.

## 1 Introduction

*Two-dimensional languages* are the extension of string languages to two dimensions: a two-dimensional string, called also *picture*, is a rectangular array of symbols taken from a finite alphabet. The first automaton model for two-dimensional languages was introduced in 1967 by M. Blum and C. Hewitt [5]: it was the *four-way automaton* and generalized the two-way automaton for strings. It was an unsatisfactory model because recognized languages lack of basic closure properties. Since then, different types of models have been introduced to generate or recognize two-dimensional languages with the intent of developing a formal theory of two-dimensional recognizability.

In the past 20 years the class that got more consensus as a valid counterpart of regular string languages is the family *REC* of *tiling recognizable picture languages* introduced in [13] (see also [14]). A two-dimensional language is *tiling recognizable* if it can be obtained as a projection of a local picture language (given by a finite set of allowed sub-pictures called *tiles*). The local picture language together with the alphabetic projection is called *tiling system*. REC family

* Partially supported by 60% Projects of University of Catania, Roma "Tor Vergata", Salerno (2011).

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 133–145, 2013.

inherits several properties from the class of regular string languages and it has characterizations also in terms of domino tiles, Wang systems, logic formulas and of some kind of regular expressions ([9,14,15,18]). Moreover REC is equivalent to the class of languages recognized by on-line tesselation automata (2OTA) [17].

It is worth remarking that tiling systems are intrinsically non-deterministic and that, unlike the one-dimensional case, REC is not closed under complementation. The definition of unambiguous recognizable languages (UREC class) is quite natural. The definition of a first deterministic subclass, referred to as DREC, was inspired to the above mentioned on-line tesselation acceptors. Moreover other REC subclasses (called Col-UREC and Row-UREC) were introduced and studied in [2] as intermediate classes between UREC and DREC. Later it was proved that Col- and Row-UREC correspond to a deterministic class called Snake-DREC based on a boustrophedonic scanning strategy for the input ([20]). Recently, two different versions of automata, *tiling automata* and *Wang automata*, [1,19,20] were proposed as a "machine version" of tiling systems, with the main intent of understanding the notion of determinism in tiling systems.

The initial motivation of this paper is the observation that all the theory of tiling recognizable picture languages collect several results and characterizations always obtained with *ad hoc* techniques that come from different approaches and points of view for class REC. Furthermore all approaches started from one-dimensional language theory, but then they had no more relation with it. Especially the definitions of deterministic subclasses seem to be difficult to be accepted as "natural" generalizations of determinism for strings.

Then, we restart from the string language theory and in particular from the definition of finite automaton. To go from one-dimensional to two-dimensional world, we take inspiration from geometry: the roles of points and lines in one dimension are played, in two dimensions, by lines and planes respectively. By analogy, we assume that in formal language theory, symbols are "points", strings are "segments", or portion of a "line", and pictures are "rectangles", or portions of a "plane". In one dimension, *finite* sets of symbols are used to define *finite* automata that accept *rational* sets of strings. Here we use *rational* sets of strings to introduce *rational* automata that define *recognizable* sets of pictures. We then replace all words "symbol" with "string" and "finite" with "rational". We call the model, a *(two-dimensional) rational automaton*, *RA* for short, to emphasize the replacement of the finiteness hypothesis with the rationalness hypothesis.

Let $\Sigma$ be a finite alphabet we use in one dimension, then in two dimensions we take as alphabet the rational set of strings $\Sigma^+$ to be considered as vertical strings, we simply call *columns*. In this setting, a picture $p$ over $\Sigma$ can be viewed as a string (read from left-to right) over the alphabet $\Sigma^+$ of columns. The set of states will be a rational set of strings (again columns) over a finite set $Q$ of symbols and the transition function is such that its graph is a *rational* relation over $(Q \times \Sigma)$ and $Q$. This implies that the transition function corresponds to a rational transduction that hence can be computed by a transducer. We point out that other authors used transducers or rational graphs when dealing with two-dimensional languages, although the context was different (see [18,10,8,6]).

As a two dimensional version of a fundamental theorem for strings (see [11]), we prove that: a picture language is recognized by a two-dimensional rational automaton if and only if it is the projection of a local picture language. Namely, picture languages accepted by RA correspond to family REC.

From FA we also import definitions of unambiguity and determinism. A *two-dimensional unambiguous RA* has the property that each input picture has at most one accepting computation. A RA is *deterministic* if the transition function corresponds to a rational function. We observe that we can also define determinism as a stronger property of the transducer that computes the transition function, by requiring that it is left-sequential (we call it *strong determinism*). Then we show that the classes UREC, Col-UREC and DREC can be characterized as families of languages recognized by unambiguous, deterministic and strongly deterministic, respectively, rational automata.

As interesting consequence we show that this new characterization of REC via rational automata allows us to prove all known results uniformly inside this new setting, without *ad hoc* special techniques, by just using the theory of finite automata for strings or by applying old results on transducers.

## 2 Preliminaries

We introduce some definitions about two-dimensional languages ([14,7]).

A *picture* over a finite alphabet $\Sigma$ is a two-dimensional rectangular array of elements of $\Sigma$. Given a picture $p$ with $m$ rows and $n$ columns, the pair $(m, n)$ is the *size* of $p$. The set of all pictures over $\Sigma$ is denoted by $\Sigma^{**}$. A *two-dimensional language*, or *picture language*, over $\Sigma$ is a subset of $\Sigma^{**}$. Let $p, q \in \Sigma^{**}$ be pictures of size $(m, n)$ and $(m', n')$, respectively, the *column concatenation* of $p$ and $q$ (denoted by $p \oplus q$) and the *row concatenation* of $p$ and $q$ (denoted by $p \ominus q$) are partial operations, defined only if $m = m'$ and if $n = n'$, respectively, as:

$$p \oplus q = \begin{array}{|c|c|}\hline p & q \\ \hline \end{array} \qquad p \ominus q = \begin{array}{|c|}\hline p \\ \hline q \\ \hline \end{array}.$$

The definitions can be extended to define languages concatenations. By iterating the row (column, resp.) concatenation for language $L$, we obtain the row (column, resp.) *star* of $L$, denoted by $L^{*\ominus}$ ($L^{*\oplus}$, resp.).

To give definitions of recognizability, we consider for a picture $p$, the *bordered picture* $\widehat{p}$ obtained by surrounding $p$ with a special *boundary symbol* $\# \notin \Sigma$. A *tile* is a picture of size $(2, 2)$ and $[[p]]$ is the set of all sub-blocks of size $(2, 2)$ of picture $\widehat{p}$. Given a finite alphabet $\Gamma$, a two-dimensional language $L \subseteq \Gamma^{**}$ is *local* if there exists a set $\Theta$ of tiles over $\Gamma \cup \{\#\}$ (the set of *allowed blocks*) such that $L = \{p \in \Gamma^{**} |\ [[p]] \subseteq \Theta\}$ and we will write $L = L(\Theta)$.

A *tiling system* is a quadruple $(\Sigma, \Gamma, \Theta, \pi)$ where $\Sigma$ and $\Gamma$ are finite alphabets, $\Theta$ is a set of tiles over $\Gamma \cup \{\#\}$ and $\pi : \Gamma \to \Sigma$ is a projection. A picture $p' \in \Gamma^{**}$ is a *pre-image* of $p \in \Sigma^{**}$ if $\pi(p') = p$. A two-dimensional language $L \subseteq \Sigma^{**}$ is recognized by a *tiling system* $(\Sigma, \Gamma, \Theta, \pi)$ if $L = \pi(L(\Theta))$. The family of all *tiling recognizable* picture languages over the alphabet $\Sigma$, is denoted by *REC*.

*Example 1.* Consider the language $L$ of square pictures over $\Sigma = \{a\}$, that is pictures with the same number of rows and columns. $L$ is not a local language, but it belongs to REC. Indeed it can be obtained as the projection of the local language of squares over the alphabet $\Gamma = \{0, 1, 2\}$ in which all the symbols in the main diagonal are 1, positions below it carry symbol 2, whereas the remaining positions carry symbol 0. An example of a picture $p \in L$ together with its pre-image $p'$ is given below. The reader can infer the set $\Theta$ of tiles by taking $[[p']]$. The projection $\pi$ maps any symbol in $\Gamma$ to $a$.

$$p = \begin{array}{|c|c|c|c|}\hline a & a & a & a \\ \hline a & a & a & a \\ \hline a & a & a & a \\ \hline a & a & a & a \\ \hline \end{array} \qquad p' = \begin{array}{|c|c|c|c|}\hline 1 & 0 & 0 & 0 \\ \hline 2 & 1 & 0 & 0 \\ \hline 2 & 2 & 1 & 0 \\ \hline 2 & 2 & 2 & 1 \\ \hline \end{array}.$$

*Wang systems* are a model, that is equivalent to tiling systems, introduced in [9] and based on labeled Wang tiles. A *labeled Wang tile* is a 5-tuple consisting of four colors $w, e, n, s$, chosen from a finite set, placed at the borders of a picture with a label $x$ taken in a finite alphabet, usually represented as follows: $\begin{array}{|c|}\hline n \\ w\ x\ e \\ s \\ \hline \end{array}$.

A tiling system is *unambiguous* if any recognized picture has a unique pre-image. *UREC* is the class of languages recognized by an unambiguous tiling system [13]. The notion of determinism is instead related to some "direction" of computation. *DREC* is the class of all languages recognized by a deterministic tiling system along one of the four corner-to-corner directions. Note that, the determinism referred to different directions, yields different subfamilies of REC. In this paper we consider family $DREC_{tl}$, corresponding to the direction from the top-left corner to the bottom-right corner. A language in $DREC_{tl}$ is recognized by a tiling system $(\Sigma, \Gamma, \Theta, \pi)$ with the property that, for any three symbols $\gamma_1, \gamma_2, \gamma_3 \in \Gamma$ and $a \in \Sigma$ there is at most one tile $\begin{array}{|c|c|}\hline \gamma_1 & \gamma_2 \\ \hline \gamma_3 & \gamma_4 \\ \hline \end{array} \in \Theta$ with $\pi(\gamma_4) = a$.

In [2], the *column-unambiguity* was introduced as an intermediate notion between determinism and unambiguity. It refers to a scanning strategy along the left-to-right (or right-to-left) direction and then reading pictures column after column. Roughly speaking, a tiling system is column-unambiguous if any column of a recognized picture has a unique pre-image: it is actually a sort of unambiguity defined on each single column. *Col-UREC* denotes the class of languages recognized by a column-unambiguous tiling system, along some side-to-side direction. In this paper we will be interested in *Col-UREC*$_l$, the class of languages recognized by l2r-unambiguous tiling systems. In [20] the class Col-UREC$_l$ is characterized in terms of *snake-determinism*. In [2,3] it is proved that DREC $\subsetneq$ Col-UREC $\subsetneq$ UREC $\subsetneq$ REC.

We conclude this section by recalling some definitions about rational transductions and rational functions following [4] (other textbooks could use different terminology, e.g. [16,23]). Let $X$ and $Y$ be two alphabets. A *transduction* $\tau$ from $X^*$ into $Y^*$ is a function from $X^*$ to the set $2^{Y^*}$ of subsets of $Y^*$. The

*graph of* $\tau$ is the relation $R = \{(u, v) \in X^* \times Y^* \mid v \in \tau(u)\}$. A transduction $\tau : X^* \longrightarrow Y^*$ is *rational* if and only if its graph is a rational relation over $X$ and $Y$. The machines realizing rational transductions are called transducers. A *transducer* $T = (X, Y, R, r_0, R_F, E)$ is composed of an input alphabet $X$, an output alphabet $Y$, a finite set of states $R$, an initial state $r_0$, a set of final states $R_F$, and a finite set of transitions or edges $E \subseteq R \times X^* \times Y^* \times R$. If $E \subseteq R \times X \times Y \times R$, then T is said *letter-to-letter*. It is well known that a transduction $\tau$ is rational if and only if $\tau$ is realized by a transducer. A *rational function* is a rational transduction which is a partial function.

A "deterministic" model of transducers is given by left sequential transducers. The transitions in a *left sequential transducer* $L$ are usually given by two partial functions $\mu : R \times X \longrightarrow R$ and $\nu : R \times X \longrightarrow Y^*$. The function realized by $L$ is the output of $L$ when it starts in the initial state and reads $u$ from left to right. A partial function is a *left sequential transduction* if it can be realized by some left sequential transducer $L$. Similar definitions are given for *right sequential transducer/transduction* that read the input from right to left. Not all partial functions can be realized by a left sequential transducer. Nevertheless, the following "decomposition" theorem was proved ([12,4]).

**Theorem 1 (Elgot, Mezei, 1965).** *Let $\alpha : X^* \longrightarrow Y^*$ be a partial function. Then $\alpha$ is rational if and only if there are a left sequential transduction $\lambda : X^* \longrightarrow Z^*$ and a right sequential transduction $\rho : Z^* \longrightarrow Y^*$ such that $\alpha = \lambda \circ \rho$.*

## 3 Two-Dimensional Rational Automata

We now extend the classical model of *finite automaton (FA)* for string languages, to recognize sets of pictures. To go from one to two dimensions, we take inspiration from geometry where objects defined in $d$ dimensions are used to build and define objects in $d + 1$ dimensions. While in one dimension we use *finite* sets of *symbols* to introduce *finite* automata that define *rational* sets of strings, in this paper we use *rational* set of *strings* to introduce *rational* automata that define *recognizable* set of pictures.

Let $\Sigma$ be a finite set of symbols we use as alphabet in one dimension, then in two dimensions we take as alphabet the rational set of strings $A_\Sigma = \Sigma^+$. Without loss of generality, think of strings in $A_\Sigma$ as vertical strings to be read from top to bottom, to which we refer as *columns*. A column in $A_\Sigma$ will be indicated as $\underline{a}$. Moreover, as in one dimension we identify symbols with strings of length 1, here we consider a vertical string of length $m$ as a picture of size $(m, 1)$ and, with a little abuse of notation, we use the column concatenation operator $\oplus$ to concatenate strings in $A_\Sigma$. In this setting, a picture $p$ over $\Sigma$ can be viewed as a string over the alphabet of columns $A_\Sigma$ and we write $p = \underline{a}_1 \oplus \underline{a}_2 \oplus \ldots \oplus \underline{a}_m$. This is coherent with our aim: by definition, the use of $\oplus$-concatenation produces only rectangular pictures and we can write ${A_\Sigma}^{+\oplus} = \Sigma^{++}$.

To extend the definition of automaton to two dimensions, we then substitute all words "symbol" with "string" and all words "finite" with "rational" and define a *two-dimensional rational automaton* whose alphabet is $A_\Sigma$. The set of

states will be a rational set of strings over a finite set $Q$ of symbols we denote by $S_Q$. Again strings in $S_Q$ will be considered as vertical strings, while $\underline{s}$ will be a generic column state in $S_Q$. We consider a distinguished *initial* symbol $q_0 \in Q$ from which we define the set $S_0 = q_0^+$ of initial states, while a rational subset $F_Q \subseteq S_Q$ will be taken as set of accepting states. The transition function will be defined as usual, i.e. it takes a state and a symbol and gives a certain number of possible new states. Since in FA the transition function is such that its graph is a *finite* relation over $(Q \times \Sigma)$ and $Q$, here, consistently, we impose the constrain that the graph of the transition function is a *rational* relation over $(Q \times \Sigma)$ and $Q$. This implies that the transition function corresponds to a rational transduction $\tau : (Q \times \Sigma)^+ \to 2^{Q^+}$ computed by a transducer $T$. Remark that the transducer $T$ needs to be letter-to-letter to ensure that the transition function produces a column state "compatible" with the next column of the picture.

**Definition 1.** *A* (two-dimensional) rational automaton*, (RA for short), over the alphabet $\Sigma$ is a quintuple $\mathcal{H} = (A_\Sigma, S_Q, S_0, \delta_T, F_Q)$ where:*
- $A_\Sigma = \Sigma^+$ *is the* alphabet
- $S_Q \subseteq Q^+$*, for some finite set $Q$, is a rational language called the set of* states
- $S_0 = \{q_0\}^+$*, for some $q_0 \in Q$, is the set of* initial states
- $\delta_T : S_Q \times A_\Sigma \longrightarrow 2^{S_Q}$ *is the rational relation computed by a letter-to-letter transducer $T = (Q \times \Sigma, Q, R, r_0, R_F, E)$, and is called the* transition function
- $F_Q \subseteq S_Q$ *is a rational language called the set of* final states.

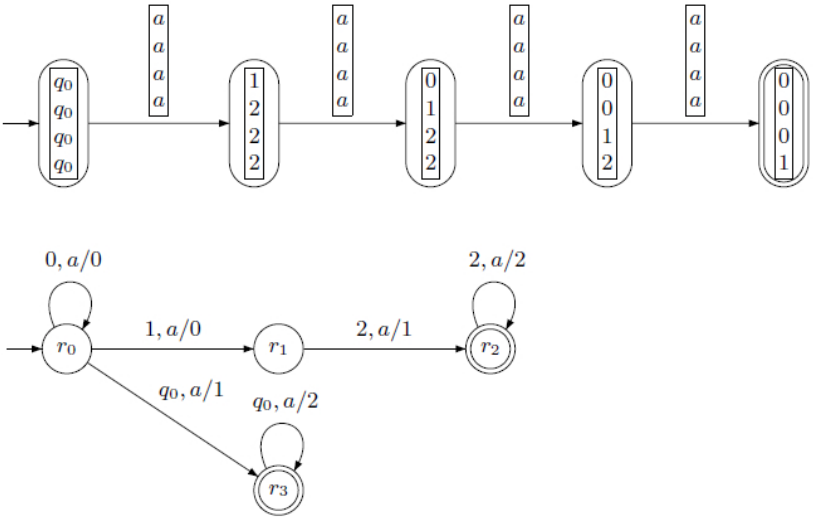
More precisely: let $\underline{s} = s_1 s_2 \dots s_m$ be a state in $S_Q$ and let $\underline{a} = a_1 a_2 \dots a_m$ be a column in $A_\Sigma$. Then $\delta_T(\underline{s}, \underline{a}) = \underline{q}$ where $\underline{q} = q_1 q_2 \dots q_m$ is the output of the transducer $T$ on the string $(s_1, a_1)(s_2, a_2) \dots (s_m, a_m)$ over the alphabet $Q \times \Sigma$.

A computation of a RA $\mathcal{H}$ on a picture $p \in \Sigma^{++}$ of size $(m, n)$ is given exactly as in FA, just considering the input picture as a string of columns $p = \underline{p}_1 \circledcirc \underline{p}_2 \circledcirc \dots \circledcirc \underline{p}_n$. The computation starts from the initial state of length $m$, i.e. from $q_0^m$, reading the columns from left to right. Then $p$ is *recognized* by $\mathcal{H}$ if at the end of the computation, we end up in a state $\underline{s}_f \in F_Q$ (the condition can be checked by a FA recognizing $F_Q$). The *language recognized by a two-dimensional rational automaton* $\mathcal{H}$, denoted $\mathcal{L}(\mathcal{H})$, is the language of all pictures recognized by $\mathcal{H}$; whereas $\mathcal{L}(RA)$ denotes the class of all languages recognized by a RA.

The state graph of a RA can be defined as in the case of FA: remark that in this case the graph is infinite, and that all vertices labeled $q_0^i$, for some $i \geq 1$, are initial states. The definition of picture recognized by a RA can be given as well by using the state graph, and the state graph can be viewed as the disjoint union of the graphs used for the recognition of pictures of fixed height $m$, for any $m \geq 1$. Such graphs correspond indeed to the automata introduced by O. Matz in [21] for proving non-recognizability conditions of picture languages. Notice that rational automata manage the Matz's automata all together by the transducer of the transition function. It is also interesting to notice that the rational automata resemble the rational graphs described in [6] to recognize string languages. It can be seen that rational automata includes as particular case such rational graphs. Two-dimensional tilings have been sometimes put in relation with iteration of rational transducers although in a different context

[8,10,18]. The model of rational automata we introduce does not correspond to a mere iteration of a transducer since, at each step of the computation, it combines a column of states with a new column of input symbols. This comes from the fact that two-dimensional languages are recognized by *labeled* Wang tiles.

*Example 2.* Let $L$ be the set of square pictures over $\Sigma = \{a\}$, as in Example 1. To obtain a rational automaton $\mathcal{H} = (A_\Sigma, S_Q, S_0, \delta_T, F_Q)$ recognizing $L$ let us set $Q = \{q_0, 0, 1, 2\}$, $A_\Sigma = a^+$, $S_Q = q_0^+ + 0^*12^*$, $S_0 = q_0^+$ and $F_Q = 0^*1$. The transition function $\delta_T$ maps the pair $(q_0^m, a^m) \in S_Q \times A_\Sigma$ to state $12^{m-1} \in S_Q$, and the pair $(0^h12^k, a^{h+k+1}) \in S_Q \times A_\Sigma$ to state $0^{h+1}12^{k-1} \in S_Q$, for any $m, h \geq 0$ and $k \geq 1$. The state graph of $\mathcal{H}$ is the disjoint union of the state graphs recognizing pictures with fixed height $m$, for any $m \geq 1$.




It is immediate to verify that there is a one-to-one correspondence between the above successful computation of $\mathcal{H}$ on a picture $p$ of size $(4, 4)$ throughout states $\underline{s}_{04}, \underline{s}_{14}, \underline{s}_{24}, \underline{s}_{34}, \underline{s}_{44}$ and the local picture $p'$ given in Example 1. Actually it is exactly $p' = \underline{s}_{14} \circledcirc \underline{s}_{24} \circledcirc \underline{s}_{34} \circledcirc \underline{s}_{44}$. In general given a tiling system for a language $L$, we can define a RA $\mathcal{H}$ that accepts $L$, whose transitions are defined in a way that the sequence of states of an accepting computation for a picture $p \in L$ corresponds to its pre-image $p'$ in the tiling system.

On the converse, given a a rational automaton $\mathcal{H}$ with transition function $\delta_T$ computed by transducer $T$, we can always define a tiling system in a way that a successful computation of $\mathcal{H}$ on a picture $p$ is represented as its pre-image $p'$ in the tiling system. The key idea is to take the transitions of the transducer $T$ as symbols of the local alphabet in the tiling system and write them in

a "spatial shape". More precisely, a transition $(r_h, (q_i, a), q_j, r_k)$ will correspond to the local symbol $\begin{array}{|ccc|}\hline & r_h & \\ q_i & a & q_j \\ & r_k & \\ \hline\end{array}$ that already "resemble" a labeled Wang tile.

All these reasonings give the evidence of the fact that picture languages accepted by two-dimensional rational automata coincide with all tiling recognizable languages (i.e. projection of local picture languages). We state this as the following theorem. We omit the proof for lack of space.

**Theorem 2.** *A picture language is recognized by a two-dimensional rational automaton if and only if it is tiling recognizable.*

It is interesting to remark that Theorem 2 is the two-dimensional version of a classical theorem (credited to Y. Medvedev [22], see [11]) stating that a string language is recognized by a FA iff it is the projection of a local string language. Then rational automata recognize REC family confirming the robustness of the definition of this class that can be characterized in several contexts and settings.

## 4 Studying REC via Rational Automata

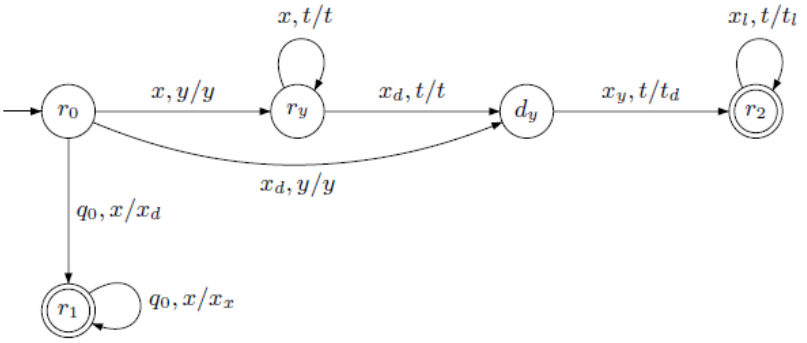
In this section we show how to use the model of rational automaton to obtain results on (recognizable) two-dimensional languages. As a main outcome, we obtain that most of the results on REC proved in the past 20 years using *ad hoc* techniques, exploiting the different characterizations for REC, obtain now a more natural and uniform placement in the theory of rational automata. More precisely, we show how several properties (closure properties, non-inclusion results and decidability issues) on REC can be proved by using known results on transducers (to be applied to the rational automaton transition function) or by "translating" proof techniques from finite automata to rational automata.

First of all, let us point out some peculiar features of the model of RA, with respect to other representations of two-dimensional languages. Example 2 presents a RA that mimics a tiling system to recognize the language of unary squares. Remark that, while being equivalent to a tiling system, RA besides their states can exploit the "extra memory" of the states of the transducer, as in the following example. This could simplify the recognition algorithm of a language.

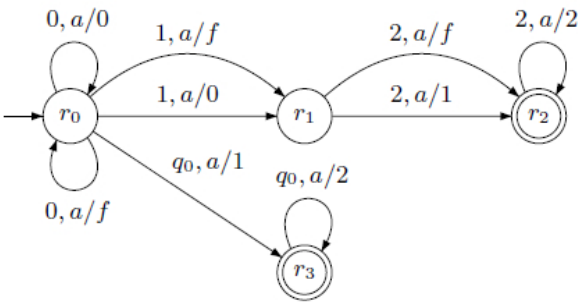
*Example 3.* Let $L_{fr=fc}$ be the language of squares over a two letters alphabet $\Sigma = \{a, b\}$ with the first row equal to the first column. A rational automaton $\mathcal{H} = (A_\Sigma, S_Q, S_0, \delta_T, F_Q)$ recognizing $L_{fr=fc}$ is the following: $A_\Sigma = \{a, b\}^+$; $Q = \{q_0, x_y, x_d \mid x, y \in \Sigma, d \notin \Sigma\}$; a state in $S_Q$ is either a string in $q_0^+$ or any column $x(1)x(2)\cdots x(k-1)x(k)_d x(k+1)_{l_{k+1}} \cdots x(n)_{l_n}$, where $1 \leq k \leq n$, and $x(i), l_j \in \Sigma$, for $1 \leq i < k$, $k < j \leq n$; $S_0 = q_0^+$ and $F_Q$ is the set of states $x(1)x(2)\cdots x(n-1)x(n)_d$. The idea is that in each column state the subscript $d$ denotes the position on the diagonal, while subscripts $l_j$ propagate to the right the corresponding symbol of the first column.

The transducer $T$ that realizes $\delta_T$ is schematized here below. Its states are: $r_0$, $r_1$, $r_2$, and $r_y$ and $d_y$, for any $y \in \Sigma$. In fact the generic path from $r_0$ to $r_2$

(or $r_1$) here depicted, represents several paths. The state $r_y$ keeps track of the first symbol $y$ read in the column; the information is passed to next state $d_y$, whenever the diagonal is reached, and then matched with the symbol propagated from the first column by the transition from $d_y$ to $r_2$ on input $x_y$.




*Remark 1.* The nondeterminism of the RA model allows to recognize all languages in REC, even when restricting the model to those RA whose set of final states is $F_Q = f^+$, for some $f \in Q$. As an example, consider the language $L$ of Example 2. We can define a RA for $L$ by modifying the one in Example 2 by adding the new element $f$ to the set $Q$ and by using the following transducer $T'$ to compute the transition function $\delta_{T'}$. The idea is, to insert the state $f^m$ in $\delta_{T'}(\underline{s}, \underline{a})$, for any $m$ and pair $(\underline{s}, \underline{a}) \in S_Q \times A_\Sigma$ such that $\delta_T(\underline{s}, \underline{a}) \cap F_Q \neq \emptyset$.




We now show how some closure properties of REC can be proved with RA in a way that naturally extends techniques from finite automata. Consider for example the column concatenation of two recognizable languages. In one dimension the concatenation of two FA can be simply achieved using as first automaton a FA with a single final state with no outgoing transitions, and identifying this final state with the initial state of the second automaton. In the RA framework we can naturally extends this proof. On the other hand, the original proof by tiling systems, needs a special manipulation of the tiles to glue the columns, while OTA or logic formulas, seem to be not at all adequate to prove such results.

**Proposition 1.** *The class of languages recognized by rational automata is closed under union, intersection, column and row concatenations and stars.*

*Proof. (Sketch)* The proof of the closure under row concatenation starts remarking that, if $R_1$ and $R_2$ are graphs of two transductions then the transduction with graph $R_1 R_2$ is still rational [4], and working on details.

To prove the closure under column concatenation, notice that one can suppose that the first RA has final states of the form $f^+$ (for some fresh symbol $f$) (see Remark 1). Then it is sufficient to combine states and transitions of the two RA, replacing any occurrence of $f$ with the initial state of the second RA.

In a similar way, the closure under union, intersection, column and row closure can be proved generalizing analogous proofs for FA. □

The notions of unambiguity and determinism are fundamental for formal languages and their effectiveness. While the definition of unambiguity is well accepted, the definition of determinism for 2d languages is still controversial: it needs the choice of some "scanning strategy" and, in a "non-oriented" model as tiling systems are, it is not clear which strategy is the "right" one. Let us introduce these definitions in the RA framework. An *unambiguous two-dimensional rational automaton* (URA, for short) is a RA with the property to admit at most one accepting computation on each input picture. Regarding determinism, two different definitions can be naturally given, depending whether the transition function corresponds to a rational function or the transducer is left-sequential.

**Definition 2.** *A two-dimensional rational automaton $\mathcal{H} = (A_\Sigma, S_Q, S_0, \delta_T, F_Q)$ is a* deterministic rational automaton *(DRA) if $\delta_T : S_Q \times A_\Sigma \to S_Q$; $\mathcal{H}$ is a* strongly deterministic rational automaton *(SDRA) if $T$ is left-sequential.*

As usual $\mathcal{L}(URA)$, $\mathcal{L}(DRA)$, $\mathcal{L}(SDRA)$ will denote the corresponding classes of languages. As an example, the RA given in Example 2 is strongly deterministic, while the RA depicted in Remark 1 for the same language is neither strongly deterministic nor deterministic.

Theorem 2 states a correspondence between the classes of languages recognized by RA and REC. By working on the details of the constructions, it can be seen that such correspondence is inherited by restricted version of rational automata. We emphasize the relevance of Theorem 3: it somehow warrants (putting them in a common setting with non-determinism and unambiguity) the definitions of determinism and column unambiguity, that were sometimes regarded as arbitrary.

**Theorem 3.** *Let $L$ be a picture language.*
*$L \in \mathcal{L}(URA)$ if and only if $L \in UREC$.*
*$L \in \mathcal{L}(DRA)$ if and only if $L \in Col\text{-}UREC_l$.*
*$L \in \mathcal{L}(SDRA)$ if and only if $L \in DREC_{tl}$.*

In [2] it is proved that all the inclusions $\mathrm{DREC}_{tl} \subsetneq \mathrm{Col\text{-}UREC}_l \subsetneq \mathrm{UREC} \subsetneq \mathrm{REC}$ are strict. It is interesting to remark that the original proofs can be directly used and they are even "more natural" in the setting of rational automata, since they are based on Matz's automata ([21]). We then state the following corollary.

**Corollary 1.** $\mathcal{L}(SDRA) \subsetneq \mathcal{L}(DRA) \subsetneq \mathcal{L}(URA) \subsetneq \mathcal{L}(RA)$.

Let us now consider decidability issues. Using known decidability results on transducers, we directly obtain decidability results for rational automata (that in turns imply decidability results for tiling systems). Among all, we state the following result whose proof follows from the fact that is decidable whether the transduction realized by a transducer is a function and whether it is a left sequential transduction [4].

**Proposition 2.** *It is decidable whether a rational automaton is deterministic (strongly deterministic, resp.).*

As a last exciting example of application of the rational automaton model, we consider the result by Elgot and Metzei [12] (see Theorem 1). If we apply this theorem to the transducer of a deterministic rational automaton we obtain that its transition function corresponds to a composition of two sequential transducers; any input column is processed twice: the first transducer reads it from top to bottom and then the second reads the output column from bottom to top.

**Theorem 4.** *A picture language is recognized by a deterministic rational automaton if and only if it is recognizable by a rational automaton where the transition function is computed by a pair of letter-to-letter transducers, a left-sequential one and a right-sequential one, used alternatingly on the columns.*

*Proof.* Let $L$ be recognized by a DRA $\mathcal{H} = (A_\Sigma, S_Q, S_0, \delta_T, F_Q)$. Since $\mathcal{H}$ is deterministic, $\delta_T$ is computed by a a transduction $\tau$ that is a rational partial function. From Elgot and Mezei's Theorem, we can both write $\tau = \rho \circ \lambda$ and $\tau = \lambda' \circ \rho'$, where $\lambda$ and $\lambda'$ are left sequential transductions and $\rho$ and $\rho'$ are right sequential transductions. The idea is now to use $\lambda$ and $\lambda'$ to construct a new left sequential transduction, say $\overline{\lambda}$, and to use $\rho$ and $\rho'$ to construct a new right sequential transduction, say $\overline{\rho}$, that together will play the same role as $\tau$ in computing $\delta_T$. More exactly, suppose $\lambda : (Q \times \Sigma)^+ \to Z^+$, $\rho : Z^+ \to Q^+$, $\rho' : (Q \times \Sigma)^+ \to Z'^+$ and $\lambda' : Z'^+ \to Q^+$. Define $\overline{\rho} : (Z \times \Sigma)^+ \to Z'^+$ and $\overline{\lambda} : (Z' \times \Sigma)^+ \to Z^+$ such that for $(z, \sigma) \in (Z \times \Sigma)^+$, $\overline{\rho}(z, \sigma) = \rho'(\rho(z), \sigma)$ and for $(z', \sigma) \in (Z' \times \Sigma)^+$, $\overline{\lambda}(z, \sigma) = \lambda(\lambda'(z'), \sigma)$. Consider now the left-sequential transducer $T_{\overline{\lambda}}$ realizing $\overline{\lambda}$ and the right-sequential transducer $T_{\overline{\rho}}$ realizing $\overline{\rho}$. Observe that, since $\tau$ is realized by a letter-to-letter transducer, then also $\overline{\lambda}$ and $\overline{\rho}$ can be realized by letter-to-letter transducers. Hence $L$ is recognized by a RA obtained from $\mathcal{H}$ replacing $Q$ with $Z \cup Z'$ and where the new transition function can be computed by the pair $(T_{\overline{\lambda}}, T_{\overline{\rho}})$ used alternatingly on the columns.

The converse follows directly by the definitions. □

Remark that Theorems 3 and 4 together show that Col-UREC$_l$ corresponds to a deterministic model of tiling system called Snake-DREC. This result can be found in [20] where it is proved with *ad hoc* techniques.

All the results in this section show that, despite a rational automaton is in principle more complicated than a tiling system, it has some major advantages.

It unifies some concepts that seemed to come from different motivations (e.g. notions of determinism) and allows to use results of the string language theory. Further a very interesting step will be to look for other results on transducers and finite automata to prove new properties of REC.

## References


1. Anselmo, M., Giammarresi, D., Madonia, M.: A computational model for recognizable two-dimensional languages. Theoret. Comput. Sci. 410(37), 3520–3529 (2009)
2. Anselmo, M., Giammarresi, D., Madonia, M.: Deterministic and unambiguous families within recognizable two-dimensional languages. Fund. Inform. 98(2-3), 143–166 (2010)
3. Anselmo, M., Madonia, M.: Deterministic and unambiguous two-dimensional languages over one-letter alphabet. Theor. Comput. Sci. 410(16), 1477–1485 (2009)
4. Berstel, J.: Transductions and Context-Free Languages. Teubner Studienbücher (1979)
5. Blum, M., Hewitt, C.: Automata on a 2-dimensional tape. In: FOCS, pp. 155–160. IEEE (1967)
6. Carayol, A., Meyer, A.: Context-sensitive languages, rational graphs and determinism. Logical Methods in Computer Science 2(2), 1–24 (2006)
7. Crespi Reghizzi, S., Giammarresi, D., Lonati, V.: Two dimensional models. In: Pin, J.E. (ed.) Automata from Mathematics to Application. Eur. Math. Soc. (to appear)
8. Culik II, K., Kari, J.: An Aperiodic Set of Wang Cubes. In: Puech, C., Reischuk, R. (eds.) STACS 1996. LNCS, vol. 1046, pp. 137–146. Springer, Heidelberg (1996)
9. De Prophetis, L., Varricchio, S.: Recognizability of rectangular pictures by Wang systems. Journal of Automata, Languages and Combinatorics 2(4), 269–288 (1997)
10. Dolzhenko, E., Jonoska, N.: On Complexity of Two Dimensional Languages Generated by Transducers. In: Ibarra, O.H., Ravikumar, B. (eds.) CIAA 2008. LNCS, vol. 5148, pp. 181–190. Springer, Heidelberg (2008)
11. Eilenberg, S.: Automata, Languages and Machines, vol. A. Academic Press, N.Y. (1974)
12. Elgot, C.C., Metzei, J.E.: On relations defined by generalized finite automata. IBM J. Res. Develop. 9, 47–68 (1965)
13. Giammarresi, D., Restivo, A.: Recognizable picture languages. Int. J. Pattern Recogn. Artif. Intell. 6(2-3), 241–256 (1992)
14. Giammarresi, D., Restivo, A.: Two-dimensional languages. In: Rozenberg, G., et al. (eds.) Handbook of Formal Languages, vol. 3, pp. 215–267. Springer (1997)
15. Giammarresi, D., Restivo, A., Seibert, S., Thomas, W.: Monadic second-order logic over rectangular pictures and recognizability by tiling systems. Inf. Comput. 125(1), 32–45 (1996)
16. Harju, T., Karhumäki, J.: Finite transducers and rational transduction. In: Pin, J.E. (ed.) Automata from Mathematics to Application. Eur. Math. Soc. (to appear)
17. Inoue, K., Nakamura, A.: Some properties of two-dimensional on-line tessellation acceptors. Information Sciences 13(2), 95–121 (1977)
18. Latteux, M., Simplot, D.: Recognizable picture languages and domino tiling. Theor. Comput. Sci. 178(1-2), 275–283 (1997)
19. Lonati, V., Pradella, M.: Strategies to scan picture with automata based on Wang tiles. RAIRO - Theor. Inf. and Appl. 45(1), 163–180 (2011)

20. Lonati, V., Pradella, M.: Deterministic recognizability of picture languages with Wang automata. Discr. Math. and Theoret. Comput. Sci. 4, 73–94 (2010)
21. Matz, O.: On Piecewise Testable, Starfree, and Recognizable Picture Languages. In: Nivat, M. (ed.) FoSSaCS 1998. LNCS, vol. 1378, pp. 203–210. Springer, Heidelberg (1998)
22. Medvedev, Y.T.: On the class of events representable in a finite automaton. In: Moore, E.F. (ed.) Sequential Machines - Selected Papers (translated from Russian), pp. 215–227. Addison-Wesley, New York (1964)
23. Sakarovitch, J.: Elements of Automata Theory. Cambridge University Press (2009)

# Flow Decompositions in External Memory

Maxim Babenko

Moscow State University, Yandex LLC
maxim.babenko@gmail.com

**Abstract.** Let $G = (V, E)$ be a digraph with disjoint sets of *sources* $S \subset V$ and *sinks* $T \subset V$ endowed with an $S$–$T$ flow $f\colon E \to \mathbb{Z}_+$. It is a well-known fact that $f$ decomposes into a sum $\sum_{st} f_{st}$ of $s$–$t$ flows $f_{st}$ between all pairs of sources $s \in S$ and sinks $t \in T$. In the usual RAM model, such a decomposition can be found in $O(E \log \frac{V^2}{E})$ time. The present paper concerns the complexity of this problem in the external memory model (introduced by Aggarwal and Vitter). The internal memory algorithm involves random memory access and thus becomes inefficient. We propose two novel methods. The first one requires $O(Sort(E) \log \frac{V^2}{E})$ I/Os and the second one takes $O(Sort(E) \log U)$ expected I/Os (where $U$ denotes the maximum value of $f$).

## 1 Introduction

Network flows is, no doubt, one of best-known and widely-studied subject of combinatorial optimization. The notion of a flow proved to be a useful tool for solving many practical problems.

In this paper we deal with the following flow decomposition problem. Given a digraph $G = (V, E)$ with distinguished sets of *sources* $S$, *sinks* $T$, and a multiterminal $S$–$T$ flow $f$, the goal is to decompose $f$ into a sum of (single-terminal) flows between all possible pairs $(s, t)$ of sources $s \in S$ and sinks $t \in T$.

These decompositions play an important role in some multiflow algorithms (see, e.g., [3]). They can also be used during the second phase of the two-phase push-relabel algorithm [5] to get rid of excesses and turn a maximum preflow into a maximum flow.

The simplest way of finding such a decomposition of $f$ is as follows. Decompose $f$ into elementary flows (i.e. flows along simple $S$–$T$ paths and simple circuits) and collect paths having same endpoints $(s, t)$. This method, however, is not very efficient: a network flow may have no elementary decomposition of size less than $O(VE)$. Hence any direct approach would take at least $O(VE)$ time, which is too much.

A more efficient method was proposed in [3]. It employs edge splitting and takes $O(E \log \frac{V^2}{E})$ time. Interestingly, the logarithmic factor in this bound does not come from any sophisticated data structure.

The goal of the paper is to extend the above approach to the external memory setting. Suppose that graph $G$ is huge, possibly consisting of billions of vertices and edges. Then even storing $G$ in the usual internal memory is unfeasible.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 146–156, 2013.

Following the external memory model (proposed by Agrawal and Vitter [2]) we assume that the algorithm is given an internal memory (RAM) of size $M$ while $G$ resides in an external memory (e.g. HDD) of larger size. The external memory handles read and write requests for contiguous *blocks* of size $B$ (typically $B \ll M$). The external memory is also used for storing the output and and any intermediate data the algorithm finds necessary. In this framework, the complexity is measured as the number of I/Os (reads and writes) the algorithm performs in the worst case for inputs of a given size. The latter complexity may depend on parameters $M$ and $B$.

For example, consider the standard Sorting problem: given a sequence of $N$ integers (each fitting into a machine word), the goal is to reorder its elements in non-decreasing order. The external memory version of Merge-Sort algorithm solves this problem in $O(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$ I/Os. Define $Sort(N) := \frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B}$; the latter can often be found as a part of various complexity estimates. Also set $Scan(N) := \lceil \frac{N}{B} \rceil$, which is the number of I/Os needed to perform a sequential scan of $N$ items in external memory.

Another example, List Ranking problem, is somewhat less trivial. Consider a linked list with $N$ items (given by a collection of pairs $(x, y)$ indicating that $y$ is the immediate successor of $x$). The goal is to compute the *ranks* of its items, i.e. distances from the beginning of the list. While the task can be easily solved in $O(N)$ time in internal memory, this straightforward approach involves $\Theta(N)$ random reads. A substantially better solution, which takes $Sort(N)$ I/Os, is known [2,6].

The above-described external memory model proved to be a useful framework for estimating performance of real-world algorithms dealing with huge data sets. Despite the fact that efficient external memory solutions were found for a large number of classical algorithmic problems, much remains to be explored. In the present paper we address the flow decomposition problem and devise new efficient algorithms for constructing flow decompositions in external memory.

Our results are as follows:

**Theorem 1.** *An arbitrary integer flow in a graph $G = (V, E)$ can be decomposed in $O(Sort(E) \log \frac{V^2}{E})$ I/Os.*

**Theorem 2.** *An arbitrary integer flow with values in range $[0, U]$ in a graph $G = (V, E)$ can be decomposed in $O(Sort(E) \log U)$ expected I/Os.*

The outline of the paper is as follows. In Section 2 we state the decomposition problem formally, briefly describe the edge splitting technique, and estimate its internal memory complexity. In Section 3 we explain how edge splitting can be extended to the external memory setting. Namely, Subsection 3.1 deals with the case of dense graphs. Subsection 3.2 presents the multipass approach suitable for sparse graphs. Subsection 3.3 concerns the case of small flow values. Subsection 3.4 introduces the partial disassembly and randomized partitioning techniques that ultimately lead to an efficient weakly-polynomial method. Finally Subsection 3.5 describes how the backward phase, which is an essential part of all our algorithms, can be implemented in external memory.

## 2 Preliminaries and Internal Memory Algorithms

### 2.1 Network Flows and Decompositions

We use some standard graph-theoretic and algorithmic notation throughout the paper. Let $G = (V, E)$ be a digraph with distinguished subsets of *sources* $S \subset V$ and *sinks* $T \subset V$ ($S \cap T = \emptyset$). We allow parallel edges in $G$ but not loops. Consider a *flow* function $f\colon E \to \mathbb{Z}_+$ and define

$$\mathrm{div}_f(v) := \sum_{e \in \delta^{\mathrm{out}}(v)} f(e) - \sum_{e \in \delta^{\mathrm{in}}(v)} f(e),$$

which is the *divergence* of $f$ at $v$. Here $\delta^{\mathrm{in}}(v)$ (resp. $\delta^{\mathrm{out}}(v)$) denotes the set of edges that enter (resp. leave) $v$.

Informally, $\mathrm{div}_f(v)$ indicates the amount of flow produced at $v$. Then $f$ is called an *$S$–$T$ flow* if $\mathrm{div}_f(v)$ is non-negative for $v \in S$, non-positive for $v \in T$, and zero for $v \in V - (S \cup T)$. If $S = \{s\}$ and $T = \{t\}$ then call $f$ an *$s$–$t$ flow*.

Using the *flow decomposition theorem* [7], one can show that $f$ admits a decomposition

$$f = \sum_{(s,t) \in S \times T} f_{st}, \tag{1}$$

where for each $(s, t) \in S \times T$, $f_{st}$ denotes some integer $s$–$t$ flow in $G$. Such a decomposition is not unique. Throughout the paper, we explore various efficient methods for computing some decomposition (1) of a given $f$.

Cardinalities $|S|$ and $|T|$ are assumed to be fixed constants. In notation involving complexity bounds, we often indicate sets for their cardinalities, e.g. write $O(E)$ instead of $O(|E|)$. We also assume that $|V| \le |E| \le |V|^2/2$ (which can be readily achieved by removing isolated vertices, merging parallel edges, and canceling flows on opposite edges).

### 2.2 Edge Splitting

The following reduction is the cornerstone of all our algorithms. Let us assume that $f$ is positive on all edges of $G$ (for otherwise edges with zero flow can be dropped). Fix an arbitrary *inner* node $v$ (i.e. $v \in V - (S \cup T)$) and suppose that $v$ has an incoming edge $e^+ = (u, v)$ and an outgoing edge $e^- = (v, w)$. Then *splitting* of $e^-$ and $e^+$ is done as follows. First decrease both $f(e^+)$ and $f(e^-)$ by a positive parameter $\varepsilon$ (to be chosen later). Next if $u \neq w$ then add a new edge $e^\bullet = (u, w)$ endowed with flow $f(e^\bullet) := \varepsilon$. (See Fig. 1.)

If $u = w$ then splitting cancels flows on oppositely directed edges. The value of $\varepsilon$ is chosen to be maximum possible preserving non-negativity of $f$, i.e.

$$\varepsilon := \min\left(f(e^+), f(e^-)\right).$$

This transformation decreases at least one of flows $f(e^+)$ and $f(e^-)$ to zero; the corresponding edges are removed from $G$. The transformation also adds at most

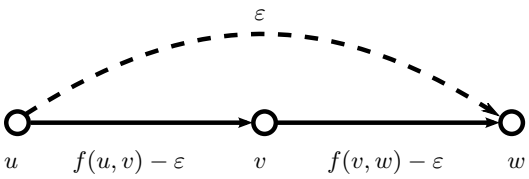



**Fig. 1.** Splitting edges $(u, v)$ and $(v, w)$

one new edge $e^\bullet$ to $G$, so the total number of edges in $G$ does not increase. (In particular, if $G$ contains another edge from $u$ to $w$ before the splitting then $G$ will contain a pair of such parallel edges after the splitting.) It is clear that the updated $f'$ still forms an $S$–$T$ flow in the resulting graph $G'$.

The fundamental property of a splitting is its *reversibility*. Suppose that the updated flow $f'$ in $G'$ is decomposed as (cf. (1))

$$f' = \sum_{(s,t)\in S\times T} f'_{st},$$

where $f'_{st}$ are some $s$–$t$ flows. Then a decomposition of $f$ in $G$ can be constructed as follows. If $u = w$ then we just ignore this splitting. (Hence at the end we obtain a decomposition of some flow $f_0$ that differs from $f$ by a certain *circulation*, i.e. an everywhere conservative function. This circulation can be added to an arbitrary component of the decomposition.) Otherwise edge $e^\bullet$ is present in $G'$ but not in $G$. For each $(s, t) \in S \times T$, we add $\varepsilon = f_{st}(e^\bullet)$ to both $f_{st}(e^+)$ and $f_{st}(e^-)$. This way, we get the desired decomposition for $f$ in $G$. (See [3] for a detailed exposition.)

### 2.3 Vertex Disassembly

Let us choose a sequence splittings that leads to a graph without inner vertices. For the latter one, flow decomposition can be found directly. The whole process of converting the input graph into a trivial one is called the *forward phase*.

Edge splittings are combined into the following *vertex disassembly* operation. Let $v$ be an inner vertex, as before. Consider all incoming edges $e_1^+, \ldots, e_p^+$ of $v$ and also all outgoing edges $e_1^-, \ldots, e_q^-$ of $v$. We scan incoming edges $e_i^+$ and outgoing edges $e_j^-$ and apply splittings to current pairs $(e_i^+, e_j^-)$. Initially $i = 1$ and $j = 1$. When the flow on $e_i^+$ (resp. $e_j^-$) decreases to zero, we increase $i$ (resp. $j$) by 1 thus switching to the next edge. At the end all incoming and outgoing edges get exhausted (since the total flow on incoming edges equals the total flow on outgoing edges), so $v$ becomes isolated. We remove such $v$ and proceed.

The forward phase finishes when no inner vertices remain. At this point the desired decomposition can be found trivially. (For an edge $e = (s', t') \in S \times T$, define $f_{st}(e) = f(e)$ if $(s, t) = (s', t')$ and $f_{st}(e) = 0$ otherwise). The final step involves undoing splittings and converting the decomposition of the final flow into a decomposition of the original one. This part of the algorithm is called the *backward phase*.

### 2.4 Internal Memory Min-Degree Disassembly

Consider the usual internal memory model. For a vertex $v$, the complexity of its disassembly is $O(\deg v)$, where $\deg v$ denotes the *total degree* of $v$, i.e. the total number of edges (both incoming and outgoing) that are incident to $v$.

It remains to choose an order in which to process the vertices to ensure a good complexity bound. A very simple strategy turns out to be useful: on each step pick an inner node with minimum total degree and disassemble it. Let $n$ (resp. $m$) be the number of inner vertices (resp. edges incident to at least one inner vertex) in the original graph $G$. A counting argument implies that there exists an inner vertex of total degree at most $\frac{2m}{n}$. When this vertex gets disassembled, $n$ decreases by 1 and $m$ does not increase. Hence on the second step we pick a vertex of total degree $\frac{2m}{n-1}$, and so on. The total complexity of the forward phase is

$$\frac{2m}{n} + \frac{2m}{n-1} + \ldots + \frac{2m}{1} = O(m \log n) = O(E \log V).$$

The complexity of the backward phase is proportional to the number of splittings during the forward phase and thus is also $O(E \log V)$.

Note that if the initial $G$ is dense then one may maintain $f$ using a $V \times V$-matrix (merging parallel edges as they appear). Also one may be picking inner vertices in an arbitrary order. Disassembling each vertex takes $O(V)$ time and the whole algorithm runs in $O(V^2)$ time. By combining the min-degree disassembly with this flow-matrix approach the complexity can be improved to $O(E \log \frac{V^2}{E})$ (see [3]). We will discuss similar improvements in the context of external memory algorithms in Subsection 3.1 and Subsection 3.2.

## 3 External Memory Algorithms

Let us now consider the external memory model and see how the above algorithms apply here.

### 3.1 Dense Graphs

The case when $G = (V, E)$ is dense is the simplest one since the $O(V^2)$-time algorithm from [3] can be adopted to external memory quite easily. We apply the *time-forward processing* [1], which is a generic technique for simulating internal memory algorithms with a "static" data pipeline in an external memory.

The outline of this method is as follows. Consider an algorithm whose goal is to compute a function $\phi$ on vertices of some auxiliary acyclic *data flow graph* $\Gamma = (V_\Gamma, E_\Gamma)$. We assume that vertices of $\Gamma$ are numbered from 1 to $|V_\Gamma|$ and identify each vertex with its number. For each $v \in V_\Gamma$, value $\phi(v)$ may depend on $v$ and also on values $\phi(u)$ of immediate predecessors $u$ of $v$ (i.e. $(u, v) \in E_\Gamma$). Informally speaking, when $\phi(v)$ gets computed, it is propagated along the edges of $\Gamma$ to all immediate successors of $v$. Graph $\Gamma$ need not be known in advance.

Instead it is constructed incrementally: when $\phi(v)$ is computed the algorithm also lists all immediate successors of $v$ (thus deciding where $\phi(v)$ is propagated to). It is important, however, for the topological ordering of $\Gamma$ to be known in advance. Namely we require $u < v$ to hold for each edge $(u, v)$ of $\Gamma$.

The time-forward processing incurs an overhead of $O(\frac{1}{B} \log_{\frac{M}{B}} \frac{N}{B})$ amortized I/Os per each propagated value (where $N$ is the maximum number of simultaneously propagated values). Its implementation relies on an *external memory heap* $H$. When the algorithm decides to propagate $\phi(v)$ from vertex $v$ to vertex $w$ (with a larger index), it inserts $(w, \phi(v))$ into $H$ (using $w$ as key). Also before processing any vertex $v$ the algorithm extracts all pairs with key $v$ from $H$. This way it "receives" values that were earlier "sent" to $v$ from its immediate predecessors.

We argue that the flow-matrix algorithm from Subsection 2.4 can be run in external memory with the help of time-forward processing. Let us number the vertices of $G$ such that $v_1, \ldots, v_n$ are all inner vertices. Vertices of the data-flow graph $\Gamma$ correspond to $v_1, \ldots, v_n$. To disassemble $v_k$, the algorithm applies splittings to edges $(v_i, v_k)$ and $(v_k, v_j)$ for $i > k$ and $j > k$ (at this point all vertices with numbers less than $k$ are already disassembled). We need to know the current flows $f(v_i, v_k)$ and $f(v_k, v_j)$. Original values of $f$ can be extracted by a row-wise scan of the original flow matrix and its transposed copy. (Constructing these matrices and scanning them takes $O(Sort(V^2))$ I/Os in total.) Changes to the values of $f$ are tracked by means of time-forward processing. Splitting edges $(v_i, v_k)$ and $(v_k, v_j)$ increases $f(v_i, v_j)$ by $\varepsilon$. This increase is propagated either to $v_i$ or to $v_j$, whichever will need this value first (i.e. to $v_i$ if $i < j$ and to $v_j$ otherwise).

The algorithm performs $O(V^2)$ splittings and thus propagates $O(V^2)$ values. The total overhead caused by the time-forward processing is $O(\frac{V^2}{B} \log_{\frac{M}{B}} \frac{V^2}{B}) = O(Sort(V^2))$ I/Os. Another $O(Sort(V^2))$ I/Os are needed to perform the backward phase and to reverse splittings, see Subsection 3.5. The total complexity is $O(Sort(V^2))$ I/Os (which is consistent with Theorem 1 for $E = \Theta(V^2)$).

### 3.2 Sparse Graphs

Now let $G = (V, E)$ be sparse. Ideally we would like to adapt the min-degree disassembly method from Subsection 2.4. The major obstacle here is as follows: while we can easily compute the total degrees of vertices in the initial graph, maintaining these degrees and picking a min-degree vertex on each step is nontrivial. Even worse, the algorithm needs to access the incidence lists of min-degree vertices on each step. Since the order in which vertices are chosen is unpredictable, it is difficult to arrange edge information appropriately. (This issue is quite common to all graph-traversal algorithms that scan vertices in a dynamic order; see [4] for examples.)

We propose the following multi-pass technique to overcome this issue. The algorithm runs a sequence of *passes*. At the beginning of each pass compute the *initial* degrees $\deg_0(v)$ and sort inner vertices $v$ by these degrees thus

constructing a list $L_V = (v_1, \ldots, v_n)$ of all inner vertices (where $\deg_0(v_i) \le \deg_0(v_{i+1})$ for $i = 1, \ldots, n-1$).

Next, prepare the edge set for processing. For each edge $e = (u, v)$ in $G$ we generate tuples $(u, e)$ and $(v, e)$ and sort these tuples by their first component (comparing vertices by indices). The algorithm now scans $L_V$ (in order of increasing degrees) and disassembles certain vertices. For each vertex $v$ to be disassembled, it fetches all incoming and outgoing edges for $v$. (The total cost of preparation and fetching is $O(Sort(E))$ I/Os.)

Recall that $L_V$ reflects the ordering w.r.t. the initial degrees $\deg_0(v)$. The algorithm also maintains the current degrees $\deg(v)$ of vertices $v$ (see details below) and only disassembles $v$ if $\deg(v) \le 2\deg_0(v)$. This degree check guarantees that each pass involves $O(E)$ splittings and (as we will show later) takes $O(Sort(E))$ I/Os.

**Lemma 1.** *Each pass disassembles at least half of all inner nodes.*

*Proof.* Call an inner vertex $v_i$ *good* (resp. *bad*) if $v_i$ was (resp. was not) disassembled during the pass. When $v_i$ gets disassembled its current total degree obeys $\deg(v_i) \le 2\deg_0(v_i)$. Splitting of $v_i$ may increase degrees of other vertices in $G$; their total increase is at most $\deg(v_i)$. Let $d_{ij}$ denote the increase of $\deg(v_j)$ caused by disassembling $v_i$ (i.e. the additional degree propagated from $v_i$ to $v_j$). In particular, $d_{ij}$ can only be positive if $i < j$ and $v_i$ is good. Also $\sum_j d_{ij} \le 2\deg_0(v_i)$.

Fix an index $t = 1, \ldots, n$ and denote $d^t := \sum (d_{ij} : i < t, j \ge t)$ (i.e. the total degree propagated through the "cut" between $\{v_1, \ldots, v_{t-1}\}$ and $\{v_t, \ldots, v_n\}$). Let $n^t_{good}$ (resp. $n^t_{bad}$) be the number of good (resp. bad) vertices among $v_t, \ldots, v_n$. We claim the following:

$$n^t_{bad} - n^t_{good} \le \frac{d^t}{\deg_0(v_t)}. \tag{2}$$

To establish (2), we use induction on $t$ (in the reverse order). For $t = n$, the LHS of (2) is either 0 or 1. Moreover if LHS is 1 then $v_t$ is bad, so when the algorithm comes to processing $v_t$ its current degree is larger than $2\deg_0(v_t)$. Thus $d^t > \deg_0(v_t)$, which implies (2).

Now consider an arbitrary $t \in \{1, \ldots, n-1\}$. Two cases are possible. Let $v_t$ be good. Then $n^t_{bad} = n^{t+1}_{bad}$, $n^t_{good} = n^{t+1}_{good} + 1$. When the algorithm disassembles $v_t$ its current degree is $\deg_0(v_t) + \sum (d_{it} : i < t)$. Hence $v_t$ propagates $\sum (d_{tj} : j > t) \le \deg_0(v_t) + \sum (d_{it} : i < t)$ of degree units to the right. Therefore $d^{t+1} = d^t + \sum (d_{tj} : j > t) - \sum (d_{it} : i < t) \le d^t + \deg_0(v_t)$. Applying the inductive hypothesis for $t + 1$ we get

$$n^t_{bad} - n^t_{good} = n^{t+1}_{bad} - n^{t+1}_{good} - 1 \le \frac{d^t + \deg_0(v_t)}{\deg_0(v_t)} - 1 = \frac{d^t}{\deg_0(v_t)}.$$

Now suppose $v_t$ is bad. Then $n^t_{bad} = n^{t+1}_{bad} + 1$, $n^t_{good} = n^{t+1}_{good}$, and $d^{t+1} \le d^t - \deg_0(v_t)$ (by a similar reasoning as above). Again applying the inductive hypothesis for $t + 1$ we obtain

$$n^t_{bad} - n^t_{good} = n^{t+1}_{bad} - n^{t+1}_{good} + 1 \le \frac{d^t - \deg_0(v_t)}{\deg_0(v_t)} + 1 = \frac{d^t}{\deg_0(v_t)}.$$

Hence the induction follows.

For $t = 1$ we have $d^t = 0$ and thus $n^1_{bad} \le n^1_{good}$. Therefore at least half of the vertices are good. □

Let $n$ (resp. $m$) be the number of nodes (resp. edges) in the original graph. We run passes until less than $\sqrt{m}$ inner nodes remain. A total of $O(\log \frac{n}{\sqrt{m}}) = O(\log \frac{V^2}{E})$ passes are needed, each taking $O(Sort(E))$ I/Os. The remaining dense instance is disassembled by the algorithm from Subsection 3.1; this takes another $O(Sort(E))$ I/Os. The total complexity is $O(Sort(E) \log \frac{V^2}{E})$, as required by Theorem 1.

It remains to explain how the current degrees and the current edge set are tracked. When a vertex $v$ gets disassembled all its incident edges $(u, v)$ and $(v, w)$ vanish. Hence if we scan $u$ or $w$ after $v$ we have to take care not to use these edges again. Similarly $u$ and $w$ may have additional incident edges produced by splittings.

The needed bookkeeping is carried out with the help of the time-forward processing. We maintain a heap $H_d$ to track degree changes. When splitting edges $(u, v)$ and $(v, w)$, we check if degrees of $u$ and $w$ get changed. The degree of $u$ (resp. $w$) increases by one if $(u, v)$ (resp. $(v, w)$) does not vanish during this particular splitting. For each vertex $x$ that gets degree increase and appears *after* $v$ in $L_V$, we insert $(x, +1)$ (using $x$ as a key) into $H_d$. Also before processing a vertex $v$ we extract all propagated changes of $\deg(v)$ from $H_d$. This way the algorithm figures out the up-to-date degree of $v$.

Maintaining the current edge set is done similarly. We keep a heap $H^+_E$ for storing newly added edges. When a new edge $e = (u, w)$ is formed we insert $(u, e)$ to $H^+_E$ if $u$ appears after $v$ in $L_V$ (and is thus processed after $v$) and also $(w, e)$ if $w$ appears after $v$ in $L_V$. When the algorithm starts processing $v$, it first extracts all tuples with key $v$ from $H^+_E$ to see what new edges incident to $v$ have been added. A similar heap $H^-_E$ is used for bookkeeping of edges destroyed by splittings.

Handling a vertex $v$ of current degree $\deg(v)$ results into at most $\deg(v)$ splittings. Since $\deg(v) \le 2 \deg_0(v)$ (for vertices $v$ that get disassembled) each pass leads to $O(E)$ splittings. A splitting causes $O(1)$ heap insertions and the overhead incurred by the time-forward processing is $O(\frac{1}{B} \log_{\frac{M}{B}} \frac{E}{B})$ amortized I/Os per splitting. Hence each pass takes $O(Sort(E))$ I/Os. Combining this with Lemma 1, we obtain Theorem 1.

### 3.3 Pseudo-polynomial Algorithm

If all values of $f$ are small then the above techniques are suboptimal. To see this, suppose that $f(e) = 1$ for all $e \in E$. Then one can decompose $f$ in $O(Sort(E))$ I/Os as follows. Compute a decomposition $\mathcal{D}$ of $f$ into elementary flows along paths and circuits (each carrying a unit of flow). Since flow values are 1 the size

of $\mathcal{D}$ is $O(E)$. For each pair $(s,t) \in S \times T$, combine all $s$–$t$ paths of $\mathcal{D}$ into a flow $f_{st}$. Finally, attach circuits from the above decomposition to an arbitrary component $f_{st}$.

To construct $\mathcal{D}$ consider an inner vertex $v$ and group incoming and outgoing edges into pairs $\{(u,v),(v,w)\}$ (in an arbitrary way). (The number of incoming edges equals the number of outgoing edges, so such a matching exists.) We say that $(v,w)$ is the *successor* of $(u,v)$. Now $E$ partitions into a collection of linked lists. Run List Ranking algorithm from [4] and decompose $E$ into a collection of edge-disjoint paths (connecting terminals) and circuits. The total complexity is $O(Sort(E))$ I/Os.

This method extends to the case of small flows as follows. Define $\sigma(f) := \sum_e f(e)$. Splitting each edge $e$ with flow $f(e)$ into $f(e)$ copies each carrying unit flow, we can solve the decomposition problem in $O(Sort(\sigma))$ I/Os.

### 3.4 Weakly-Polynomial Algorithm

We now improve the results of Subsection 3.3 and give a weakly-polynomial algorithm. The idea is to partition the edge set into a collection of edge-disjoint stars and then run splittings for these stars independently.

We may assume that $G$ does not contain edges connecting pairs of terminals. For each edge $e = (u,w)$, consider a variable $\omega(e) \in \{u,w\}$. We say that $e$ *belongs* to $\omega(e)$. Consider an arbitrary vertex $v$ and let $S_v^+$ (resp. $S_v^-$) be the set of edges that enter (resp. leave) $v$ and belong to $v$. Also define $S_v := S_v^+ \cup S_v^-$. Then $S_v$ is a *star* centered at $v$ and the whole edge set is partitioned into stars. Let us run splittings for each inner vertex $v$ using edges in $S_v$ only. This step is similar to the usual disassembly for $v$ but only works with a subset of edges incident to $v$. We call it a *partial disassembly*.

When the partial disassembly of $v$ is finished, vertex $v$ may still have incident edges left. (E.g. the total flow on incoming edges of $S_v$ may differ from that on outgoing edges.) However observe that $\sigma(f)$ decreases by

$$\Delta(\omega) := \sum (\Delta(\omega,v) : v \text{ is inner}),$$

where for an inner vertex $v$ we define

$$\begin{aligned}\Delta(\omega,v) &:= \min(\Delta^+(\omega,v), \Delta^-(\omega,v)),\\ \Delta^+(\omega,v) &:= \textstyle\sum (f(e) : e \in S_v^+),\\ \Delta^-(\omega,v) &:= \textstyle\sum (f(e) : e \in S_v^-).\end{aligned}$$

The goal is to choose $\omega$ that maximizes $\Delta(\omega)$. A simple randomized strategy proves to be efficient:

**Lemma 2.** *For each $e = (u,w)$, let $\omega(e) \in \{u,w\}$ be chosen randomly, uniformly, and independently. Then $E[\Delta(\omega)] \geq \frac{1}{48}\sigma(f)$.*

*Proof.* Consider an arbitrary inner vertex $v$ and let $a_1^+, \ldots, a_p^+$ (resp. $a_1^-, \ldots, a_q^-$) be flows on edges entering (resp. leaving) $v$. Define $A := \sum_i a_i^+$ (which is also

equal to $\sum_i a_i^-$ due to flow conservation). Let $\xi_1^+, \ldots, \xi_p^+$ and $\xi_1^-, \ldots, \xi_q^-$ be independent Bernoulli variables taking values 0 and 1 with probability $\frac{1}{2}$. Then $\Delta^+(\omega, v) = \sum_i \xi_i^+ a_i^+$ and $\Delta^-(\omega, v) = \sum_i \xi_i^- a_i^-$. Note that $E[\Delta^+(\omega, v)] = \frac{1}{2}A$ and hence $E[A - \Delta^+(\omega, v)] = \frac{1}{2}A$. Now from Markov's inequality it follows that $P\left[A - \Delta^+(\omega, v) \geq \frac{2}{3}A\right] \leq \frac{3}{4}$ and thus $P\left[\Delta^+(\omega, v) \geq \frac{1}{3}A\right] \geq \frac{1}{4}$. Similarly $P\left[\Delta^-(\omega, v) \geq \frac{1}{3}A\right] \geq \frac{1}{4}$ and hence (since $\{\xi_i^+\}$ and $\{\xi_i^-\}$ are independent) we have $P\left[\Delta(\omega, v) \geq \frac{1}{3}A\right] \geq \frac{1}{16}$, which implies $E[\Delta(\omega, v)] \geq \frac{1}{48}A$. Summing over all inner vertices $v$ we obtain the desired bound. □

Hence a randomly chosen $\omega$ decreases $\sigma(f)$ by a constant fraction. Rearranging edges $e$ according to the endpoint $\omega(e)$ they belong to and running partial disassembly for all stars $S_v$ takes $O(Sort(E))$ I/Os. Suppose that initially $f(e) \leq U$ holds for all $e \in E$, thus $\sigma(f) \leq U|E|$. Then after $O(\log U)$ expected iterations $\sigma(f)$ becomes $O(E)$. The final step deals with the remaining flow by running the algorithm from Subsection 3.3. Summing the complexity estimates one gets Theorem 2.

### 3.5 Backward Phase

Let us explain how we deal with the backward phase in external memory. This phase can be implemented to take $O(\frac{s}{B} \log_{\frac{M}{B}} \frac{E}{B})$ I/Os, where $s$ denotes the number of splittings to be reversed. The latter bound coincides with that for the forward phase.

Roughly speaking, the data flow graph is known in advance so we can apply the time-forward processing. The details are as follows. During the forward phase we assign consequent integer numbers to edges of the original graph and also new edges (added during splittings). In particular, the edges of the original $G$ are numbered from 1 to $|E|$, the first split edge is numbered $|E| + 1$ and so on. Let the $i$th edge be denoted by $e_i$. Consider a splitting that was applied to edges $e_i, e_j$ and generated a new edge $e_k$. We record this splitting by a triple $(i, j, k)$. Let $L_S$ be the list of these triples.

We need list $L_S$ to be sorted by $k$. This can be done in $O(Sort(s))$ I/Os but there is a better way. During the forward phase we maintain a heap and for each recorded triple $(i, j, k)$ insert key $k$ into the heap (augmented with data $(i, j)$). Also when splitting is applied to edge $e_l$, we extract and list all items with key less than or equal to $l$. (Since edges are numbered sequentially we cannot have any new keys less that or equal to $l$ added after this point.) It is clear that at any point the total number of keys in the heap does not exceed the number of edges in the current graph, which is bounded by $|E|$.

This way the algorithm sorts all triples $(i, j, k)$ by $k$, as required. Constructing a flow decomposition of $f$ for the final graph is trivial. Thus for each edge $e$ of this final graph, we have a tuple $D(e) = (f_{st}(e) : (s, t) \in S \times T)$. Place these tuples $D(e_k)$ into a heap $H_D$ using $k$ as keys. Scan $L_S$ and join it with the items in $H_D$ on $k$ in decreasing order. Let $k$ be the current key. Let $(i, j, k)$ be the current element in $L_D$. Pop all tuples $D_1, \ldots, D_l$ with key $k$ from $H_D$. Add these tuples $D_1, \ldots, D_l$ componentwise; their sum $D(e_k)$ describes the decomposition for $e_k$.

To undo the splitting that has produced $e_k$, insert two copies of $D(e_k)$ into $H_D$: one with key $i$ and another with key $j$.

Note that for edges $e_k$ of the initial $G$ there are no triples $(i, j, k)$ in $L_D$. Hence when $L_S$ is fully scanned, the algorithm stops adding items to $H_D$ but still continues extracting items from $H_D$ and constructing tuples $D(e_k)$. These tuples give the desired decomposition of $f$ in the initial graph.

Each heap operation takes $O(\frac{1}{B} \log_{\frac{M}{B}} \frac{E}{B})$ I/Os (amortized). Hence the complexity of the whole backward pass is $O(\frac{s}{B} \log_{\frac{M}{B}} \frac{E}{B})$, as claimed.

**Acknowledgement.** The author is thankful to anonymous referees for providing valuable comments and also to Kamil Salikhov and Stepan Artamonov for fruitful discussions.

## References

1. Arge, L.: The Buffer Tree: A New Technique for Optimal I/O-Algorithms (Extended Abstract). In: Sack, J.-R., Akl, S.G., Dehne, F., Santoro, N. (eds.) WADS 1995. LNCS, vol. 955, pp. 334–345. Springer, Heidelberg (1995)
2. Aggarwal, A., Vitter, J.: The input/output complexity of sorting and related problems. Commun. ACM 31(9), 1116–1127 (1988)
3. Babenko, M.A., Karzanov, A.V.: Free multiflows in bidirected and skew-symmetric graphs. Discrete Appl. Math. 155(13), 1715–1730 (2007)
4. Chiang, Y.-J., Goodrich, M.T., Grove, E.F., Tamassia, R., Vengroff, D.E., Vitter, J.S.: External-memory graph algorithms. In: Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1995, pp. 139–149 (1995)
5. Goldberg, A.V., Tarjan, R.E.: A new approach to the maximum-flow problem. Journal of ACM 35(4), 921–940 (1988)
6. Meyer, U., Sanders, P., Sibeyn, J.F. (eds.): Algorithms for Memory Hierarchies. LNCS, vol. 2625. Springer, Heidelberg (2003)
7. Tarjan, R.: Data structures and network algorithms. Society for Industrial and Applied Mathematics, Philadelphia (1983)

# Improved Approximations for Ordered TSP on Near-Metric Graphs*,** (Extended Abstract)

Hans-Joachim Böckenhauer[1] and Monika Steinová[2]

[1] Department of Computer Science, ETH Zurich, Switzerland
hjb@inf.ethz.ch
[2] Google Inc., Zurich, Switzerland
mon@google.com

**Abstract.** The traveling salesman problem with precedence constraints is one of the most important problems in operations research. Here, we consider the well-known variant where a linear order on $k$ special vertices is given that has to be preserved in any feasible Hamiltonian cycle. This problem is called Ordered TSP and we consider it on input instances where the edge-cost function satisfies a $\beta$-relaxed triangle inequality, i. e., where the length of a direct edge cannot exceed the cost of any detour via a third vertex by more than a factor of $\beta > 1$.

We design two new polynomial-time approximation algorithms for this problems. The first algorithm essentially improves over the best previously known algorithm for almost all values of $k$ and $\beta < 1.12651$. The second algorithm gives a further improvement for $2n \geq 11k + 7$ and $\beta < 2/\sqrt{3}$, where $n$ is the number of vertices in the graph.

## 1 Introduction

One of the most prominent problems in operations research is the famous Traveling Salesman Problem (TSP). This problem is studied under many generalizations and modifications [12]. It is well known that TSP is very hard to approximate, even very restricted special cases remain $\mathcal{APX}$-hard. If no restrictions are given on the edge-cost function, the problem is not approximable by any polynomial (dependent on the input size). On the other hand, if the edge costs satisfy the standard triangle inequality, the problem becomes approximable within a factor 1.5 due to Christofides' algorithm [9]. If the metrics is even more restricted and the vertices are placed into a fixed-dimension Euclidean space, the problem turns out to have a PTAS [2]. For a brief overview on the different notions of hardness and approximability of problems, refer, e. g., to [13].

In many applications, additional restrictions are given for the set of feasible solutions, one of the most common type of such restrictions are precedence constraints, i. e., demanding that certain vertices have to be visited before certain

* This work was partially supported by SNF grant 200021-132510/1.
** This work was done while the second author was staying at ETH Zurich.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 157–168, 2013.

others. In general, such precedence constraints can be expressed by an arbitrary partial ordering on the vertices. However, in this paper, we focus on instances where a linear order on a subset of $k$ vertices is given as a part of the input and every feasible solution has to contain these special vertices in the prescribed order. We call this variant of the problem Ordered TSP or $k$-OTSP for short.

As we have discussed above, the properties of the edge-cost function can influence the hardness of a problem. Here, we consider the $\beta$-relaxed triangle inequality that can be formulated as $cost(uv) \leq \beta(cost(uw) + cost(wv))$, where $u, v, w$ are arbitrary vertices of the graph and $\beta \geq 1$ is a constant. For $\beta = 1$ this yields the well-known standard triangle inequality. If $\beta > 1$, the inequality is referred to as $\beta$-relaxed triangle inequality. The instances where the standard or relaxed triangle inequality holds are called *metric* or *near-metric* instances, respectively. By the relaxed triangle inequality, we model how close the edge-cost function is to being metric. The cases where $\beta$ is close to 1 (and hence the edge-cost function is almost a triangle inequality) are most interesting for the following two reasons. First of all, many real world instances tend to have $\beta$ equal or close to 1 which supports the study of these cases. Secondly, from the theoretical viewpoint, the study of such instances allows us to better understand how the triangle inequality influences the hardness of problems and the stability of approximation algorithms [11,13].

In the metric case, the best known approximation algorithm for $k$-OTSP with approximation ratio $2.5 - 2/k$ is due to [7,8]. In this paper, we focus on the version of $k$-OTSP where a $\beta$-relaxed triangle inequality holds which is denoted as $k$-$\Delta_\beta$OTSP. In [6], the first polynomial-time approximation algorithm solving $k$-$\Delta_\beta$OTSP with ratio $((k+1) \cdot \min\{4\beta^{2+\log_2(k-1)}, 1.5\beta^{3+\log_2(k-1)}, (\beta+1)\beta^{2+\log_2(k-1)}\})$ is presented. A slightly different approach was used later in [7] to improve the approximability of $k$-$\Delta_\beta$OTSP to ratio $k\beta^{\log_2(3k-3)}$ and further to $k\beta^{\log_2(\lfloor 3k/2 \rfloor + 1)}$ in the journal version [8].

In this paper, we introduce several new observations that allow us to improve the original approximation algorithm for $k$-$\Delta_\beta$OTSP from [6]. Our improved polynomial-time approximation algorithm yields a ratio of $3/2 \cdot \lceil k/2 \rceil \cdot \beta^{2+\log_2(2\lceil k/2 \rceil + k - 3)}$ which is better than the currently best known approximation algorithm for all instances with $\beta \leq (3/4)^{1/\log_2(3/16)} < 1.12651$, except for those with very small $k$. The pairs $(\beta, k)$ for which we improved the approximation ratio are depicted in Figure 3. By a more involved analysis, for instances where the number of vertices is at least $5.5k + 3.5$, we achieve a further improvement even for $\beta < 2/\sqrt{3} \approx 1.1547$. We believe, that some of our observations can be generalized to other problems where the $\beta$-relaxed triangle inequality holds.

The paper is organized as follows. In Section 2, we give the formal definition of the problem and fix our notation; in Section 3 we present our approximation algorithm for $k$-$\Delta_\beta$OTSP. Section 4 discusses the more involved analysis of our algorithm and in Section 5 we conclude the paper with open problems. Due to space restrictions, some of the proofs are omitted in this extended abstract.

## 2 Preliminaries

In this paper, we use standard notions of graph theory [10,14]. For an undirected graph $G = (V, E)$, the edge representation $\{u, v\}$ is simplified into an unordered pair $uv$ $(u, v \in V)$. The set of vertices and the set of edges of a graph $G$ are denoted as $V(G)$ and $E(G)$, respectively. A list of vertices $v_0, \ldots, v_n$, where each vertex is used at most once and such that $v_i v_{i+1} \in E$ $(0 \leq i < n)$, is called a *path* with the *endpoints* $v_0$ and $v_n$. The length of such path is $n$. Once the endpoints of a path are identical, i. e., $v_0 = v_n$, we speak about a *cycle* of length $n$. A *Hamiltonian cycle* of a graph $G$ is a cycle of length $|V(G)|$. A graph consisting of all possible edges is referred to be *complete*. In a graph $G$, a *subpath* of a path $P = v_0, \ldots, v_n$ is a path $v_{i_0}, \ldots, v_{i_k}$, where $i_0 < i_1 < \cdots < i_k$ and $v_{i_j} v_{i_{j+1}} \in E(G)$ for all $0 \leq j < k$. We refer to the edges $v_{i_j} v_{i_{j+1}}$ of a subpath to be a *bypass* or a *jump* and the vertices between $v_{i_j}$ and $v_{i_{j+1}}$ (exclusively) in the path $P$ to be *bypassed* by the edge $v_{i_j} v_{i_{j+1}}$.

The *cost* of a subgraph of a graph is the sum of the costs of its edges. For simplicity, we write $cost(H)$ when we mean the cost of a subgraph $H$.

The problem of finding a minimum-cost Hamiltonian cycle in a near-metric complete undirected graph is called *Near-Metric Traveling Salesman Problem* and is referred to as $\Delta_\beta$-TSP. If we fix, in addition, an ordered sequence of $k$ *special vertices* that have to be present in the minimum-cost Hamiltonian cycle in the given order, we speak about the *Ordered Traveling Salesman Problem*. If the underlying graph is near-metric, we refer to the problem as the *Ordered Near-metric Traveling Salesman Problem* and we denote it as $k$-$\Delta_\beta$OTSP.

Note that $k$-$\Delta_\beta$OTSP with $k \leq 3$ turns into the standard Near-metric Traveling Salesman Problem. Hence, in this paper, we assume that $k \geq 4$.

For the Near-metric Traveling Salesman Problem, we know three different approximation algorithms, each is best for certain values of $\beta$.

**Theorem 1 (Andreae [1], Bender & Chekuri [4], Böckenhauer et al. [5]).** *There is a polynomial-time approximation algorithm solving* $\Delta_\beta$-TSP *with approximation ratio*

$$M := \min\left\{\frac{3}{2}\beta^2, \beta^2 + \beta, 4\beta\right\}.$$

*Furthermore,* $M = \begin{cases} \frac{3}{2}\beta^2 & \text{if} \quad \beta \leq 2, \\ \beta^2 + \beta & \text{if} \quad 2 \leq \beta \leq 3, \\ 4\beta & \text{if} \quad \beta \geq 3. \end{cases}$

To be able to estimate the cost of bypasses of a path in near-metric complete graphs, we recall a very basic lemma.

**Lemma 1 (Bandelt et al., [3]).** *Let $G$ be a complete graph with edge-cost function cost that satisfies the $\beta$-relaxed triangle inequality. Let $W = v_0, \ldots, v_\ell$ be a path in $G$. For $0 = a_0 < a_1 < \cdots < a_q = \ell$, where $1 \leq q \leq \ell$ holds, let $m := \max_{0 \leq i < q}\{a_{i+1} - a_i\}$. Then*

$$\sum_{i=0}^{q-1} cost(v_{a_i}, v_{a_{i+1}}) \leq \beta^{\log_2 m} \cdot cost(W).$$

Intuitively, this lemma says that the length of a jump over a path of length $\ell$ can be bounded from above by $\beta^{\log_2 \ell}$ times the cost of the bypassed path.

The algorithm that we present in the following section builds on the first known approximation algorithm for $k$-$\Delta_\beta$OTSP designed by Böckenhauer et al. in [6] as Algorithm 2. Its general idea is as follows. First, connect all vertices into a Hamiltonian cycle. Secondly, color the non-special vertices cyclically by $k+1$ colors. Then connect pairs of the consecutive special vertices by traversing the Hamiltonian cycle and by including the vertices of just one fixed color (for each pair of vertices a different one) in between the two special vertices to a subpath. Use another subpath to connect the special vertex $s_k$ with a vertex $x$ adjacent to $s_1$ via vertices of color $k$. Start the last subpath in the vertex $x$ and traverse the entire cycle to the vertex $s_1$ and use the remaining vertices not yet included to the subpaths. Finally, in the end, connect all the subpaths sequentially into a feasible solution. In this paper we refer to this algorithm as HC-Algorithm.

A couple of years after the HC-Algorithm was introduced, the currently best known approximation algorithm was presented in [7] and a slightly improved version in [8]. The algorithm is based on a different underlying structure. Instead of a Hamiltonian cycle, a cheaper minimum spanning tree is used as an underlying structure. The vertices of the tree are colored again by $k$ colors and the special vertices are then connected by the unique paths in the tree via vertices of fixed colors. The vertices not lying on these paths are in the end attached to the cycle using som special rules and a feasible solution is obtained. The approximation ratio achieved by this algorithm is summarized by the following theorem.

**Theorem 2 (Böckenhauer et al., [8]).** *There exists a* $k \cdot \beta^{\log_2(\lfloor 3k/2 \rfloor + 1)}$*-approximation algorithm that solves* $k$-$\Delta_\beta$OTSP *in polynomial time.*

## 3 Building on an Underlying Hamiltonian Cycle

As a stepping stone to improve the approximability of $k$-$\Delta_\beta$OTSP ($k \geq 4$), we use the HC-Algorithm. We modify this algorithm by incorporating three new observations which yields Algorithm 1.

The general idea of both HC-Algorithm and Algorithm 1 is the same. We first build a Hamiltonian cycle $C$ that does not respect the given order of the special vertices. The cycle is then used to construct paths between consecutive special vertices that are then merged together into a cycle where the special vertices are present in the right order. However, during the process of the path construction, some vertices might be omitted and hence are not present anymore in the new cycle. These vertices are in the end added into the cycle and the required Hamiltonian cycle is obtained.

The three observations that are improving HC-Algorithm are as follows.

– We can always connect three special vertices present on a cycle in the prescribed order by a single path – the path starts in the first vertex and traverses, either clockwise or counterclockwise, the cycle through the second prescribed vertex to the last vertex.

**Algorithm 1.** (Approximation algorithm for $k$-$\Delta_\beta$OTSP)

**Input:** A complete graph $G = (V, E)$ with edge cost function $cost : E \to \mathbb{Q}^+$ that satisfies the $\beta$-relaxed triangle inequality ($\beta > 1$), and a sequence of special vertices $(s_1, \ldots, s_k)$ from $V$ ($k \geq 4$).

1: Using some constant-approximation algorithm, construct an approximate $\Delta_\beta$-TSP solution $C$ on $(G, cost)$, disregarding the order on $(s_1, \ldots, s_k)$.

2: Let $P$ be one of the two paths that may be obtained by removing one of the edges incident with $s_1$ in $C$, and let $W = (w_1, \ldots, w_{n-k})$ be the sequence of the non-special vertices in $P$, beginning with the non-special vertex closest to $s_1$ in $P$. Let $f$ be a $\lceil k/2 \rceil$-cyclic coloring of non-special vertices of $C$ defined as follows: $f : V \to \{-1, 0, \ldots, \lceil k/2 \rceil - 1\}$ where $f(s_i) := -1$, for all $1 \leq i \leq k$, and $f(w_{i+1}) := i \bmod \lceil k/2 \rceil$, for all $0 \leq i < n - k$.

3: For $1 \leq i \leq \lceil k/2 \rceil - 1$, let $L_{i-1}$ be the subpath in $C$ (either clockwise or counter-clockwise) from $s_{2i-1}$ to $s_{2i}$ and then to $s_{2i+1}$, restricted to $s_{2i-1}$, $s_{2i}$, and $s_{2i+1}$ plus all vertices $w$ with $f(w) = i - 1$ in between $s_{2i-1}$ and $s_{2i+1}$.

4: If $k$ is odd, let $L_{\lceil k/2 \rceil - 1}$ be the subpath in $C$ from $s_k$ to $s_1$ that is restricted to $s_k$, $s_1$ and the non-special vertices $w$ that have color $f(w) = \lceil k/2 \rceil - 1$.
If $k$ is even, let $L_{\lceil k/2 \rceil - 1}$ be the subpath in $C$ from $s_{k-1}$ through $s_k$ to $s_1$, restricted to $s_{k-1}$, $s_k$, $s_1$ and the non-special vertices $w$ with $f(w) = \lceil k/2 \rceil - 1$.

5: Create new subpaths $L'_0, \ldots, L'_{\lceil k/2 \rceil - 1}$ from $L_0, \ldots, L_{\lceil k/2 \rceil - 1}$ by including the non-special vertices bypassed by all $L_i$ as shown in Figure 1 and discussed later.

6: Let $Y$ be the path of $C$ containing all the vertices that are not included in any of the $L'_i$. If such a path does not exist, skip this step. Otherwise, let $x$ and $y$ be the two endpoints of $Y$ and let $s_{2\gamma+3}$ be the special vertex that is neighboring with $x$ in $C$ (see Figure 2 for details).
Remove the endpoint $s_{2\gamma+3}$ from the subpath $L'_\gamma$ and extend it by including all the vertices $w$ of $Y$ in the direction from $x$ to $y$ colored by $f(w) = \gamma$ and terminate it by the vertex $y$ (if its color differs from $\gamma$).
Extend the subpath $L'_{\gamma+1}$ from $s_{2\gamma+3}$ by including all the vertices of $Y$ from $x$ to $y$ that were not included into $L'_\gamma$ and terminate the modified subpath in vertex $y$.

7: Merge the subpaths $L'_0, \ldots, L'_{\lceil k/2 \rceil - 1}$ into a cycle $H$.

**Output:** The Hamiltonian cycle $H$.

- The vertices that are bypassed by a subpath, but not included into any subpath, can be just included to the bypassing subpath – the cost of the solution is not increased this way.
- Let $H$ be a Hamiltonian cycle on a complete weighted graph and let $L_0, \ldots, L_{p-1}$ be some paths on $H$ that can be sequentially merged in their endpoints into a walk $W$, i. e., for all $0 \leq i < p$, paths $L_i$ and $L_{(i+1) \bmod p}$ share one endpoint. Then the vertices of $H$ that are not covered by $W$ form a single path $Y$ of $H$. Moreover, the adjacent vertex of an endpoint of $Y$ in $H$ is a vertex in which two paths $L_i$ and $L_{i+1}$ are merged. The path $Y$ split into two subpaths can be used to extend the paths $L_i$ and $L_{i+1}$ such that all vertices of $H$ are covered.

Note that, when the special vertices are connected by paths in the HC-Algorithm, the cycle is traversed always in one direction so that the special

vertex $s_1$ is never bypassed. In our algorithm, instead, we connect an ordered triple of special vertices together in one traversal of the cycle. (If $k$ is odd, the last path is connecting just two special vertices – $s_k$ and $s_1$.) In such a traversal, we cannot always avoid the bypasses of the special vertex $s_1$ in the cycle.

The advantage of this approach is that connecting special vertices in triples is roughly halving the estimated cost of the solution: In our algorithm we always bound the cost of two paths connecting a pair of special vertices (except possible the last one) by the cost of the underlying structure (the cycle $C$). This approach of estimating the cost of more structures at once is unique and was not applied in any of the previous approximation algorithms for $k$-$\Delta_\beta$OTSP. In all the previous approximation algorithms the cost of each path connecting a pair of special vertices was bounded by the cost of the entire underlying structure. Furthermore, the side effect of our approach is that our algorithm needs only $\lceil k/2 \rceil$ vertex-colors instead of $k+1$.

The drawback of this approach is that the bypasses might be slightly longer. More precisely, once we are passing the vertex $s_1$ in our traversal, we might have a *cut in the coloring*: the segment of the cycle between $s_1$ and the next vertex of the same color in the other direction than the one used for the coloring might miss vertices of some colors. This happens if the number of non-special vertices in the cycle is not divisible by the number of colors and thus the cycle does not contain the same number of vertices of each color. Here, the algorithm might be forced to bypass at most two vertices of the same color.

**Lemma 2.** *In steps 3 and 4 of Algorithm 1, at most* $2\lceil k/2 \rceil + k - 3$ *edges of* $C$ *are bypassed between two consecutive vertices of subpath* $L_i$*, for all* $0 \leq i \leq \lceil k/2 \rceil - 1$*. Furthermore, a bypass of at most* $2\lceil k/2 \rceil + k - 3$ *edges can occur only if the vertex* $s_1$ *is bypassed. All other bypasses jump over at most* $\lceil k/2 \rceil + k - 2$ *edges.*

The second observation is used in step 5 of Algorithm 1 in the same way as it was used in step 4 of Algorithm 3 in [7] – including bypassed vertices into any subpath that is bypassing them cannot increase the cost of the subpath. The bottom line why this is so, is that the inclusion of a bypassed vertex preserves the vertex order in the subpath and does not increase the number of vertices which are bypassed. The process of the inclusion of bypassed vertices into subpaths $L_i$ in the construction of the subpaths $L'_i$ is depicted in Figure 1 and is described in detail in [7].

**Lemma 3.** *The cost of the paths* $L'_i$ *constructed in step 5 from paths* $L_i$ *cannot increase, i. e., for all* $i$*,* $0 \leq i \leq \lceil k/2 \rceil - 1$*,* $cost(L'_i) \leq cost(L_i)$*.*

At this point, if we sequentially merge the subpaths $L'_0, \ldots, L'_{\lceil k/2 \rceil - 1}$ by their endpoints, we obtain a cycle $Q$ that contains each vertex at most once and that respects the given order on the special vertices. However, some vertices of $C$ might not be present in $Q$ as no subpath is bypassing them. This case is covered by the third observation – see Figure 2 for details. The vertices not included in $Q$ must form a single path $Y$ in $C$ and are included into two consecutive subpaths as follows. Since $Y$, with the endpoints $x$ and $y$, is a path of $C$, there must exist

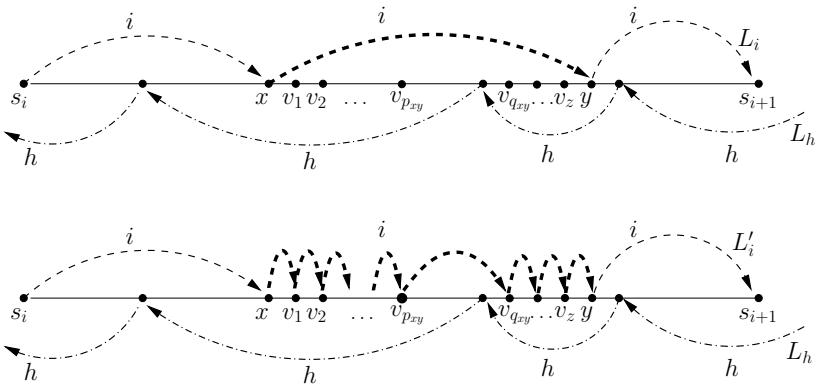



**Fig. 1.** Modifying path $L_i$ to path $L'_i$: All vertices which are bypassed by the edge $xy$ of $L_i$ and not yet included into other subpaths are connected to subpath $L_i$

subpaths $L'_\gamma$ and $L'_{\gamma+1}$ with an endpoint $s_{2\gamma+3}$ that is adjacent with $x$. We split $Y$ into two subpaths and extend each of the subpaths $L'_\gamma$ and $L'_{\gamma+1}$ by one of these split subpaths of $Y$. The outcome of these extensions is that $y$ becomes the endpoint of both subpaths and each vertex of $Y$ is included into exactly one of the extended subpaths. Notice that, in the extension of the subpath $L'_\gamma$, its special vertex $s_{2\gamma+3}$ is removed and then it is continued by inclusion of the vertices of the color $\gamma$ into the subpath up to the vertex $y$ (inclusively). Since $Y$ was not covered by $L'_\gamma$, the vertices of color $\gamma$ are not already included in any subpath and this operation is sound. The other extension can be evaluated in the same way. The subpath $L'_{\gamma+1}$ (which still contains the special vertex $s_{2\gamma+3}$) is extended by including all the remaining vertices of $Y$ into the subpath and is terminated in $y$. By the same reasoning as above, this operation is also sound. For our convenience of presenting the statements comprehensively, we call the extended subpaths still $L'_\gamma$ and $L'_{\gamma+1}$.

After the transformation of the subpaths $L'_\gamma$ and $L'_{\gamma+1}$ in step 6, observe that two consecutive vertices in the extended subpaths are bypassing at most $2\lceil k/2\rceil + k - 3$ edges in $C$. This is true as (1) the non-extended parts have this property as discussed above; (2) in the extended part of $L'_\gamma$, only the vertices of color $\gamma$ are used and the distance between the last vertex of the color $\gamma$ and $y$, which may be of a different color, cannot be too high as well; (3) the extended part of $L'_{\gamma+1}$ from $s_{2\gamma+3}$ to $y$ contains all the remaining vertices. In particular, this also includes the vertices of the color $\gamma+1$. Two consecutive vertices of this color are at distance at most $2\lceil k/2\rceil + k - 3$ in the cycle $C$. Note that this is also true for the distance between the last vertex $x$ of the extended subpath and the vertex $s_{2\gamma+3}$. The inclusion of the vertices of the other colors in between the vertices of color $\gamma+1$ cannot increase the distance.

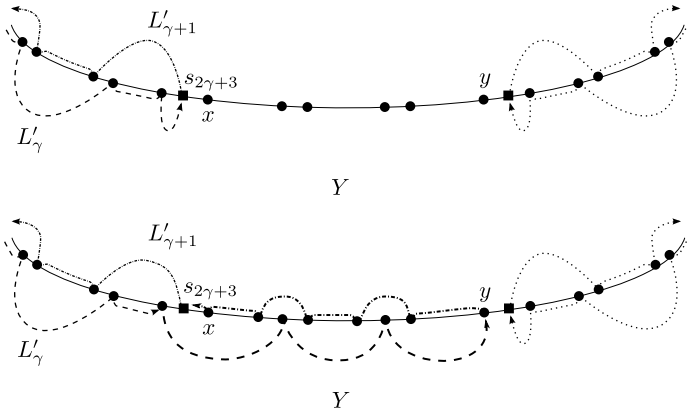



**Fig. 2.** An extension of the paths $L'_\gamma$ and $L'_{\gamma+1}$ by the path $Y$ in step 6 of Algorithm 1. Squares denote special vertices, dots denote non-special vertices. The terminal vertex $s_{2\gamma+3}$ is removed from $L_\gamma$, the subpath $L_\gamma$ is extended by inclusion of all the vertices of color $\gamma$ in $Y$ and terminated by the vertex $y$. The subpath $L'_{\gamma+1}$, in which the special vertex $s_{2\gamma+3}$ is preserved, is extended by inclusion of all the remaining vertices of $Y$ and is terminated by the vertex $y$.

**Lemma 4.** *The distance between two consecutive vertices in subpaths $L'_i$ (for $0 \le i \le \lceil k/2 \rceil - 1$ and $i \notin \{\gamma, \gamma+1\}$) and in the extended subpaths $L'_\gamma$ and $L'_{\gamma+1}$ is at most $2\lceil k/2 \rceil + k - 3$ on the cycle $C$.*
*Furthermore, a bypass of at most $2\lceil k/2 \rceil + k - 3$ edges can occur only if the vertex $s_1$ is bypassed (i. e., if the bypass occurs on the color cut). All other bypasses jump over at most $\lceil k/2 \rceil + k - 2$ edges.*

*Proof.* The lemma is proven by the discussion above and a similar reasoning as we used in Lemma 2. □

From the description above, it is clear that the cycle $H$ computed by our algorithm is a feasible solution, i. e., each vertex of the graph is included in $H$ once and the special vertices are present in the given order. From now on, we omit to write colorings and indexing modulo $\lceil k/2 \rceil$, when it is clear from the context. Now we are ready to estimate the approximation ratio of our algorithm.

**Theorem 3.** *For $k \ge 4$, Algorithm 1 computes, in polynomial time, an approximate solution for $k$-$\Delta_\beta$OTSP with a ratio of at most*

$$\left\lceil \frac{k}{2} \right\rceil \min \left\{ 4\beta^{1+\log_2(2\lceil k/2 \rceil + k - 3)}, \frac{3}{2}\beta^{2+\log_2(2\lceil k/2 \rceil + k - 3)}, (\beta+1)\beta^{1+\log_2(2\lceil k/2 \rceil + k - 3)} \right\}.$$

The following theorem characterizes the instances for which our algorithm improves over the $k\beta^{\log_2(\lfloor 3k/2 \rfloor + 1)}$-approximation algorithm of [8].

**Theorem 4.** *For any $1 < \beta < (3/4)^{\frac{1}{\log_2 3/16}} < 1.12651$, there exists a fixed $k_0 \ge 4$ such that, for all $k \ge k_0$, the approximation ratio of Algorithm 1 is better*

*than the ratio achieved by Algorithm 2 of [8] for the pairs* $(\beta, k)$*. In these cases, the approximation ratio of Algorithm 1 is at most*

$$\left\lceil \frac{k}{2} \right\rceil \cdot \frac{3}{2} \cdot \beta^{2+\log_2(2\lceil k/2 \rceil + k - 3)}.$$

Our calculations from the proof of Theorem 4 imply that, for arbitrary $1 < \beta < \left(\frac{3}{4}\right)^{\frac{1}{\log_2 3/16}}$, there exists an initial value $k_0 \geq 4$ such that, for all $k \geq k_0$, the approximation ratio of Algorithm 1 is better than the ratio of the previously best known algorithm. The pairs $(k, \beta)$ for which we obtain an improvement are depicted in Figure 3.

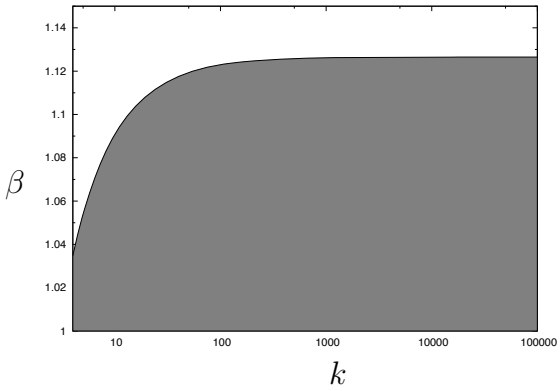



**Fig. 3.** The graph shows the pairs $(k, \beta)$ for which Algorithm 1 has provably better approximation ratio than the best known approximation algorithm

## 4 Improving the Coloring

We can identify three parts of Algorithm 1 which contribute to the overall approximation ratio $3/2\lceil k/2 \rceil \cdot \beta^{2+\log_2(2\lceil k/2 \rceil + k - 3)}$. The factor $3/2\beta^2$ is given by the algorithm computing cycle $C$ in step 1. The factor $\lceil k/2 \rceil$ comes from structural properties of a cycle (i. e., the number of times $cost(C)$ is used to estimate the connected special vertices is halved as triples of such vertices can be connected at once). We do not aim here for the improvement of these two factors. The last part produces the factor $\beta^{\log_2(2\lceil k/2 \rceil + k - 3)}$. Its exponent corresponds to the maximal length of a bypass in the solution and is dependent on the used vertex-coloring procedure. In this section, we introduce another observation which allows us to decrease the length of bypasses and hence to improve the approximation ratio of our algorithm.

All our bypasses in Algorithm 1 that are not bypassing the color cut around $s_1$ contain at most $\lceil k/2 \rceil + k - 3$ vertices. Only the bypasses that occur around this cut are the bottleneck of our estimations and are increasing the estimated length of the bypasses from $\lceil k/2 \rceil + k - 2$ to $2\lceil k/2 \rceil + k - 3$ in all our calculations.

For the ease of the presentation, we call a bypass that bypasses also vertex $s_1$ (and hence the color cut) a *color-cut bypass*.

We estimated that each color-cut bypass contains at most $2 \cdot (\lceil k/2 \rceil - 1) + x$ vertices, where $x$ is the number of special vertices that are bypassed. Hence, the main idea of the improvement is to choose a vertex from which the coloring is started (and which also places the color cut into the cycle) such that each color-cut bypass contains a minimum number of special vertices.

Let $d$ be a positive integer whose precise value is specified later. Let us divide the cycle $C$ into $d$ parts of roughly the same size. From the pigeon-hole principle, one of these parts contains at most $\lfloor k/d \rfloor$ special vertices and at least $\lfloor n/d \rfloor$ of all the vertices of $C$. If this part of the cycle contains at least $3 \cdot \lceil k/2 \rceil - 1$ non-special vertices, we can place the color cut into this part such that starting and ending vertices of all color-cut bypasses are present in this part. Then, the number of bypassed vertices in each color-cut bypass is at most $2 \cdot (\lceil k/2 \rceil - 1) + \lfloor k/d \rfloor$. Hence, if we choose $d$ to be large (but still $d \leq n$), the number of vertices bypassed by each color-cut bypass converges to $k + \frac{k}{n}$.

The last part that we have to show is that, indeed, the chosen part contains at least $3 \cdot \lceil k/2 \rceil - 1$ vertices with respect to $d$ and $n$. We can estimate the number of non-special vertices in the chosen part of $C$ which is at least

$$\left\lfloor \frac{n}{d} \right\rfloor - \left\lfloor \frac{k}{d} \right\rfloor \geq \frac{n-d+1}{d} - \frac{k}{d} = \frac{n-d+1-k}{d}.$$

Hence, if

$$\frac{n-d+1-k}{d} \geq 3 \cdot \left\lceil \frac{k}{2} \right\rceil - 1,$$

we can place the color cut into the chosen part. The inequality above is true if

$$\frac{n-d+1-k}{d} \geq 3 \cdot \frac{k+1}{2} - 1,$$

and hence,

$$2 \cdot (n-d+1-k) \geq 3d \cdot (k+1) - 2d.$$

If we further modify the inequality, we are able to bound $d$ by

$$d \leq \frac{2(n+1-k)}{3k+3}. \tag{1}$$

Any positive integer value $d$ satisfying the inequality (1) can be used to divide the cycle into $d$ parts, one of these parts contains only $\lfloor k/d \rfloor$ special vertices which allows each color-cut bypass to contain at most $2 \cdot (\lceil k/2 \rceil - 1) + \lfloor k/d \rfloor$ vertices. To minimize the length of the color-cut bypass, we need to minimize $\lfloor k/d \rfloor$ and hence $d$ has to be maximal.

Since our coloring achieves bypasses of $\lceil k/2 \rceil + k - 3$ vertices for each segment of the cycle without the color cut, the partition of the cycle $C$ into many parts does not always help to decrease the length of bypasses.

If $d = 2$ (and $n \geq 4k + 2$), the color-cut bypasses contain at most $2(\lceil k/2 \rceil - 1) + \lfloor k/2 \rfloor = \lceil k/2 \rceil + k - 2$ vertices which is by 1 vertex more than the number of vertices bypassed in the remainder of the cycle. Therefore, if $n \geq 4k + 2$, Algorithm 1 enhanced by the special placement of the color cut has an approximation ratio of

$$\left\lceil \frac{k}{2} \right\rceil \cdot \min \left\{ 4\beta^{1+\log_2(\lceil k/2 \rceil + k - 1)}, \frac{3}{2}\beta^{2+\log_2(\lceil k/2 \rceil + k - 1)}, (\beta + 1)\beta^{1+\log_2(\lceil k/2 \rceil + k - 1)} \right\}.$$

If $k \geq 4$, then $\lfloor k/3 \rfloor \leq \lfloor k/2 \rfloor - 1$. Thus, if in addition $d = 3$ (and hence $2n \geq 11k + 7$), the color-cut bypasses become at most as long as the non-color-cut bypasses and the enhanced Algorithm 1 is achieving ratio

$$\left\lceil \frac{k}{2} \right\rceil \cdot \min \left\{ 4\beta^{1+\log_2(\lceil k/2 \rceil + k - 2)}, \frac{3}{2}\beta^{2+\log_2(\lceil k/2 \rceil + k - 2)}, (\beta + 1)\beta^{1+\log_2(\lceil k/2 \rceil + k - 2)} \right\}.$$

For the latter approximation ratio with bypasses of length $\lceil k/2 \rceil + k - 2$ edges, we can do similar calculations as we did in Section 3 when the $\frac{3}{2}\beta^2$-approximation algorithm for $\Delta_\beta$-TSP from Böckenhauer et al. [5] is plugged into step 1 which yields the following theorem.

**Theorem 5.** *For any $1 < \beta < \frac{2}{\sqrt{3}}$, there exists a fixed $k_0 \geq 4$ such that, for all $k \geq k_0$, the approximation ratio of Algorithm 1 with improved coloring is better than the ratio achieved by Algorithm 2 of [8] for the pairs $(\beta, k)$. In these cases, the approximation ratio of Algorithm 1 is at most*

$$\left\lceil \frac{k}{2} \right\rceil \cdot \frac{3}{2} \cdot \beta^{2+\log_2(\lceil k/2 \rceil + k - 2)}.$$

## 5 Conclusion

In this paper, we investigated approximation algorithms for the Ordered Nearmetric Traveling Salesman Problem ($k$-$\Delta_\beta$OTSP). The first, general approximation algorithm improves the best known approximation ratio for instances with $\beta < 1.12651$ and the last ratio is an improvement for $\beta < 2/\sqrt{3}$. In both cases, this is true for all $k$ except for the first finitely many small values.

Even though the obtained improvement of our algorithm does not cover the entire space of instances, it broadens our knowledge on the approximability of the most interesting instances – those that are close to being metric.

There are still several open questions related to $k$-$\Delta_\beta$OTSP and $k$-OTSP. First of all, we are not aware of any hard instances for approximation algorithms which could suggest whether the estimated ratios are tight or there is still space for improvements. Furthermore, our algorithm, same as the algorithms of [6,7], is using a "static" coloring of vertices that does not reflect the actual situation in the initial Hamiltonian cycle. We believe that the approximation ratio, especially the part that is dependent on the length of bypasses, could be improved by a more sophisticated coloring.

Our enhancement from Section 4 for $n$-vertex graphs can be applied only if $n \geq 4k + 2$ (or, to be a little bit better, if $2n \geq 11k + 7$), i. e., if the number of special vertices is not too high. However, it seems that the cases where $n$ and $k$ are of the same order of magnitude are the hardest ones. For these cases, one may consider an approximation algorithm that is different in its used structures from all our approaches. We can even generalize the idea and consider the problem on instances where the number of special vertices $k$ is a fraction of $n$.

## References

1. Andreae, T.: On the traveling salesman problem restricted to inputs satisfying a relaxed triangle inequality. Networks 38(2), 59–67 (2001)
2. Arora, S.: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. Journal of the ACM 45(5), 753–782 (1998)
3. Bandelt, H.-J., Crama, Y., Spieksma, F.C.R.: Approximation algorithms for multi-dimensional assignment problems with decomposable costs. Discrete Applied Mathematics 49(1-3), 25–50 (1994)
4. Bender, M., Chekuri, C.: Performance guarantees for TSP with a parametrized triangle inequality. Information Processing Letters 73, 17–21 (2000)
5. 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. In: Bongiovanni, G., Petreschi, R., Gambosi, G. (eds.) CIAC 2000. LNCS, vol. 1767, pp. 72–86. Springer, Heidelberg (2000)
6. Böckenhauer, H.-J., Hromkovič, J., Kneis, J., Kupke, J.: On the Approximation Hardness of Some Generalizations of TSP (Extended Abstract). In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 184–195. Springer, Heidelberg (2006)
7. Böckenhauer, H.-J., Klasing, R., Mömke, T., Steinová, M.: Improved Approximations for TSP with Simple Precedence Constraints. In: Calamoneri, T., Diaz, J. (eds.) CIAC 2010. LNCS, vol. 6078, pp. 61–72. Springer, Heidelberg (2010)
8. Böckenhauer, H.-J., Mömke, T., Steinová, M.: Improved approximations for TSP with simple precedence constraints. Journal of Discrete Algorithms (to appear)
9. Christofides, N.: Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University (1976)
10. Diestel, R.: Graph Theory, 3rd edn. Springer (2005)
11. Gonzalez, T.F. (ed.): Handbook of Approximation Algorithms and Metaheuristics. Chapman & Hall/CRC Computer and Information Science Series. Chapman & Hall/CRC (2007)
12. Gutin, G., Punnen, A.P. (eds.): The Traveling Salesman Problem and Its Variations. Combinatorial Optimization. Springer, New York (2007)
13. Hromkovič, J.: Algorithmics for Hard Problems. Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics. Springer (2003)
14. West, D.B.: Introduction to Graph Theory, 2nd edn. Prentice Hall Inc., Upper Saddle River (2000)

# Asymptotic Risk Analysis for Trust and Reputation Systems

Michele Boreale[1] and Alessandro Celestini[2,⋆]

[1] Università di Firenze, Italy
michele.boreale@unifi.it
[2] IMT Institute for Advanced Studies Lucca, Piazza S. Ponziano 6, 551 00 Lucca, Italy
alessandro.celestini@imtlucca.it

**Abstract.** Trust and reputation systems are decision support tools used to drive parties' interactions on the basis of parties' reputation. In such systems, parties rate with each other after each interaction. Reputation scores for each ratee are computed via reputation functions on the basis of collected ratings. We propose a general framework based on Bayesian decision theory for the assessment of such systems, with respect to the number of available ratings. Given a reputation function $g$ and $n$ independent ratings, one is interested in the value of the loss a user may incur by relying on the ratee's reputation as computed by the system. To this purpose, we study the behaviour of both Bayes and frequentist risk of reputation functions with respect to the number of available observations. We provide results that characterise the asymptotic behaviour of these two risks, describing their limits values and the exact exponential rate of convergence. One result of this analysis is that decision functions based on Maximum-Likelihood are asymptotically optimal. We also illustrate these results through a set of numerical simulations.

**Keywords:** trust, reputation, information theory, Bayesian decision theory.

## 1 Introduction

Trust and reputation systems are used as decision support tools for different applications in several contexts. Probably the best known applications are related to e-commerce: well-known examples in this context are the auction site eBay and the online shop Amazon, cf. [10].

Trust management systems are used in many other contexts and applications, where huge amount of data related to reputations of peers are usually available, such as ad-hoc networks [18], P2P networks [25,24] and sensor networks [4]. The idea at the base of trust and reputation systems is to let users of the system, the raters, rate the provided services, or ratees, after each interaction. Then other users or the parties themselves may use aggregate ratings to compute reputation scores for a given party: such values are used to drive parties' interactions. This approach to trust managing is referred to as *computational trust*. Whereas traditional credential-based approaches [6,17] rely on

⋆ Corresponding author: Alessandro Celestini, IMT Institute for Advanced Studies Lucca, Italy.
Work partially supported by the EU project ASCENS under the FET open initiative in FP7.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 169–181, 2013.

access control policies and/or use of certificates for evaluating parties' trustworthiness, in computational trust parties' trustworthiness is evaluated by on the basis of the parties' past behaviour.

In this paper, we focus on *probabilistic* trust [11,5,16], which represents a specific approach to computational trust. The basic postulate of probabilistic trust is that the behaviour of each party can be modeled as a probability distribution, drawn from a given family, over a certain set of interaction *outcomes* – *success/failure* being the simplest case. Once this postulate is accepted, the task of computing reputation scores boils down to inferring the *true* distribution's parameters for a given party. Information about party's past behaviour is used for parameters inference: rating values are treated as statistical data.

The potential usefulness and applicability of probabilistic trust is by now demonstrated by a variety of tools that have been *experimentally* tested in several contexts. One important example is the work on the TRAVOS system [23].

On the contrary, there are very few *analytical* results on the behaviour of such systems – with the notable exception of the work by Sassone and collaborators [20], which is discussed in the concluding section. Examples of questions that could be addressed by an analytical approach are: How do we quantify the *confidence* in the decisions calculated by the system? And how is this confidence related to such parameters as decision strategy and number of available ratings? Is there an optimal strategy that maximizes confidence as more and more information becomes available?

In this paper, we intend to address the above questions, and propose a framework to analyze probabilistic trust systems based on *bayesian decision theory* [19,9,13]. A prominent aspect of this approach is the use of a priori probability distributions to model prior belief on the set of possible parties' behaviours. However, we also consider confidence measures that dispense with such prior belief. We study the behaviour of trust and reputation systems relying on the concept of *loss* function. We quantify confidence in the system in terms of *risk* quantities based on *expected (a.k.a. bayes)* and *worst-case* loss. We study the behaviour of these quantities with respect to the available information, that is number of available rating values. Our results allow to characterize the asymptotic behaviour of probabilistic trust systems. In particular, we show how to determine the limit value of both bayes and worst risks, and their exact exponential rates of convergence, in the case of *independent and identically distributed* (i.i.d) observations.

In particular, we show that decision functions based on maximum likelihood or maximum a posteriori decision functions are asymptotically optimal. We complement these theoretical results with set of numerical simulations.

The rest of the paper is organized as follows. Section 2 introduces some terminology and preliminary notions. Section 3 presents the formal setting of our framework. Section 4 introduces the confidence measures based on expected and bayes risk. Section 5 reports the main results on the analysis of the asymptotic behaviour of reputation functions. Section 6 presents a set of numerical simulations whose outcome support our theoretical study.

Finally, Section 7 contain some concluding remarks and discussion of related work. In particular, we touch upon an extension of the framework where raters tend to under- or over-evaluate their interactions with the ratees.

## 2 Notations and Preliminary Notions

Let $X$ be a random variable taking values in $\mathcal{X}$: we say that $X$ is distributed according to a probability distribution $p(\cdot)$ if for each $x \in \mathcal{X}$, $Pr(X = x) = p(x)$, and we write $X \sim p(\cdot)$. The *support* of $p(\cdot)$ is defined as $\text{supp}(p) \triangleq \{x \in \mathcal{X} | p(x) > 0\}$. We let $p^n(\cdot)$ denote the $n$-th extension of $p(\cdot)$, defined as $p^n(x^n) \triangleq \prod_{i=1}^n p(x_i)$, where $x^n = (x_1, x_2, ..., x_n)$; this is in turn a probability distribution on the set $\mathcal{X}^n$. For any $A \subseteq \mathcal{X}$ we let $p(A)$ denote $\sum_{x \in A} p(x)$.

When $A \subseteq \mathcal{X}^n$ and $n$ is clear from the context, we shall abbreviate $p^n(A)$ as just $p(A)$. Given a sequence $x^n = (x_1, x_2, ..., x_n) \in \mathcal{X}^n$ ($n \geq 1$), we denote by $t_{x^n}$ its *type* or *empirical distribution*, which is a probability distribution on $\mathcal{X}$ defined thus: $t_{x^n}(x) \triangleq \frac{\mathrm{n}(x|x^n)}{n}$, where $\mathrm{n}(x|x^n)$ denotes the number of occurrences of $x$ in $x^n$.

Given two probability distributions $p(\cdot)$ and $q(\cdot)$ on $\mathcal{X}$, the relative entropy, or *Kullback-Leibler divergence*, (KL-divergence) between $p(\cdot)$ and $q(\cdot)$ is defined as

$$D(p(\cdot) \parallel q(\cdot)) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \tag{1}$$

with the convention that: $0 \log \frac{0}{0} = 0$, $0 \log \frac{0}{q(\cdot)} = 0$ and $p(\cdot) \log \frac{p(\cdot)}{0} = \infty$. It can be shown that KL-divergence is always nonnegative and is 0 if and only if $p(\cdot) = q(\cdot)$. KL-divergence is not symmetric and does not satisfy the triangle inequality.

Let $\Theta$ be a set of parameters: we let $\{p(\cdot|\theta)\}_{\theta \in \Theta}$ denote a parametrized family of probability distributions. When convenient, we shall denote a member of this family, $p(\cdot|\theta)$, as just $p_\theta$. Given a sequence $x^n$ that is a realization of $n$ i.i.d. random variables $X^n = X_1, ..., X_n$, with $X_i \sim p(\cdot|\theta)$, a standard problem is to decide which of the distributions $p(\cdot|\theta)$ generated the data. This is a general hypothesis-testing problem, where the distributions $p(\cdot|\theta)$, $\theta \in \Theta$, are the hypotheses: the classical, binary formulation is given for $|\Theta| = 2$. We represent the decision-making process by a guessing function $g : \mathcal{X}^n \to \Theta$ and we define the error probability for an hypothesis $\theta$ as follows. For $n \geq 1$ and each $\theta$, let $A_\theta^{(n)} \triangleq g^{-1}(\theta) \subseteq \mathcal{X}^n$ be the *acceptance region* for hypothesis $\theta$ (relatively to $g$). Then the probability of error for $\theta$ is (by $A^{\mathsf{c}}$ we denote the complement of set $A$)

$$P_\theta^{(g)}(n) \triangleq p(A_\theta^{(n)\mathsf{c}}) . \tag{2}$$

In a Bayesian framework an a priori probability $\pi(\theta)$ is assigned to each hypothesis, and the overall error probability is defined as the average, assuming $\Theta$ is discrete:

$$P_e^{(g)}(n) \triangleq \sum_\theta \pi(\theta) P_\theta^{(g)}(n) . \tag{3}$$

It is well-known (see e.g. [22]) that optimal strategies, i.e. strategies $g$ minimizing the error probability $P_e^{(g)}(n)$, are obtained when $g$ satisfies the *Maximum A Posteriori* (MAP) criterion

In this case, provided $p(\cdot|\theta) \neq p(\cdot|\theta')$ for $\theta \neq \theta'$, it holds that as $n \to +\infty$, $P_e^{(g)}(n) \to 0$. What is also important, though, is to characterize *how fast* the probability of error approaches 0. Intuitively, we want to be able to determine an exponent $\rho \geq 0$ such that, for large $n$, $P_e^{(g)}(n) \approx 2^{-n\rho}$. To this purpose, we introduce the notion of rate for a generic non-negative real-valued function $f$.

**Definition 1 (Rate).** *Let* $f : \mathbb{N} \to \mathbb{R}^+$ *be a nonnegative function. Assume* $\gamma = \lim_{n\to\infty} f(n)$ *exists. Then, provided the following limit exists, we define the following nonnegative quantity:*

$$\mathrm{rate}(f) \stackrel{\triangle}{=} \lim_{n\to\infty} -\frac{1}{n}\log|f(n)-\gamma|.$$

*This is also written as* $|f(n)-\gamma| \doteq 2^{-n\rho}$*, where* $\rho = \mathrm{rate}(f)$.

Intuitively, $\mathrm{rate}(f) = \rho$ means that, for large $n$, $|f(n)-\gamma| \approx 2^{-n\rho}$.

Note that we do allow $\mathrm{rate}(f) = +\infty$, a case that arises for example when $f(n)$ is a constant function.

The rate of decrease of $P_e^{(g)}(n)$ is given by *Chernoff Information*. Given two probability distributions $p(\cdot)$, $q(\cdot)$ on $\mathcal{X}$, we let their Chernoff Information be

$$C(p(\cdot),q(\cdot)) \stackrel{\triangle}{=} -\min_{0\le\lambda\le 1}\log\Big(\sum_{x\in\mathrm{supp}(p)\cap\mathrm{supp}(q)} p^{\lambda}(x)q^{1-\lambda}(x)\Big), \tag{4}$$

where we stipulate that $C(p,q) = +\infty$ if $\mathrm{supp}(p)\cap\mathrm{supp}(q) = \emptyset$. Here $C(p(\cdot),q(\cdot))$ can be thought of as a sort of distance between $p(\cdot)$ and $q(\cdot)$: the more $p(\cdot)$ and $q(\cdot)$ are far apart, the less observations are needed to discriminate between them. Assume we are in the binary case, $\Theta = \{\theta_1,\theta_2\}$ and let $p_i = p(\cdot|\theta_i)$ for $i = 1,2$. Then a well-known result gives us the rate of convergence for the probability of error, with the proviso that $\pi(\theta_1) > 0$ and $\pi(\theta_2) > 0$ (cf. [22]): $P_e^{(g)}(n) \doteq 2^{-nC(p_1,p_2)}$ (here we stipulate $2^{-\infty} = 0$). Note that this rate does not depend on the prior distribution $\pi(\cdot)$ on $\{\theta_1,\theta_2\}$, but only on the probability distributions $p_1$ and $p_2$. This result extends to the case $|\Theta| < +\infty$, it is enough to replace $C(p_1,p_2)$ by $\min_{\theta\neq\theta'} C(p(\cdot|\theta),p(\cdot|\theta'))$, thus (see [14,3]): $P_e^{(g)}(n) \doteq 2^{-n\min_{\theta\neq\theta'} C(p(\cdot|\theta),p(\cdot|\theta'))}$ with the understanding that, in the min, $\pi(\theta) > 0$ and $\pi(\theta') > 0$.

## 3 A Bayesian Framework for Trust and Reputation

In the following section we discuss the main features of trust and reputation systems and we introduce our framework based on Bayesian decision theory for modelling such systems. The formal framework is composed by two main components: an *observation framework*, which describes how observations are probabilistically generated, and a *decision framework*, which describes how decisions are taken. Two essential ingredients of the latter are *loss* and *decision functions*.

Parties in a trust and reputation system are free to interact and rate with each other. After each interaction, a *rater* assigns a score to a *ratee*. We denote by $O = \{o_1,...,o_m\}$ a finite, non-empty set of rating values: in the rest of the paper we use the terms *outcomes*, *observables* and *rating values* interchangeably. We focus on trust and reputation systems where the behaviour/reputation of each party is modelled by a probability distribution on $O$.

**Definition 2 (Observation System).** *An* observation system *is a quadruple* $\mathcal{S} = (O,\Theta,\mathcal{F},\pi(\cdot))$*, composed by a finite non-empty set of* observations $O$*, a set of* world states *or* parameters $\Theta$*, a set* $\mathcal{F} = \{p(\cdot|\theta)\}_{\theta\in\Theta}$ *of probability distributions on* $O$ *indexed by* $\Theta$*, and an a priori probability measure* $\pi(\cdot)$ *on* $\Theta$.

In a trust setting, the a priori measure $\pi(\cdot)$ expresses the user's belief over possible behaviours $\theta \in \Theta$ of the observed system. The data-model $\mathcal{F}$ represents how data are generated: the value $p(o|\theta)$ denotes the probability of observing a rating value $o \in O$ in an interaction with a party whose behaviour is $\theta \in \Theta$.

*Remark 1.* The set $\Theta$ can be in principle discrete or continuous. In the following sections, unless otherwise stated, it is assumed that $\Theta$ is in fact *finite*. Moreover, we shall always assume the following conditions that simplify our treatment, with no significant loss of generality:

- for each $\theta$, the distribution $p(\cdot|\theta)$ has full support, that is $p(o|\theta) > 0$ for each $o$. We shall sometimes denote the distribution $p(\cdot|\theta)$ as $p_\theta$;
- for any pair of parameters $\theta \neq \theta'$, one has $p_\theta \neq p_{\theta'}$.

*Example 1.* A very simple possibility, but one widely used in practice, is to assume a set of binary outcomes representing *success* and *failure*, say $O = \{o, \bar{o}\}$, which are generated according to a Bernoulli distribution: $p(o|\theta) \triangleq \theta$ and $p(\bar{o}|\theta) \triangleq 1-\theta$, where $\theta \in \Theta \subseteq (0, 1)$. Another possibility is to rate a service' quality by an integer value in a range of $n + 1$ values, $O = \{0, 1, ..., n\}$. In this case, it is sometimes sensible to model the parties' behaviour by binomial distribution $\mathcal{B}in(n, \theta)$, with $\theta \in \Theta \subseteq (0, 1)$ (somehow the discrete analog of a continuous Gaussian distribution). That is, the probability of an outcome $o \in O$ for an interaction with a party with a behaviour $\theta$ is $p(o|\theta) \triangleq \binom{n}{o}\theta^o(1-\theta)^{n-o}$. In the following, we shall mostly concentrate on discretized sets of parameters. E.g. $\Theta = \{0.1, 0.2, \cdots, 0.9\}$.

Reputation scores used to drive parties' interactions are computed on the basis of a party's past behaviour, given as a sequence of observations $o^n = (o_1, ..., o_n)$ $(n \geq 1)$. These may derive from direct interaction of the user, or be acquired by the user by different means (e.g. they may be gathered and provided by an online evaluation support system). Irrespective of how $o^n$ is acquired, the basic idea – which will be studied analytically in Section 5 – is that the sequence $o^n$ is a realization of a random vector $O^n = (O_1, ...O_n)$, where the r.v. $O_i$'s are i.i.d. given $\theta \in \Theta$: $O_i \sim p(\cdot|\theta)$. A decision $d = g(o^n) \in \mathcal{D}$ is taken on the basis of the past behaviour. This decision may however incur in a loss $\mathrm{L}(\theta, d)$, depending on the true behaviour $\theta$ of the ratee and on the taken decision itself. These concepts are formalized below.

**Definition 3 (Decision Framework).** *A* decision framework *is a quadruple* $\mathcal{DF} = (\mathcal{S}, \mathcal{D}, \mathrm{L}(\cdot,\cdot), \{g^{(n)}\}_{n\geq 1})$*, composed by an observation system* $\mathcal{S} = (O, \Theta, \mathcal{F}, \pi(\cdot))$*, a decision set* $\mathcal{D}$*, a loss function* $\mathrm{L}(\cdot,\cdot) = \Theta \times \mathcal{D} \to \mathbb{R}^+$*, and a family of decision functions* $\{g^{(n)}\}_{n\geq 1}$*, one for each* $n \geq 1$*,* $g^{(n)} : O^n \to \mathcal{D}$*.*

$\mathrm{L}(\theta, d)$ is a (in general, user-defined) function that quantifies the loss incurred when making a decision $d \in \mathcal{D}$, given that the real behaviour of the party is $\theta \in \Theta$.

In the bayesian decision theory, the decision-making process is formalized via decision functions. For any $n$, a decision function is a function $g^{(n)} : O^n \to \mathcal{D}$ (the superscript $^{(n)}$ will be normally omitted when $n$ is clear from the context).

There are two main types of decisions one may wish to make when interacting with a party: evaluation of the party's behaviour, hence reputation; or the prediction of the

outcome of the next interaction. In order to distinguish between these two cases, we define two instances of the above decision framework that differ by the definition of the decision set. In a *reputation framework*, one has $\mathcal{D} = \Theta$, that is the decision is made about the behaviour. In a *prediction* framework, one has $\mathcal{D} = O$, that is the decision is made about the outcome of the (next) interaction.

Loss functions evaluate the consequences of possible decisions associating a loss to each decision. The choice of such functions depend on the application at hand and is, ultimately, responsibility of the user of the reputation system. For example, there could be monetary or economic losses connected to taking a given decision in a given state of the world.

Below, we shall limit ourselves to indicate a few concrete examples of such loss functions.

For a reputation framework ($\mathcal{D} = \Theta$), one's objective may be to minimize a sort of distance between the true behaviour $\theta$ and the inferred reputation $\theta'$. A common way to do so is to employ KL-divergence as a measure of distance between probability distributions, and set: $\mathrm{L}(\theta, \theta') \triangleq D(p(\cdot|\theta')||p(\cdot|\theta))$. This loss function takes on a (proper) minimum value when $\theta = \theta'$, with $\mathrm{L}(\theta, \theta') = 0$. It is also a legitimate choice to consider the two arguments exchanged: $\mathrm{L}(\theta, \theta') \triangleq D(p(\cdot|\theta)||p(\cdot|\theta'))$, although the significance of this measure is less clear to us. Other measures can be based on distance between probability distributions seen as real valued vectors; one we shall consider is norm-1 distance: $\mathrm{L}(\theta, \theta') \triangleq ||p(\cdot|\theta) - p(\cdot|\theta')||_1$. Finally, if $\Theta \subseteq \mathbb{R}$, a sensible choice might be $\mathrm{L}(\theta, \theta') \triangleq |\theta - \theta'|$. For a prediction framework ($\mathcal{D} = O$), one generally considers loss functions that are minimized when the probability of an outcome is maximum. One such loss function is $\mathrm{L}(\theta, o) \triangleq -\log p(o|\theta)$, that is, the Shannon information content of an outcome $o$: less probable the outcome $o$, more information/surprise (and loss) it conveys. Such function takes its minimum value for $o_\theta = \arg\max_{o \in O} p(o|\theta)$ with $\mathrm{L}(\theta, o_\theta) = -\log \max_{o \in O} p(o|\theta) = H_\infty(p_\theta)$. Here, $H_\infty$ denotes *min-entropy* of a probability distribution/random variable.

As discussed in this Section, in trust and reputation systems the decision-making process consists of choosing a behaviour $\theta \in \Theta$ or a rating value $o \in O$. We model such a process via decision functions. It is possible to formulate decision functions through classical statistical inference procedures such as *Maximum Likelihood* (ML) and *Maximum A Posteriori* (MAP) estimation (see e.g. [8]). Essentially, the ML rule yields the $\theta$ maximizing the likelihood of the observed $o^n$ - equivalently, minimizing the KL distance between the empirical distribution of $o^n$ and $p_\theta$ (see [12]). The MAP rule yields the $\theta$ whose posterior probability given $o^n$ is maximum.

**Definition 4 (ML and MAP decision function).** *Let $o^n = (o_1, ..., o_n)$ be a sequence of observations. Then a* ML *decision function $g^{(\mathrm{ML})} : O^n \to \Theta$ satisfies*

$$g^{(\mathrm{ML})}(o^n) = \mathrm{argmin}_\theta D(t_{o^n}||p(\cdot|\theta)).$$

*A* MAP *decision function $g^{(\mathrm{MAP})} : O^n \to \Theta$ satisfies*

$$g^{(\mathrm{MAP})}(o^n) = \theta \text{ implies } p(\theta|o^n) \geq p(\theta'|o^n) \text{ for each } \theta' \in \Theta.$$

Note that implementation of the MAP rule implies knowledge of the a priori distribution $\pi(\cdot)$, which may be sometimes difficult or impossible to estimate. Fortunately, MAP and

ML are asymptotically equivalent (and optimal). Despotovic and Aberer in [5] propose the use of reputation functions based on ML estimation; we use such functions in Sections 5. Similarly, Jøsang and Ismail in [11] propose the use of a reputation function based on the beta probability density function $Beta(\alpha, \beta)$.

## 4 Evaluation of Reputation Functions

In this section we introduce two measures for the evaluation of reputation functions, based on the expected and worst loss, respectively. We also discuss the notion of rate of convergence. In what follows, we fix a generic decision framework $\mathcal{S}$; for each $\theta \in \Theta$, we assume a decision $d_\theta \in \mathcal{D}$ exists that minimizes the loss given $\theta$: $d_\theta = \text{argmin}_d \mathrm{L}(\theta, d)$. Let us first consider the definition of *frequentist risk*. For a parameter $\theta \in \Theta$, the frequentist risk associated to a decision function $g$ after $n$ observation is just the expected loss computed over $O^n$, that is explicitly

$$R^n(\theta, g) \triangleq \sum_{o^n \in O^n} p(o^n|\theta)\mathrm{L}(\theta, g(o^n)).$$

Relying on this definition we introduce first the bayes risk.

**Definition 5 (Bayes risk).** *Let $g : O^n \to \mathcal{D}$ be a decision function and $\pi(\cdot)$ a prior probability distribution on the parameters set $\Theta$. The* bayes risk *associated to $g$ after $n$ observations is the expected loss with respect to $\theta$*

$$r^n(\pi, g) \triangleq \mathbb{E}_\pi[R^n(\theta, g)] = \sum_\theta \pi(\theta) R^n(\theta, g).$$

*The* minimum bayes risk *is defined as $r^* \triangleq \sum_\Theta \pi(\theta)\mathrm{L}(\theta, d_\theta)$.*

The bayes risk is the expected value of the risk $R^n(\theta, g)$, computed with respect to the a priori distribution $\pi(\cdot)$, that represents the a priori information over possible behaviours in the system.

The second measure we introduce is the worst risk.

**Definition 6 (Worst risk).** *Let $g : O^n \to \mathcal{D}$ be a decision function and $\Theta$ the parameters set. The* worst risk *associated to $g$ after $n$ observations is given by*

$$w^n(g) \triangleq \max_{\theta \in \Theta} R^n(\theta, g).$$

*The* minimum worst risk *is defined as $w^* \triangleq \max_{\theta \in \Theta} \mathrm{L}(\theta, d_\theta)$*

The worst risk is thence the maximum risk $R^n(\theta, g)$ over possible parameters $\theta \in \Theta$.

*Example 2.* We compute the values of both minimum bayes and worst risks for two specific loss functions. We use the definitions of loss funcions given in previous sections. The first definition is for a reputation framework, the second for a prediction framework. Let $\mathrm{L}(\theta, \theta') = D(p(\cdot|\theta')||p(\cdot|\theta))$ be the loss function for a reputation framework. Then we have

$$r^* = \sum_\Theta \pi(\theta) D(p(\cdot|\theta)||p(\cdot|\theta)) = 0\,, \qquad w^* = \max_{\theta \in \Theta} D(p(\cdot|\theta)||p(\cdot|\theta)) = 0.$$

Let now $\mathrm{L}(\theta, o) = -\log p(o|\theta)$ be the loss function for a prediction framework. We have

$$r^* = \sum_\Theta \pi(\theta) \log \frac{1}{p(o_\theta|\theta)} = \sum_\Theta \pi(\theta) H_\infty(p_\theta)\,, \qquad w^* = \max_{\theta \in \Theta} \log \frac{1}{p(o_\theta|\theta)} = \max_{\theta \in \Theta} H_\infty(p_\theta).$$

## 5 Results

In this section we discuss some results about the convergence of risk quantities $r^n = r^n(\pi, g)$ and $w^n = w^(g)$ to minimum bayes risk and worst risk, respectively, and their rates of convergence. We first examine risks in a reputation and in a decision framework; then briefly discuss exponential bounds on the probability of exceeding a given loss.

Given a decision framework, it is important to establish not only the *limit* of the risk functions, $r^n$ and $w^n$, as the number $n$ of available ratings grows, but also *how fast* this limit is approached. The *rate* of convergence tells us how fast this limit is approached. The concept of rate is important for two reasons. Firstly, it is desirable to distinguish between reputation functions with different rates, as a reputation function with a high rate may require considerably less observations in order to achieve an improvement of the risks value, compared to a reputation function with a low rate. Secondly, knowledge of the rate will allow us to obtain quick and accurate estimations of the risk functions $r^n$ and $w^n$ depending on $n$.

In what follows, we fix a generic reputation framework $\mathcal{RF} = (\mathcal{S}, \Theta, \mathrm{L}(\cdot,\cdot), \{g^{(n)}\}_{n\geq 1})$. In order to discuss our results in the simplest possible form, we shall assume $\Theta$ is finite, and that $\mathrm{L}(\theta,\theta') > \mathrm{L}(\theta,\theta)$ for each $\theta \neq \theta'$. We let

$R \triangleq \min_{\theta\neq\theta'} C(p_\theta, p_{\theta'})$

be the least Chernoff Information between any pair of distinct distributions $p_\theta$ and $p_{\theta'}$ in $\mathcal{F} = \{p(\cdot|\theta) \mid \theta \in \Theta\}$.

We shall often abbreviate $r^n(\pi, g)$ as just $r^n$, similarly for $w^n$.

The following theorem states that the best achievable rate of convergence to the minimum values of any decision function, for both bayes and worst risks, is bounded above by $R$. The proof can be found in the

extended version available online [2].

**Theorem 1.** *Assume* $\lim r^n(\pi, g)$ *exists. Then*

- $\lim r^n(\pi, g) \geq r^*$*;*
- *if* $\lim r^n(\pi, g) = r^*$ *then* $\mathrm{rate}(r^n(\pi, g)) \leq R$*, if defined.*

*Similarly for the worst risks* $w^n$ *and* $w^*$.

The following theorem confirms that both MAP and ML are asymptotically optimal decision functions. Such functions achieve minimum loss value and maximum rate of convergence. The proof can be found in the extended version available online [2].

**Theorem 2.** *Assume* $g$ *is either a* MAP *or a* ML *decision function. Then* $\lim_n r^n = r^*$ *and moreover* $\mathrm{rate}(r^n) = R$*. Similarly for* $w^n$.

In essence, the above results can be summarized by saying that, under an optimal decision function, $r^n$ behaves as $\approx r^* + 2^{-nR}$, and $w^n$ as $\approx w^* + 2^{-nR}$.

We fix a generic prediction framework $\mathcal{PF} = (\mathcal{S}, \mathcal{O}, \mathrm{L}(\cdot,\cdot), \{h^{(n)}\}_{n\geq 1})$. For a distribution $p$, let $\mathrm{Argmax}(p)$ denote the set of observables maximizing $p(o)$. We assume $\Theta$ is finite, and that for each $\theta$ there is $o_\theta \in \mathrm{Argmax}(p_\theta)$ such that $\mathrm{L}(\theta, o_\theta) > \mathrm{L}(\theta, o)$ for each $o \neq o_\theta$. Like in the preceding subsection, we let $R$ denote the minimum Chernoff Information between any pair of distinct distributions on $\mathcal{O}$.

We shall limit our discussion to the following result of practical interest, which describes the form of the (optimal) prediction function: as expected, given $o^n$, one has to first identify the underlying distribution $p_\theta$ and then take the observable that maximizes this distribution. The proof goes along the lines of the results in the preceding subsection and is omitted.

**Theorem 3.** *Let $g = \{g^{(n)}\}_{n\geq 1}$ be a family of* ML *decision functions. Assume $h^{(n)}(o^n) \triangleq o_\theta$ where $\theta \triangleq g^{(n)}(o^n)$. Then $\lim r^n = r^*$ and $\mathrm{rate}(r^n) = R$. Similarly for $w^n$.*

We discuss here the probability that the loss deviates from its minimal value above a given threshold. These results apply to the case $g$ is the ML or MAP decision functions. Such results do not depend on the parameter $\theta$ but only on the number $n$ of observations and the threshold value fixed for the loss. The results also apply to the case when $\Theta$ is continuous. We begin by considering the KL-loss function (proofs can be found in the extended version available online [2]).

**Theorem 4.** *Let $g$ be the* ML *or* MAP *decision function. Let $\mathrm{L}(\theta, \theta') \triangleq D(p(\cdot|\theta')||p(\cdot|\theta))$ be the KL loss function. Fix any $\theta \in \Theta$ and assume $O^n$ is a n-sequence of random variables i.i.d. given $\theta$, that is $O_i \sim p(\cdot|\theta)$. Let $\epsilon > 0$. Then*

$$\Pr(\mathrm{L}(\theta, g(O^n)) > \epsilon) \leq (n+1)^{|O|} 2^{-n\epsilon}\,.$$

The above result can be easily extended to the case of norm-1 loss function.

**Corollary 1.** *Let $g$ be the* ML *or* MAP *decision function. Let $\mathrm{L}(\theta, \theta') = \|p(\cdot|\theta') - p(\cdot|\theta)\|_1$, be the norm-1 distance loss function. Fix any $\theta \in \Theta$ and assume $O^n$ is a n-sequence of random variables i.i.d. given $\theta$, that is $O_i \sim p(\cdot|\theta)$. Let $\gamma > 0$ . Then*

$$\Pr(\mathrm{L}(\theta, g(O^n)) > \gamma\ ) \leq (n+1)^{|O|} 2^{-n\frac{\gamma^2}{2\ln 2}}.$$

## 6 Examples of Systems Assessment

We simulate a system composed by a number of peers. In our first experiment, the behaviour of each peer is modelled as a Bernoulli distribution $\mathcal{B}(\theta)$ over the set $O = \{0, 1\}$, representing successful and unsuccessful interactions, with $\theta$ being the success probability. The parameter $\theta$ is drawn from a fixed, finite set $\Theta \subseteq (0, 1)$: we assume $\Theta$ to be a discrete set of $N$ points $0 < \gamma, 2\gamma, ..., N\gamma < 1$, for a fixed positive parameter $\gamma$. Moreover, we assume a uniform a priori distribution $\pi(\cdot)$ over $\Theta$. We consider

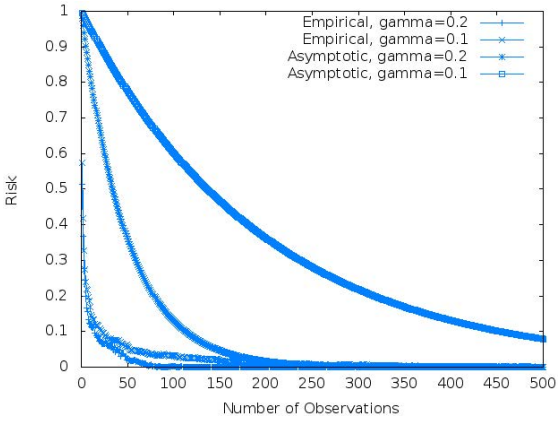



**Fig. 1.** Empirical and asymptotic Bayes risk trend

the case of reputation, and fix the loss function to be $\mathrm{L}(\theta, \theta') \triangleq \|p(\cdot|\theta) - p(\cdot|\theta')\|_1$. The purpose of this first experiment is twofold: (a) to study the rate of convergence of the risk function to their limit value depending on $\gamma$; and (b) to compare the analytical approximations $r^n \approx r^* + 2^{-nR}$ and $w^n \approx w^* + 2^{-nR}$ where $R \triangleq \min_{\theta \neq \theta'} C(p_\theta, p_{\theta'})$ with the empirical values of $r^n$ and $w^n$ obtained through the simulations. The value of $\gamma$ affects the risk value. Intuitively, for large values of $\gamma$, the incurred loss will be exactly zero in most cases. On the contrary, for small values of $\gamma$, the incurred loss will be small but nonzero in most cases.

A ML reputation function $g$ is used: we know by Theorem 2 that the convergence to both minimum bayes and worst risks is assured. Varying the value of $\gamma$, we analyse how the rate of convergence changes. We consider the following values: $\gamma = 0.2$, $\gamma = 0.1$ and $\gamma = 0.05$, for which we get the values of the rate: 0.029447, 0.0073242 and 0.0018102, respectively. As expected, for smaller values of $\gamma$ an improvement on the risks values requires a larger amount of observations, compared to larger values of $\gamma$. Figure 1 graphically shows the trend of bayes risk with respect to different values of $\gamma$. We compare the asymptotic approximation of the bayes risk with its empirical value obtained trough simulations, for different values of $\gamma$. The results for the worst risk are similar and not shown. For the computation of the empirical values, we have considered a trust and reputation system with 70 parties where possible behaviours in $\Theta$ are uniformly distributed among the parties. As we can observe by Figure 1, the bayes risk converges to its limit value according to its rate of convergence. Moreover, there is a good agreement between the asymptotic approximation and the empirical values.

In our second experiment, we analyse a system with respect to the use of different reputation functions. We consider a reputation system with 70 parties. Like in the previous example, each peer exhibits a Bernoulli behavior, with the parameter $\theta$ ranging in $\Theta$. This time, though, the prior distribution $\pi(\cdot)$ on $\Theta$ is a binomial centered on the value $\theta = 0.5$, $\mathcal{B}in(|\Theta|, 0.5)$. We compare the use of a ML and a MAP reputation function, through the study of both bayes and worst risks trend. For simplicity, we fix the value of $\gamma$ to 0.2 and we use norm-1 distance as a loss function. Figure 2 shows the trend of both bayes and worst risks for both reputation functions. As expected, MAP performs significantly better than ML when a small number of observations is available. This difference is due to the the fact that MAP actually takes advantage of the a priori knowledge represented by $\pi(\cdot)$, differently from ML. As the number of observations increases the effect of the prior washes out and both functions converge to the same value – $r^*$ or $w^*$, depending on the chosen risk model – as predicted by the theory.

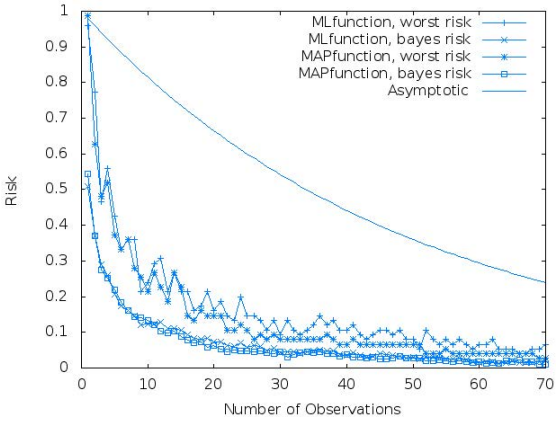



**Fig. 2.** Bayes and worst risks trend

## 7 Conclusion

We have proposed a framework to analyze probabilistic trust systems based on bayesian decison theory. We have focused on trust and reputation systems, examining the behaviour of two risk quantities: bayes and worst risks. Such quantities rely on the concept of loss function. Our results allow us to characterize the asymptotic behaviour of probabilistic trust systems. In particular, we show how to determine the limit value of both bayes and worst risks, and their exact exponential rates of convergence. Specifically, we show that decision functions based on maximum likelihood and maximum a posteriori decision functions are asymptotically optimal. We also consider the case where the raters may misbehave.

*Related Work.* Several trust and reputation models based on probabilistic concepts have been proposed in literature. Jøsang and Ismail in [11] propose the use of a reputation function based on the beta probability density function, $Beta(\alpha,\beta)$. Their approach is based on Bayesian inference and the beta distribution is used to express the a priori belief over possible parameters. The two values $\alpha$ and $\beta$ represent, respectively, the number of satisfactory and unsatisfactory interactions with a given party. The reputation score computed is the parameter $\theta$ of a Bernoulli distribution $\mathcal{B}(\theta)$. Despotovic and Aberer in [5] propose the use of reputation functions based on ML estimation: they support the appropriateness of their proposal on the basis of

simulation results. Unlike Jøsang and Ismail's proposal, the use of ML functions does not limit the choice of the probability distributions that model the parties' behaviour. In this paper we show that ML functions are asymptotically optimal.

Several studies seek to evaluate and compare trust and reputation systems. The majority of them use simulation techniques for their assessments [25,5,4]. Schlosser et al. [21] present a formal model for describing metrics in reputation systems and they evaluate the effectiveness of reputation systems on the basis of simulations. Wang and Vassileva [24] propose a Bayesian network-based trust model and evaluate their approach on the basis of simulations' results. To the best of our knowledge, only a few studies follow an approach similar to the one we propose for investigating

trust and reputation systems. Among these, Sassone et al. in [20]

propose a formal framework for the comparison of probabilistic trust models based on KL-divergence.

The KL-divergence is proposed as a measure of quality for reputation functions.

ElSalamouny et al. in [7] analyse the case of ratees exhibiting dynamic behaviours. Considerations on asymptotic behaviour are however absent. Alvim et al. [15] use the notion of gain functions in the context of information flow analysis. Such functions are the counterpart of loss functions we use. They propose different gain functions in order to quantify information leakage in several scenarios.

*Further Work.* The extension of our framework to different data models, with rating values released in different ways is one of the directions for our future research.

We shortly discuss here a possible refinement of the observation and rating mechanism that takes into account possible raters' mis-behaviour. We would like to take into account the case in which each rater is characterised by a (*unobservable*, possibly malicious) bias, which can lead him to under- or over-evaluate its interactions with the ratees.

In order to do that we have to introduce a refined data generation model, where the probability of observing a given rating value does not depend solely on the behaviour of the ratee, but also on some unobservable bias characterizing the rater's behaviour. We argue that a data-model with hidden variables is well-suited to model this kind of scenario; this naturally prompts the use of Expectation-Maximization (EM) algorithm [1] to practically perform parameter estimation in this context. The EM algorithm can be used for efficiently implementing the decision functions we have considered in the previous sections.

Another issue is how to evaluate the fitness of the model to the data actually available; and, in general, how to assess the trade-off between tractability and accuracy of the model.

## References

1. Barber, D.: Bayesian Reasoning and Machine Learning. Cambridge University Press (2012)
2. Boreale, M., Celestini, A.: Asymptotic risk analysis for trust and reputation systems, https://dl.dropbox.com/u/6235789/works/FullRisk.pdf
3. Boreale, M., Pampaloni, F., Paolini, M.: Quantitative Information Flow, with a View. In: Atluri, V., Diaz, C. (eds.) ESORICS 2011. LNCS, vol. 6879, pp. 588–606. Springer, Heidelberg (2011)
4. Boukerch, A., Xu, L., EL-Khatib, K.: Trust-based security for wireless ad hoc and sensor networks. Computer Communications 30(11-12), 2413–2427 (2007)
5. Despotovic, Z., Aberer, K.: A Probabilistic Approach to Predict Peers' Performance in P2P Networks. In: Klusch, M., Ossowski, S., Kashyap, V., Unland, R. (eds.) CIA 2004. LNCS (LNAI), vol. 3191, pp. 62–76. Springer, Heidelberg (2004)
6. Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylonen, T.: Spki certificate theory. IETF RFC (1999)
7. ElSalamouny, E., Sassone, V., Nielsen, M.: HMM-Based Trust Model. In: Degano, P., Guttman, J.D. (eds.) FAST 2009. LNCS, vol. 5983, pp. 21–35. Springer, Heidelberg (2010)
8. Heinrich, G.: Parameter estimation for text analysis. Technical report (2004)
9. Berger, J.O.: Statistical Decison Theory and Bayesian Analysis. Springer (1985)
10. Jøsang, A., Ismail, R., Boyd, C.: A survey of trust and reputation systems for online service provision. Decision Support Systems 43(2), 618–644 (2007)
11. Jøsang, A., Ismail, R.: The beta reputation system. In: Proceedings of Bled eCommerce Conference (2002)
12. Kulhavý, R.: A kullback-leibler distance approach to system identification. Annual Reviews in Control 20, 119–130 (1996)
13. Lacoste-Julien, S., Huszár, F., Ghahramani, Z.: Approximate inference for the loss-calibrated bayesian. In: Proceedings of AISTATS (2011)
14. Leang, C.C., Johnson, D.H.: On the asymptotics of m-hypothesis bayesian detection. IEEE Transactions on Information Theory 43(1), 280–282 (1997)
15. Palamidessi, C., Alvim, M., Chatzikokolakis, K., Smith, G.: Measuring information leakage using generalized gain functions. In: Proceedings of CSF 2012 (to appear, 2012)
16. Mui, L., Mohtashemi, M., Halberstadt, A.: A computational model of trust and reputation. In: Proceedings of HICSS, pp. 2431–2439 (2002)
17. Neuman, B.C., Ts'o, T.: Kerberos: an authentication service for computer networks. IEEE Communications Magazine 32(9), 33–38 (1994)
18. Nguyen, C.T., Camp, O., Loiseau, S.: A bayesian network based trust model for improving collaboration in mobile ad hoc networks. In: RIVF, pp. 144–151 (2007)

19. Robert, C.P.: The Bayesian Choice. Springer (2001)
20. Sassone, V., Nielsen, M., Krukow, K.: Towards a formal framework for computational trust. Formal Methods for Components and Objects 4, 175–184 (2007)
21. Schlosser, A., Voss, M., Brückner, L.: Comparing and evaluating metrics for reputation systems by simulation. In: Paolucci, et al. [21] (2004)
22. Cover, T.M., Thomas, J.A.: Elements of Information Theory, 2/e. Addison-Wesley (2006)
23. Teacy, W.T.L., Patel, J., Jennings, N.R., Luck, M.: Coping with inaccurate reputation sources: Experimental analysis of a probabilistic trust model. In: AAMAS (2005)
24. Wang, Y., Vassileva, J.: Trust and reputation model in peer-to-peer networks. In: P2P, pp. 150–157 (2003)
25. Xiong, L., Liu, L.: Peertrust: supporting reputation-based trust for peer-to-peer electronic communities. IEEE TKDE 16(7), 843–857 (2004)

# Being Caught between a Rock and a Hard Place in an Election – Voter Deterrence by Deletion of Candidates

Britta Dorn[1] and Dominikus Krüger[2]

[1] Universität Tübingen, Wilhelm-Schickard-Institut für Informatik, Sand 13, 72076 Tübingen, Germany
[2] Universität Ulm, Institut für Theoretische Informatik, James-Franck-Ring 5 / O27, 89081 Ulm, Germany

**Abstract.** We introduce a new problem modeling voter deterrence by deletion of candidates in elections: In an election, the removal of certain candidates might deter some of the voters from casting their votes, and the lower turnout then could cause a preferred candidate to win the election. This is a special case of the variant of the Control problem in which an external agent is allowed to delete candidates and votes in order to make his preferred candidate win, and a generalization of the variant where candidates are deleted, but no votes. We initiate a study of the computational complexity of this problem for several voting systems and obtain $\mathcal{NP}$-completeness and $\mathcal{W}[2]$-hardness with respect to the parameter *number of deleted candidates* for most of them.

## 1 Introduction

Imagine: finally, you have the chance of getting rid of your old mayor, whom you absolutely cannot stand. Luckily, in addition to the normal unscrupulous opponents, the perfect candidate is running for the vote this year. You agree with everything he says and therefore you are even looking forward to Election Day. But suddenly the word is spread that he has withdrawn his candidacy. Again, you are feeling caught between a rock and a hard place. Does it make any sense to go to the polls if you only have a choice between the lesser of two evils?

Low voter turnouts caused by scenarios such as the one in the above example may lead to modified outcomes of an election. This is reminiscent of a family of problems which have been studied extensively in the computational social choice literature recently, the Control problems [1–5] where an external agent can change the outcome of an election by adding or deleting candidates and/or voters, respectively. In particular, in the setting of constructive control by deleting candidates, the agent can prevent candidates from running for office, which causes other candidates to rise in ranking for certain voters. This may ultimately result in the external agent's preferred candidate winning the election.

In real life, this process is a little bit more complicated and control of an election can occur in a more entangled way: As in our introductory example,

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 182–193, 2013.

if some candidates do not stand for election, then certain voters will not even take part in the election because they feel that there is nothing interesting to decide or no relevant candidate to vote for. The lower turnout could have consequences for the remaining candidates: the winner of the election under normal conditions might lose points because of the lower polling after the deletion of certain candidates, and this can produce a different winner. Hence, by *deterring* the voters by means of deleting their favorite candidates, one might prevent them from casting their votes and therefore change the outcome of the election. Therefore, we call this phenomenon *voter deterrence*.

This situation can be observed in the primaries in US elections or in mayoral elections, where mayors often are elected with single-digit turnout, sometimes caused by the withdrawal of candidacy of one or several alternatives in the run-up.

As to our knowledge, this problem has not yet been considered from a computational point of view. In this paper, we want to initiate the study of the corresponding decision problem Voter Deterrence defined below. We mainly consider the case where voters are easily deterred: As soon as their most preferred candidate does not participate in the election, they refrain from the election. This is what we denote as 1-Voter Deterrence, but clearly, one can also consider $x$-Voter Deterrence, where a voter only refuses to cast his vote if his top $x$ candidates are removed. Surprisingly, it turns out that 1-Voter Deterrence is already computationally hard for several voting systems, even for Veto.

This paper is organized as follows. After introducing notation and defining the decision problem $x$-Voter Deterrence in Section 2, we investigate the complexity of this problem for the case of $x = 1$ for the voting systems Plurality (for which it turns out to be solvable in polynomial time, but with $x = 2$ it is $\mathcal{NP}$-complete), Veto, 2-approval, Borda, Maximin, Bucklin, Fallback Voting, and Copeland (for all of which the problem turns out to be $\mathcal{NP}$-complete). As a corollary, we can show that the hard problems are also $\mathcal{W}[2]$-hard with respect to the solution size, i.e., with respect to the parameter *number of deleted candidates*, which means that they remain hard even if only few candidates have to be deleted to make the preferred candidate win. This is stated in Section 4 together with a short discussion of the complexity with respect to the parameter *number of candidates*. We conclude with a discussion of open problems and further directions that might be interesting for future investigations.

## 2 Preliminaries

**Elections.** An *election* is a pair $E = (C, V)$ consisting of a *candidate* set $C = \{c_1, \ldots, c_m\}$ and a multiset $V = \{v_1, \ldots, v_n\}$ of *votes* or *voters*, each of them a linear order over $C$, i.e., a transitive, antisymmetric, and total relation over the candidates in $C$, which we denote by $\succ$. A *voting system* maps $(C, V)$ to a set $W \subseteq C$ called the *winners* of the election. All our results are given for the *unique winner case*, where $W$ consists of a single candidate.

We will consider several voting systems. Each of them is shortly described in the corresponding subsection. Most of them are *positional* scoring protocols,

which are defined by a vector of integers $\boldsymbol{\alpha} = \langle \alpha_1, \ldots, \alpha_m \rangle$, with $m$ being the number of candidates. For each voter, a candidate receives $\alpha_1$ points if he is ranked first by the voter, $\alpha_2$ if he is ranked second, etc. The *score* of a candidate is the total number of points the candidate receives. Normally, whenever candidates receive points in a voting system, the one with the highest score wins.

**Voter Deterrence, Control.** In an $x$-Voter Deterrence instance, we consider a fixed natural number $x$, and we are given an election $E = (C, V)$, a preferred candidate $p \in C$, and a natural number $k \leq |C|$, as well as a voting system. It will always be clear from the context which voting system we are using, so we will not mention it explicitly in the problem description. Let $R \subseteq C$ denote a subset of candidates, and let $V_R \subseteq V$ denote the set of voters who have ranked only candidates from $R$ among the first $x$ ranks in their vote. The task consists in determining a set $R$ of at most $k$ candidates that are removed from $C$, and who therefore prevent the set of voters $V_R$ from casting their votes, such that $p$ is a winner in the election $\widetilde{E} = (C \setminus R, V \setminus V_R)$. The set $R$ is then called a *solution* to the $x$-Voter Deterrence instance. The underlying decision problem is the following.

$x$-Voter Deterrence
**Given:** An election $E = (C, V)$, a preferred candidate $p \in C$, $k \in \mathbb{N}$ and a fixed $x \in \mathbb{N}$.
**Question:** Is there a subset of candidates $R \subseteq C$ with $|R| \leq k$, such that $p$ is the winner in the election $\widetilde{E} = (C \setminus R, V \setminus V_R)$?

$x$-Voter Deterrence is a special case of one of the many variants of the Control problem [5], where the chair is allowed to delete candidates and votes, which is defined as follows.

Constructive Control by Deleting Candidates and Votes
**Given:** An election $E = (C, V)$, a preferred candidate $p \in C$, and $k, l \in \mathbb{N}$.
**Question:** Is there a subset $C' \subseteq C$ with $|C'| \leq k$, and a subset $V' \subseteq V$ with $|V'| \leq l$, such that $p$ is a winner in the election $\widetilde{E} = (C \setminus C', V \setminus V')$?

Note that in the Voter Deterrence problem, the deleted candidates and votes are coupled, which is not necessarily the case in the above Control problem. If we set $x = m$, we obtain Constructive Control by Deleting Candidates, which is the above Constructive Control by Deleting Candidates and Votes problem with $l = 0$. The latter variant hence is a special case of $m$-Voter Deterrence, implying that the hardness results from [1] carry over to $m$-Voter Deterrence.

In this paper, we will mainly consider 1-Voter Deterrence, i.e., a voter will refuse to cast his vote if his most preferred candidate does not participate in the election. For the voting system Plurality, we also consider 2-Voter Deterrence, where a voter only refrains from voting if his two top ranked candidates are eliminated from the election.

**Parameterized Complexity.** The computational complexity of a problem is usually studied with respect to the size of the input $I$ of the problem. One can also consider the parameterized complexity [6–8] taking additionally into account the size of a so-called parameter $k$ which is a certain part of the input, such as the number of candidates in an election, or the size of the solution set. A problem is called *fixed-parameter tractable* with respect to a parameter $k$ if it can be solved in $f(k) \cdot |I|^{O(1)}$ time, where $f$ is an arbitrary computable function depending on $k$ only. The corresponding complexity class consisting of all problems that are fixed-parameter tractable with respect to a certain parameter is called $\mathcal{FPT}$.

The first two levels of (presumable) parameterized intractability are captured by the complexity classes $\mathcal{W}[1]$ and $\mathcal{W}[2]$. Proving hardness with respect to these classes can be done using an *$\mathcal{FPT}$-reduction*, which reduces a problem instance $(I, k)$ in $f(k) \cdot |I|^{O(1)}$ time to an instance $(I', k')$ such that $(I, k)$ is a yes-instance if and only if $(I', k')$ is a yes-instance, and $k'$ only depends on $k$ but not on $|I|$, see [6–8].

For all our hardness proofs, we use the $\mathcal{W}[2]$-complete Dominating Set (DS) problem for undirected graphs.

Dominating Set
**Given:** An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and a nonnegative integer $k$.
**Question:** Is there a subset $\mathcal{V}' \subseteq \mathcal{V}$ with $|\mathcal{V}'| \leq k$ such that every vertex $v \in \mathcal{V}$ is contained in $\mathcal{V}'$ or has a neighbor in $\mathcal{V}'$?

**Notation in our Proofs.** In all our reductions from Dominating Set, we will associate the vertices of the given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with candidates of the election $E = (C' \cup I, V)$ to be constructed. For that sake, we use a bijection $g\colon \mathcal{V} \to I$. By $N(v) := \{u \in \mathcal{V} \mid \{u, v\} \in \mathcal{E}\}$, we denote the set of *neighbors* or the *neighborhood* of a vertex $v \in \mathcal{V}$. Analogously, we define the *neighborhood of a candidate* $c_i \in I$ as $N(c_i) = g(N(v_i))$ for $c_i = g(v_i)$, i.e., the set of neighbors of a candidate $c_i \in I$ corresponding to the vertex $v_i \in \mathcal{V}$ is the set of candidates corresponding to the neighborhood of $v_i$ in $\mathcal{G}$. By $\overline{N(v_i)} \subset I$ we denote the set of non-neighbors of $v_i$, analogously for neighborhoods of candidates.

In our reductions, we usually need one dummy candidate for every $c_i \in C$, these will be denoted by $\hat{c}_i$. All other dummy candidates appearing are marked with a hat as well, usually they are called $\hat{d}$ or similarly. When building the votes in our reductions, we write '$k \parallel a_1 \succ \cdots \succ a_l$' which means that we construct the given vote $a_1 \succ \cdots \succ a_l$ exactly $k$ times.

In our preference lists, we sometimes specify a whole subset of candidates, e.g., $c \succ D$ for a candidate $c \in C$ and a subset of candidates $D \subset C$. This notation means $c \succ d_1 \succ \cdots \succ d_l$ for an arbitrary but fixed order of $D = \{d_1, \ldots, d_l\}$. If we use a set $\vec{D}$ in a preference list, we mean one specific, fixed (but arbitrary, and unimportant) order of the elements in $D$, which is reversed if we write $\overleftarrow{D}$. Hence, if $c \succ \vec{D}$ stands for $c \succ d_1 \succ \cdots \succ d_l$, then $c \succ \overleftarrow{D}$ means $c \succ d_l \succ \cdots \succ d_1$. Finally, whenever we use the notation $D_{\text{rest}}$ for a subset of candidates in a vote, we mean the set consisting of those candidates in $D$ that have not been positioned explicitly in this vote.

## 3 Complexity-Theoretic Analysis

In this section, we will give several hardness proofs for Voter Deterrence for different voting systems. All our results rely on reductions from the $\mathcal{NP}$-complete problem Dominating Set. We only prove $\mathcal{NP}$-hardness for the different voting systems, but since membership in $\mathcal{NP}$ is always trivially given, $\mathcal{NP}$-completeness follows immediately. For all these reductions we assume that every vertex of the input instance has at least two neighbors, which is achievable by a simple polynomial time preprocessing.

We will give the first of the following reduction proofs in detail. For the remaining reductions, we specify the constructed instances together with further helpful remarks. The proofs that these are indeed equivalent to the Dominating Set-instances are straightforward and can be found in the appendix. In each of them, one obtains a solution to the $x$-Voter Deterrence-instance by deleting exactly those candidates that correspond to the vertices belonging to a solution of the corresponding Dominating Set-instance, and vice versa.

### 3.1 Plurality

The *Plurality* protocol is the positional scoring protocol with $\boldsymbol{\alpha} = \langle 1, 0, \ldots, 0 \rangle$ [9].

It is easy to see that 1-Voter Deterrence can be solved efficiently for Plurality. One can simply order the candidates according to their score and if there are more than $k$ candidates ahead of $p$, this instance is a no-instance. Otherwise $p$ will win after deletion of the candidates that were ranked higher than him, because all the votes which they got a point from are removed. Therefore the following theorem holds.

**Theorem 1.** 1-Voter Deterrence *is in* $\mathcal{P}$ *for* Plurality.

For 2-Voter Deterrence, it is not so easy to see which candidates should be deleted. In fact, the problem is $\mathcal{NP}$-complete.

**Theorem 2.** 2-Voter Deterrence *is* $\mathcal{NP}$*-complete for* Plurality.

*Proof.* We prove Theorem 2 with an $\mathcal{FPT}$-reduction from Dominating Set. Let $\langle \mathcal{G} = (\mathcal{V}, \mathcal{E}), k \rangle$ be an instance of DS.

*Candidates:* For every vertex $v_i \in \mathcal{V}$ we need one candidate $c_i$ and one dummy candidate $\hat{c}_i$, as well as the preferred candidate $p$ and his dummy candidate $\hat{p}$, so $C = I \cup D \cup \{p\}$ with $I = \{c_1, \ldots, c_n\}$ and $D = \{\hat{c}_1, \ldots, \hat{c}_n, \hat{p}\}$. For ease of presentation we denote $I \cup \{p\}$ by $I^*$.

*Votes:* The votes are built as follows.

$$n \parallel p \succ \hat{p} \succ C_{\text{rest}}, \tag{1}$$

$$\forall c_i \in I : \quad |N(c_i)| \parallel c_i \succ \hat{c}_i \succ C_{\text{rest}}, \tag{2}$$

$$\forall c_j \in \overline{N(c_i)} \cup \{p\} : \quad 1 \parallel c_i \succ c_j \succ C_{\text{rest}}. \tag{3}$$

Note that $n$ votes are built for every candidate $c_i$. Therefore each candidate in $I^*$ has the score $n$. The score of a candidate can only be decreased if the corresponding candidate himself is deleted. Note also that the score of every dummy candidate cannot exceed $n-1$.
We will now show that one can make $p$ win the election by deleting up to $k$ candidates if and only if the DS-instance has a solution of size at most $k$.

*"⇒":* Let $S$ be a given solution to the DS-instance. Then $R = g(S)$ is a solution to the corresponding 2-Voter Deterrence-instance. Since $S$ is a dominating set, every candidate in $I$ will be at least once in the neighborhood of a candidate $c_i \in R$ or be a candidate in $R$ himself. Therefore $p$ is the only candidate who gains an additional point from every deleted candidate $c_x \in R$ from the vote built by (3) and will therefore be the unique winner.

*"⇐":* Let $R$ be a given solution to a 2-Voter Deterrence-instance. Note that every candidate in $I^*$ has the original score $n$. These scores can be increased if the corresponding candidate himself is not deleted. If $p$ wins, then with the same argument as before, we see that every candidate $c_x \in I$ either must be a member of the set $R$, or must not appear as $c_j$ on the second position of the votes built by (3) for at least one candidate of $R$, hence must be in the neighborhood of $R$. Therefore $S = g^{-1}(R)$ is a solution to the equivalent DS-instance. □

### 3.2 Veto

The positional scoring protocol with $\boldsymbol{\alpha} = \langle 1, 1, \ldots, 1, 0\rangle$ is called *Veto* [9].

**Theorem 3.** 1-Voter Deterrence *is* $\mathcal{NP}$*-complete for* Veto.

We show Theorem 3 with an $\mathcal{FPT}$-reduction from Dominating Set. Let $\langle \mathcal{G} = (\mathcal{V}, \mathcal{E}), k\rangle$ be an instance of DS.

*Candidates:* For every vertex $v_i \in \mathcal{V}$ we need one candidate $c_i$, as well as the preferred candidate $p$ and $k+1$ dummy candidates, so $C = I \cup D \cup \{p\}$ with $I = \{c_1, \ldots, c_n\}$ and $D = \{\hat{d}_1, \ldots, \hat{d}_{k+1}\}$. For ease of presentation we denote $I \cup \{p\}$ by $I^*$.

*Votes:* The votes are built as follows.

$$
\begin{aligned}
&\forall c_i \in I: \\
&\quad \forall c_j \in I^* \setminus (N(c_i) \cup \{c_i\}): \\
&\qquad 1 \parallel c_i \succ C_{\text{rest}} \succ D \succ c_j, && (1)\\
&\quad \forall c_j \in N(c_i) \cup \{c_i\}: \\
&\qquad 1 \parallel p \succ I_{\text{rest}} \succ D \succ c_j, && (2)\\
&\quad \forall \hat{d}_j \in D: \\
&\qquad 2 \parallel p \succ I \succ D_{\text{rest}} \succ \hat{d}_j. && (3)
\end{aligned}
$$

Note that every vote built by (2) and (3) can only be removed by deleting the candidate $p$, who should win the election. Therefore these votes will not be

removed. Note also that for each set of votes constructed for a candidate $c_i \in I$, every candidate in $C \setminus D$ takes the last position in one of theses votes, hence the score of every such candidate is the same. In contrast, the dummy candidates cannot win the election at all, due to the fact that they are on the last position of the constructed votes twice as often as the other candidates.

### 3.3 2-approval

The *2-approval* protocol is the positional scoring protocol with the scoring vector $\boldsymbol{\alpha} = \langle 1, 1, 0, \ldots, 0 \rangle$ [9].

**Theorem 4.** 1-Voter Deterrence *is* $\mathcal{NP}$*-complete for* 2-approval.

We show Theorem 4 by an $\mathcal{FPT}$-reduction from Dominating Set. Let $\langle \mathcal{G} = (\mathcal{V}, \mathcal{E}), k \rangle$ be an instance of DS.

*Candidates:* For every vertex $v_i \in \mathcal{V}$, we create one candidate $c_i$ and one additional dummy candidate $\hat{c}_i$, finally we need the preferred candidate $p$. So with $I = \{c_1, \ldots, c_n\}$ and $D = \{\hat{c}_1, \ldots, \hat{c}_n\}$, the candidates are $C = I \cup D \cup \{p\}$.

*Votes:* The votes are built as follows.

$$\forall c_i \in I:$$
$$\forall c_j \in N(c_i):$$
$$1 \parallel c_i \succ c_j \succ \hat{c}_j \succ C_{\text{rest}} \succ p, \tag{1}$$
$$\forall c_j \in I \setminus (N(c_i) \cup \{c_i\}):$$
$$1 \parallel \hat{c}_i \succ c_j \succ \hat{c}_j \succ C_{\text{rest}} \succ p, \tag{2}$$
$$2 \parallel \hat{c}_i \succ p \succ C_{\text{rest}}, \tag{3}$$
$$n - |N(c_i)| \parallel c_i \succ \hat{c}_i \succ C_{\text{rest}} \succ p. \tag{4}$$

Without any candidate deleted, all $c_i \in I$ and $p$ have the same score of $2n$, while the dummy candidates $\hat{c}_j \in D$ have a score less than $2n$. Note that one decreases $p$'s score by deleting a dummy candidate, because a deletion of this kind results in losing a vote built in (3). Therefore one has to delete candidates in $I$ to help $p$ in winning.

### 3.4 Borda

The positional scoring protocol with $\boldsymbol{\alpha} = \langle m-1, m-2, \ldots, 0 \rangle$ is called *Borda* [9].

**Theorem 5.** 1-Voter Deterrence *is* $\mathcal{NP}$*-complete for* Borda.

We show Theorem 5 by an $\mathcal{FPT}$-reduction from Dominating Set. Let $\langle \mathcal{G} = (\mathcal{V}, \mathcal{E}), k \rangle$ be an instance of DS.

*Candidates:* For every vertex $v_i \in \mathcal{V}$ we create one candidate $c_i$ and one dummy candidate $\hat{c}_i$, finally we need the preferred candidate $p$. So the candidates are $C = I \cup D \cup \{p\}$ with $I = \{c_1, \ldots, c_n\}$ and $D = \{\hat{c}_1, \ldots, \hat{c}_n\}$. For ease of presentation, we denote $I \cup \{p\}$ by $I^*$.

*Votes:* The votes are built as follows.

$$\forall c_i \in I:$$
$$\forall c_j \in N(c_i):$$
$$1 \parallel c_i \succ \vec{I^*}_{\text{rest}} \succ c_j \succ \hat{c}_j \succ \vec{D}_{\text{rest}} \succ \hat{c}_i, \tag{1}$$
$$1 \parallel c_i \succ c_j \succ \hat{c}_j \succ \overleftarrow{I^*}_{\text{rest}} \succ \overleftarrow{D}_{\text{rest}} \succ \hat{c}_i, \tag{2}$$
$$1 \parallel \hat{c}_i \succ \hat{c}_j \succ c_j \succ \overleftarrow{I^*}_{\text{rest}} \succ c_i \succ \overleftarrow{D}_{\text{rest}}, \tag{3}$$
$$1 \parallel \hat{c}_i \succ \vec{I^*}_{\text{rest}} \succ \hat{c}_j \succ c_j \succ c_i \succ \vec{D}_{\text{rest}}. \tag{4}$$

Recall that $\vec{A}$ denotes one specific order of the elements within the set $A$ which is reversed in $\overleftarrow{A}$. Keeping this in mind, it is easy to see that every candidate in $I^*$ has the same score within one gadget constructed by the four votes built by (1) to (4) for one $c_j$, while the dummy candidates all have a lower score. Note that the deletion of any candidate will decrease the score of every other candidate. Therefore the scores of the candidates in $I$ have to be decreased more than the one of $p$, whereas the scores of the candidates in $I^*$ can never be brought below the score of any candidate in $D$.

### 3.5 Maximin

This voting protocol is also known as *Simpson*. For any two distinct candidates $i$ and $j$, let $N(i,j)$ be the number of voters that prefer $i$ to $j$. The *maximin score* of $i$ is $\min_{j \neq i} N(i,j)$ [9].

**Theorem 6.** 1-Voter Deterrence *is* $\mathcal{NP}$*-complete for* Maximin.

We show Theorem 6 by an $\mathcal{FPT}$-reduction from Dominating Set. Let $\langle \mathcal{G} = (\mathcal{V}, \mathcal{E}), k \rangle$ be an instance of DS.

*Candidates:* For every vertex $v_i \in \mathcal{V}$ we create one candidate $c_i$ and one dummy candidate $\hat{c}_i$, finally we need the preferred candidate $p$. So the candidates are $C = I \cup D \cup \{p\}$ with $I = \{c_1, \ldots, c_n\}$ and $D = \{\hat{c}_1, \ldots, \hat{c}_n\}$.

*Votes:* The votes are built as follows.

$$\forall c_i \in I:$$
$$1 \parallel c_i \succ \vec{I}_{\text{rest}} \succ \vec{N}(c_i) \succ p \succ \vec{D}_{\text{rest}} \succ \hat{c}_i, \tag{1}$$
$$1 \parallel c_i \succ \overleftarrow{N}(c_i) \succ p \succ \overleftarrow{I}_{\text{rest}} \succ \overleftarrow{D}_{\text{rest}} \succ \hat{c}_i, \tag{2}$$
$$1 \parallel \hat{c}_i \succ \vec{I}_{\text{rest}} \succ p \succ \vec{N}(c_i) \succ \vec{D}_{\text{rest}} \succ c_i, \tag{3}$$
$$1 \parallel \hat{c}_i \succ p \succ \overleftarrow{N}(c_i) \succ \overleftarrow{I}_{\text{rest}} \succ \overleftarrow{D}_{\text{rest}} \succ c_i. \tag{4}$$

Recall that $\vec{A}$ denotes one specific order of the elements within set $A$ which is reversed in $\overleftarrow{A}$. With this in mind, it is easy to see that every candidate in $I$ has the same score as $p$, namely $2n$. The dummy candidates are not able to win the election as long as at least one of the candidates in $I$ or $p$ is remaining.

### 3.6 Bucklin and Fallback Voting

A candidate $c$'s Bucklin score is the smallest number $k$ such that more than half of the votes rank $c$ among the top $k$ candidates. The winner is the candidate that has the smallest Bucklin score [10].

**Theorem 7.** 1-Voter Deterrence *is* $\mathcal{NP}$*-complete for* Bucklin.

Note that Bucklin is a special case of *Fallback Voting*, where each voter approves of each candidate, see [11]. We therefore also obtain

**Corollary 1.** 1-Voter Deterrence *is* $\mathcal{NP}$*-complete for* Fallback Voting.

We show Theorem 7 by an $\mathcal{FPT}$-reduction from Dominating Set. Let $\langle \mathcal{G} = (\mathcal{V}, \mathcal{E}), k \rangle$ be an instance of DS.

*Candidates:* For every vertex $v_i \in \mathcal{V}$ we create one candidate $c_i$ and one dummy candidate $\hat{c}_i$. Additionally, we need the preferred candidate $p$ and several dummy candidates. We need $n(n+k)$ *filling* dummies $\hat{f}$, $k(2n+k-1)$ *security* dummies $\hat{s}$, and finally $k-1$ *leading* dummies $\hat{l}$. So the candidates are $C = I \cup D \cup S \cup F \cup L \cup \{p\}$ with $I = \{c_1, \ldots, c_n\}$, $D = \{\hat{c}_1, \ldots, \hat{c}_n\}$, $S = \{\hat{s}_1, \ldots, \hat{s}_{k(2n+k-1)}\}$, $F = \{\hat{f}_1, \ldots, \hat{f}_{n(n+k)}\}$, and $L = \{\hat{l}_1, \ldots, \hat{l}_{k-1}\}$. For ease of presentation, we denote $I \cup \{p\}$ by $I^*$.

*Votes:* The votes are built as follows.

$\forall c_i \in I$ :

$$1 \parallel c_i \succ N(c_i) \succ \{\hat{f}_{(i-1)(n+1)+1}, \ldots, \hat{f}_{i(n+1)-|N(c_i)|-1}\} \succ \{\hat{s}_{(2i-2)(k+1)+1}, \ldots, \hat{s}_{2(i-1)(k+1)}\} \succ C_{\text{rest}} \succ p, \tag{1}$$

$$1 \parallel \hat{c}_i \succ \overline{N(c_i)} \succ \{\hat{f}_{i(n+1)-|N(c_i)|}, \ldots, \hat{f}_{(i)(n+1)}\} \succ p \succ \{\hat{s}_{(2i-1)(k+1)+1}, \ldots, \hat{s}_{2i(k+1)}\} \succ C_{\text{rest}}, \tag{2}$$

$\forall r \in \{1, \ldots, k-1\}$ : one vote of the form

$$1 \parallel \hat{l}_r \succ \{\hat{f}_{n(n+1)+(r-1)n+1}, \ldots, \hat{f}_{n(n+1)+in}\} \succ \{\hat{s}_{2n(k+1)+(r-1)(k+1)+1}, \ldots, \hat{s}_{2n(k+1)+r(k+1)}\} \succ C_{\text{rest}} \succ p. \tag{3}$$

Note that every candidate in $I^*$ occurs within the first $n+2$ positions in the votes built by (1) and (2) for every candidate $c_i \in I$ exactly once. Therefore $p$ is not the unique winner without modification. Note also that deleting some of the dummy candidates is not helping $p$, as they all appear just once within the first $n+2$ positions. Because of the security dummies, no candidate in $I^*$ can move up to one of the first $n+2$ positions, if he has not been there before. After the deletion of $k$ candidates, up to $k$ votes can be removed—note that every removed vote has to be built by (1) or (3) if $p$ wins the election with this deletion.

### 3.7 Copeland

For any two distinct candidates $i$ and $j$, let again $N(i,j)$ be the number of voters that prefer $i$ to $j$, and let $C(i,j) = +1$ if $N(i,j) > N(j,i)$, $C(i,j) = 0$ if

$N(i,j) = N(j,i)$, and $C(i,j) = -1$ if $N(i,j) < N(j,i)$. The *Copeland score* of candidate $i$ is $\sum_{j \neq i} C(i,j)$ [9].

**Theorem 8.** 1-Voter Deterrence *is* $\mathcal{NP}$*-complete for the voting system* Copeland.

We show Theorem 8 by an $\mathcal{FPT}$-reduction from Dominating Set. Let $\langle \mathcal{G} = (\mathcal{V}, \mathcal{E}), k \rangle$ be an instance of DS.

*Candidates:* For every vertex $v_i \in \mathcal{V}$ we create one candidate $c_i$ and one dummy candidate $\hat{c}_i$. Additionally we need the preferred candidate $p$, one *thievish* candidate $\hat{t}$ and furthermore $n$ *filling* dummy candidates. So the candidates are $C = I \cup D \cup F \cup \{\hat{t}, p\}$ with $I = \{c_1, \ldots, c_n\}$, $D = \{\hat{c}_1, \ldots, \hat{c}_n\}$, and $F = \{\hat{f}_1, \ldots, \hat{f}_n\}$.

*Votes:* The votes are built as follows.

$\forall c_i \in I$ :

$$1 \parallel c_i \succ \vec{N}(c_i) \succ \hat{t} \succ \vec{I}_{\text{rest}} \succ p \succ \vec{F} \succ \vec{D}_{\text{rest}} \succ \hat{c}_i, \tag{1}$$

$$1 \parallel c_i \succ p \succ \overleftarrow{I}_{\text{rest}} \succ \overleftarrow{N}(c_i) \succ \overleftarrow{F} \succ \hat{t} \succ \overleftarrow{D}_{\text{rest}} \succ \hat{c}_i, \tag{2}$$

$$1 \parallel \hat{c}_i \succ \hat{t} \succ \vec{N}(c_i) \succ \vec{I}_{\text{rest}} \succ p \succ \vec{F} \succ c_i \succ \vec{D}_{\text{rest}}, \tag{3}$$

$$1 \parallel \hat{c}_i \succ p \succ \overleftarrow{I}_{\text{rest}} \succ \overleftarrow{F} \succ \hat{t} \succ \overleftarrow{N}(c_i) \succ c_i \succ \overleftarrow{D}_{\text{rest}}. \tag{4}$$

These $n$ gadgets (consisting of the above 4 votes) cause that the candidates have different scores. Note that the candidates of each set are always tying with the other candidates in their set, since every gadget has two votes with one specific order of the members and another two of the reversed order. Since candidates in $D$ are losing every pairwise election against all other candidates, they have a score of $-(2n+2)$. The candidates in $F$ are just winning against the candidates in $D$ and are tied against $\hat{t}$ and therefore have a score of $-1$. Since the candidates in $I$ and $p$ are on a par with $\hat{t}$, this gives them a score of $2n$ and $\hat{t}$ a score of $n$. Note that if there exists a deletion of $k$ candidates which makes $p$ win the election, there also exists a deletion of up to $k$ candidates in $I$ doing so. The main idea here is that the thievish candidate can steal exactly one point from every candidate in $I$ by winning the pairwise election between them due to the deleted candidate and thereby removed votes. Since $\hat{t}$ starts with a score of $n$, this will only bring him to a score of $2n - k$ with $k$ deleted candidates. Therefore he cannot get a higher score than $p$ initially had.

## 4 Parameterized Complexity-Theoretic Analysis

In this section, we shortly take a closer look at the parameterized complexity of Voter Deterrence for the previously considered voting systems.

Since all the $\mathcal{NP}$-hardness proofs of the previous section are based on $\mathcal{FPT}$-reductions from Dominating Set, we immediately obtain

**Corollary 2.** 1-Voter Deterrence *is* $\mathcal{W}[2]$*-hard for* Copeland*,* Veto*,* Borda*,* 2-approval*,* Maximin*,* Bucklin*, and* Fallback Voting*, and* 2-Voter Deterrence *is* $\mathcal{W}[2]$*-hard for* Plurality*, all with respect to the parameter* number of deleted candidates.

In contrast, considering a different parameter, one easily obtains the following tractability result.

**Theorem 9.** *The problem* $x$-Voter Deterrence *is in* $\mathcal{FPT}$ *with respect to the parameter* number of candidates *for all voting systems having a polynomial time winner determination.*

It is easy to see that Theorem 9 holds: An algorithm trying out every combination of candidates to delete has an $\mathcal{FPT}$-running time $\mathcal{O}(m^k \cdot n \cdot m \cdot T_{\text{poly}})$, where $m$ is the number of candidates, $n$ the number of votes, $k \leq m$ is the number of allowed deletions, and $T_{\text{poly}}$ is the polynomial running time of the winner determination in the specific voting system.

## 5 Conclusion

We have initiated the study of a voting problem that takes into account correlations that appear in real life, but which has not been considered from a computational point of view so far. We obtained $\mathcal{NP}$-completeness and $\mathcal{W}[2]$-hardness for most voting systems we considered. However, this is just the beginning, and it would be interesting to obtain results for other voting systems such as $k$-approval or scoring rules in general. Also, we have concentrated on the case of 1-Voter Deterrence and so far have investigated 2-Voter Deterrence for Plurality only.

One could also look at the *destructive* variant of the problem in which an external agent wants to prevent a hated candidate from winning the election, see, e.g., [2] for a discussion for the Control problem.

We have also investigated our problem from the point of view of parameterized complexity. It would be interesting to consider different parameters, such as the number of votes, or even a combination of several parameters (see [12]), to determine the complexity of the problem in a more fine-grained way. This approach seems especially worthwile because Voter Deterrence, like other ways of manipulating the outcome of an election, is a problem for which NP-hardness results promise some kind of resistance against this dishonest behavior. Parameterized complexity helps to keep up this resistance or to show its failure for cases where certain parts of the input are small, and thus it provides a more robust notion of hardness. See, e.g., [11,13–16], and the recent survey [17].

However, one should keep in mind that combinatorial hardness is a worst case concept, so it would clearly be interesting to consider the average case complexity of the problem or to investigate the structure of naturally appearing instances. E.g., when the voters have *single peaked preferences*, many problems become easy [18]. Research in this direction is becoming more and more popular in the computational social choice community, see for example [18–20].

**Acknowledgment.** We thank Oliver Gableske for the fruitful discussion which initiated our study of Voter Deterrence, and the referees of COMSOC 2012 and SOFSEM 2013 whose constructive feedback helped to improve this work.

## References


1. Bartholdi, J., Tovey, C., Trick, M., et al.: How Hard is it to Control an Election? Math. Comput. Mod. 16(8-9), 27–40 (1992)
2. Hemaspaandra, E., Hemaspaandra, L.A., Rothe, J.: Anyone but him: The complexity of precluding an alternative. Artif. Intell. 171(5-6), 255–285 (2007)
3. Faliszewski, P., Hemaspaandra, E., Hemaspaandra, L., Rothe, J.: Llull and Copeland voting computationally resist bribery and constructive control. J. Artif. Intell. Res. 35(1), 275–341 (2009)
4. Erdélyi, G., Rothe, J.: Control complexity in fallback voting. In: Proc. of 16th CATS, pp. 39–48 (2010)
5. Faliszewski, P., Hemaspaandra, E., Hemaspaandra, L.: Multimode control attacks on elections. J. Artif. Intell. Res. 40, 305–351 (2011)
6. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science. Springer, New York (1999)
7. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science. An EATCS Series. Springer, New York (2006)
8. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press (2006)
9. Conitzer, V., Lang, J., Sandholm, T.: How many candidates are needed to make elections hard to manipulate? In: Proc. of 9th TARK, pp. 201–214. ACM (2003)
10. Xia, L., Zuckerman, M., Procaccia, A.D., Conitzer, V., Rosenschein, J.S.: Complexity of unweighted coalitional manipulation under some common voting rules. In: Proc. 21st IJCAI, pp. 348–353 (2009)
11. Erdélyi, G., Fellows, M.: Parameterized control complexity in bucklin voting and in fallback voting. In: Proc. of 3rd COMSOC, pp. 163–174 (2010)
12. Niedermeier, R.: Reflections on multivariate algorithmics and problem parameterization. In: Proc. of 27th STACS, pp. 17–32 (2010)
13. Betzler, N., Uhlmann, J.: Parameterized complexity of candidate control in elections and related digraph problems. Theor. Comput. Sci. 410(52), 5425–5442 (2009)
14. Christian, R., Fellows, M., Rosamond, F., Slinko, A.: On complexity of lobbying in multiple referenda. Rev. Econ. Design 11(3), 217–224 (2007)
15. Dorn, B., Schlotter, I.: Multivariate complexity analysis of swap bribery. Algorithmica 64(1), 126–151 (2012)
16. Bredereck, R., Chen, J., Hartung, S., Niedermeier, R., Suchý, O., Kratsch, S.: A Multivariate Complexity Analysis of Lobbying in Multiple Referenda. In: Proc. of 26th AAAI, pp. 1292–1298
17. Betzler, N., Bredereck, R., Chen, J., Niedermeier, R.: Studies in Computational Aspects of Voting–a Parameterized Complexity Perspective. In: Bodlaender, H., Downey, R., Fomin, F.V., Marx, D. (eds.) Fellows Festschrift 2012. LNCS, vol. 7370, pp. 318–363. Springer, Heidelberg (2012)
18. Faliszewski, P., Hemaspaandra, E., Hemaspaandra, L.A., Rothe, J.: The shield that never was: Societies with single-peaked preferences are more open to manipulation and control. Inf. Comput. 209(2), 89–107 (2011)
19. Friedgut, E., Kalai, G., Nisan, N.: Elections can be manipulated often. In: Proc. of 49th FOCS, pp. 243–249 (2008)
20. Faliszewski, P., Procaccia, A.D.: AI's War on Manipulation: Are We Winning? AI Mag. 31(4), 53–64 (2010)

# Collective Additive Tree Spanners of Bounded Tree-Breadth Graphs with Generalizations and Consequences

Feodor F. Dragan and Muad Abu-Ata

Department of Computer Science, Kent State University, Kent, OH 44242, USA
{dragan,mabuata}@cs.kent.edu

**Abstract.** In this paper, we study collective additive tree spanners for families of graphs enjoying special Robertson-Seymour's tree-decompositions, and demonstrate interesting consequences of obtained results. It is known that if a graph $G$ has a multiplicative tree $t$-spanner, then $G$ admits a Robertson-Seymour's tree-decomposition with bags of radius at most $\lceil t/2 \rceil$ in $G$. We use this to demonstrate that there is a polynomial time algorithm that, given an $n$-vertex graph $G$ admitting a multiplicative tree $t$-spanner, constructs a system of at most $\log_2 n$ collective additive tree $O(t \log n)$-spanners of $G$. That is, with a slight increase in the number of trees and in the stretch, one can "turn" a multiplicative tree spanner into a small set of collective additive tree spanners. We extend this result by showing that, for every fixed $k$, there is a polynomial time algorithm that, given an $n$-vertex graph $G$ admitting a multiplicative $t$-spanner with tree-width $k-1$, constructs a system of at most $k(1+\log_2 n)$ collective additive tree $O(t \log n)$-spanners of $G$.

## 1 Introduction

One of the basic questions in the design of routing schemes for communication networks is to construct a spanning network (a so-called spanner) which has two (often conflicting) properties: it should have simple structure and nicely approximate distances in the network. This problem fits in a larger framework of combinatorial and algorithmic problems that are concerned with distances in a finite metric space induced by a graph. An arbitrary metric space (in particular a finite metric defined by a graph) might not have enough structure to exploit algorithmically. A powerful technique that has been successfully used recently in this context is to embed the given metric space in a simpler metric space such that the distances are approximately preserved in the embedding. New and improved algorithms have resulted from this idea for several important problems (see, e.g., [1,2,4,14,18]). There are several ways to measure the quality of this approximation, two of them leading to the notion of a spanner. For $t \geq 1$, a spanning subgraph $H$ of $G = (V, E)$ is called a *(multiplicative) $t$-spanner* of $G$ if $d_H(u, v) \leq t \cdot d_G(u, v)$ for all $u, v \in V$ [5,23,24]. If $r \geq 0$ and $d_H(u, v) \leq d_G(u, v) + r$, for all $u, v \in V$, then $H$ is called an *additive $r$-spanner* of $G$ [19,26].

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 194–206, 2013.

The parameter $t$ is called the *stretch* (or *stretch factor*) of $H$, while the parameter $r$ is called the *surplus* of $H$. In what follows, we will often omit the word "multiplicative" when we refer to multiplicative spanners.

Tree metrics are a very natural class of simple metric spaces since many algorithmic problems become tractable on them. A *(multiplicative) tree $t$-spanner* of a graph $G$ is a spanning tree with a stretch $t$ [3], and an *additive tree $r$-spanner* of $G$ is a spanning tree with a surplus $r$ [26]. If we approximate the graph by a tree spanner, we can solve the problem on the tree and the solution interpret on the original graph. The TREE $t$-SPANNER problem asks, given a graph $G$ and a positive number $t$, whether $G$ admits a tree $t$-spanner. Note that the problem of finding a tree $t$-spanner of $G$ minimizing $t$ is known in literature also as the Minimum Max-Stretch spanning Tree problem (see, e.g., [16] and literature cited therein). Unfortunately, not many graph families admit good tree spanners. This motivates the study of sparse spanners, i.e., spanners with a small amount of edges. There are many applications of spanners in various areas; especially, in distributed systems and communication networks. In [24], close relationships were established between the quality of spanners (in terms of stretch factor and the number of spanner edges), and the time and communication complexities of any synchronizer for the network based on this spanner. Another example is the usage of tree $t$-spanners in the analysis of arrow distributed queuing protocols [22]. Sparse spanners are very useful in message routing in communication networks; in order to maintain succinct routing tables, efficient routing schemes can use only the edges of a sparse spanner [25]. The SPARSEST $t$-SPANNER problem asks, for a given graph $G$ and a number $t$, to find a $t$-spanner of $G$ with the smallest number of edges. We refer to the survey paper of Peleg [21] for an overview on spanners.

Inspired by ideas from works of Bartal [1], Fakcharoenphol et al. [17], and to extend those ideas to designing compact and efficient routing and distance labeling schemes in networks, in [13], a new notion of *collective tree spanners* was introduced. This notion is slightly *weaker* than the one of a tree spanner and slightly *stronger* than the notion of a sparse spanner. We say that a graph $G = (V, E)$ *admits a system of $\mu$ collective additive tree $r$-spanners* if there is a system $\mathcal{T}(G)$ of at most $\mu$ spanning trees of $G$ such that for any two vertices $x, y$ of $G$ a spanning tree $T \in \mathcal{T}(G)$ exists such that $d_T(x, y) \leq d_G(x, y) + r$ (a multiplicative variant of this notion can be defined analogously). Clearly, if $G$ admits a system of $\mu$ collective additive tree $r$-spanners, then $G$ admits an additive $r$-spanner with at most $\mu \times (n - 1)$ edges (take the union of all those trees), and if $\mu = 1$ then $G$ admits an additive tree $r$-spanner. Recently, in [10], *spanners of bounded tree-width* were introduced, motivated by the fact that many algorithmic problems are tractable on graphs of bounded tree-width, and a spanner $H$ of $G$ with small tree-width can be used to obtain an approximate solution to a problem on $G$. In particular, efficient and compact distance and routing labeling schemes are available for bounded tree-width graphs (see, e.g., [12,18] and papers cited therein), and they can be used to compute approximate distances and route along paths that are close to shortest in $G$.

The $k$-TREE-WIDTH $t$-SPANNER problem asks, for a given graph $G$, an integers $k$ and a positive number $t \geq 1$, whether $G$ admits a $t$-spanner of tree-width at most $k$. Every connected graph with $n$ vertices and at most $n-1+m$ edges is of tree-width at most $m+1$ and hence this problem is a generalization of the TREE $t$-SPANNER and the SPARSEST $t$-SPANNER problems. Furthermore, spanners of bounded tree-width have much more structure to exploit algorithmically than sparse spanners.

**Our Results and Their Place in the Context of the Previous Results.** This paper was inspired by few recent results from [7,11,15,16]. Elkin and Peleg in [15], among other results, described a polynomial time algorithm that, given an $n$-vertex graph $G$ admitting a tree $t$-spanner, constructs a $t$-spanner of $G$ with $O(n \log n)$ edges. Emek and Peleg in [16] presented the first $O(\log n)$-approximation algorithm for the minimum value of $t$ for the TREE $t$-SPANNER problem. They described a polynomial time algorithm that, given an $n$-vertex graph $G$ admitting a tree $t$-spanner, constructs a tree $O(t \log n)$-spanner of $G$. Later, a simpler and faster $O(\log n)$-approximation algorithm for the problem was given by Dragan and Köhler [11]. Their result uses a new necessary condition for a graph to have a tree $t$-spanner: if a graph $G$ has a tree $t$-spanner, then $G$ admits a Robertson-Seymour's tree-decomposition with bags of radius at most $\lceil t/2 \rceil$ in $G$.

To describe the results of [7] and to elaborate more on the Dragan-Köhler's approach, we need to recall definitions of a few graph parameters. They all are based on the notion of tree-decomposition introduced by Robertson and Seymour in their work on graph minors [27].

A *tree-decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i | i \in I\}, T = (I, F))$ where $\{X_i | i \in I\}$ is a collection of subsets of $V$, called *bags*, and $T$ is a tree. The nodes of $T$ are the bags $\{X_i | i \in I\}$ satisfying the following three conditions: 1) $\bigcup_{i \in I} X_i = V$; 2) for each edge $uv \in E$, there is a bag $X_i$ such that $u, v \in X_i$; 3) for all $i, j, k \in I$, if $j$ is on the path from $i$ to $k$ in $T$, then $X_i \bigcap X_k \subseteq X_j$. Equivalently, this condition could be stated as follows: for all vertices $v \in V$, the set of bags $\{i \in I | v \in X_i\}$ induces a connected subtree $T_v$ of $T$.

For simplicity we denote a tree-decomposition $(\{X_i | i \in I\}, T = (I, F))$ of a graph $G$ by $T(G)$.

Tree-decompositions were used to define several graph parameters to measure how close a given graph is to some known graph class (e.g., to trees or to chordal graphs) where many algorithmic problems could be solved efficiently. The *width* of a tree-decomposition $T(G) = (\{X_i | i \in I\}, T = (I, F))$ is $max_{i \in I} |X_i| - 1$. The *tree-width* of a graph $G$, denoted by $\mathsf{tw}(G)$, is the minimum width, over all tree-decompositions $T(G)$ of $G$ [27]. The trees are exactly the graphs with tree-width 1. The *length* of a tree-decomposition $T(G)$ of a graph $G$ is $\lambda := \max_{i \in I} \max_{u,v \in X_i} d_G(u, v)$ (i.e., each bag $X_i$ has diameter at most $\lambda$ in $G$). The *tree-length* of $G$, denoted by $\mathsf{tl}(G)$, is the minimum of the length, over all tree-decompositions of $G$ [8]. The chordal graphs are exactly the graphs with tree-length 1. Note that these two graph parameters are not related to each other. For instance, a clique on $n$ vertices has tree-length 1 and tree-width $n-1$,

whereas a cycle on $3n$ vertices has tree-width 2 and tree-length $n$. In [11], yet another graph parameter was introduced, which is very similar to the notion of tree-length and, as it turns out, is related to the TREE $t$-SPANNER problem. The *breadth* of a tree-decomposition $T(G)$ of a graph $G$ is the minimum integer $r$ such that for every $i \in I$ there is a vertex $v_i \in V(G)$ with $X_i \subseteq D_r(v_i, G)$ (i.e., each bag $X_i$ can be covered by a disk $D_r(v_i, G) := \{u \in V(G) | d_G(u, v_i) \leq r\}$ of radius at most $r$ in $G$). Note that vertex $v_i$ does not need to belong to $X_i$. The *tree-breadth* of $G$, denoted by $\mathsf{tb}(G)$, is the minimum of the breadth, over all tree-decompositions of $G$. Evidently, for any graph $G$, $1 \leq \mathsf{tb}(G) \leq \mathsf{tl}(G) \leq 2\mathsf{tb}(G)$ holds. Hence, if one parameter is bounded by a constant for a graph $G$ then the other parameter is bounded for $G$ as well.

We say that a family of graphs $\mathcal{G}$ is *of bounded tree-breadth* (*of bounded tree-width, of bounded tree-length*) if there is a constant $c$ such that for each graph $G$ from $\mathcal{G}$, $\mathsf{tb}(G) \leq c$ (resp., $\mathsf{tw}(G) \leq c$, $\mathsf{tl}(G) \leq c$).

It was shown in [11] that if a graph $G$ admits a tree $t$-spanner then its tree-breadth is at most $\lceil t/2 \rceil$ and its tree-length is at most $t$. Furthermore, any graph $G$ with tree-breadth $\mathsf{tb}(G) \leq \rho$ admits a tree $(2\rho\lfloor \log_2 n \rfloor)$-spanner that can be constructed in polynomial time. Thus, these two results gave a new $\log_2 n$-approximation algorithm for the TREE $t$-SPANNER problem on general (unweighted) graphs (see [11] for details). The algorithm of [11] is conceptually simpler than the previous $O(\log n)$-approximation algorithm proposed for the problem by Emek and Peleg [16].

Dourisboure et al. in [7] concerned with the construction of additive spanners with few edges for $n$-vertex graphs having a tree-decomposition into bags of diameter at most $\lambda$, i.e., the tree-length $\lambda$ graphs. For such graphs they construct additive $2\lambda$-spanners with $O(\lambda n + n \log n)$ edges, and additive $4\lambda$-spanners with $O(\lambda n)$ edges. Combining these results with the results of [11], we obtain the following interesting fact (in a sense, turning a multiplicative stretch into an additive surplus without much increase in the number of edges).

**Theorem 1.** *(combining [7] and [11]) If a graph $G$ admits a (multiplicative) tree $t$-spanner then it has an additive $2t$-spanner with $O(tn + n \log n)$ edges and an additive $4t$-spanner with $O(tn)$ edges, both constructible in polynomial time.*

This fact rises few intriguing questions. Does a polynomial time algorithm exist that, given an $n$-vertex graph $G$ admitting a (multiplicative) tree $t$-spanner, constructs an additive $O(t)$-spanner of $G$ with $O(n)$ or $O(n \log n)$ edges (where the number of edges in the spanner is independent of $t$)? Is a result similar to one presented by Elkin and Peleg in [15] possible? Namely, does a polynomial time algorithm exist that, given an $n$-vertex graph $G$ admitting a (multiplicative) tree $t$-spanner, constructs an additive $(t-1)$-spanner of $G$ with $O(n \log n)$ edges? If we allow to use more trees (like in collective tree spanners), does a polynomial time algorithm exist that, given an $n$-vertex graph $G$ admitting a (multiplicative) tree $t$-spanner, constructs a system of $\tilde{O}(1)$ collective additive tree $\tilde{O}(t)$-spanners of $G$ (where $\tilde{O}$ is similar to Big-$O$ notation up to a poly-logarithmic factor)? Note that an interesting question whether a multiplicative tree spanner can be turned into an additive tree spanner with a slight increase in the stretch is (negatively)

settled already in [16]: if there exist some $\delta = o(n)$ and $\epsilon > 0$ and a polynomial time algorithm that for any graph admitting a tree $t$-spanner constructs a tree $((6/5 - \epsilon)t + \delta)$-spanner, then P=NP.

We give some partial answers to these questions in Section 2. We investigate there a more general question whether a graph with bounded tree-breadth admits a small system of collective additive tree spanners. We show that any $n$-vertex graph $G$ has a system of at most $\log_2 n$ collective additive tree $(2\rho \log_2 n)$ -spanners, where $\rho \leq \mathsf{tb}(G)$. This settles also an open question from [7] whether a graph with tree-length $\lambda$ admits a small system of collective additive tree $\tilde{O}(\lambda)$-spanners.

As a consequence, we obtain that there is a polynomial time algorithm that, given an $n$-vertex graph $G$ admitting a (multiplicative) tree $t$-spanner, constructs: i) a system of at most $\log_2 n$ collective additive tree $O(t \log n)$-spanners of $G$ (compare with [11,16] where a multiplicative tree $O(t \log n)$-spanner was constructed for $G$ in polynomial time; thus, we "have turned" a multiplicative tree $O(t \log n)$-spanner into at most $\log_2 n$ collective additive tree $O(t \log n)$-spanners); ii) an additive $O(t \log n)$-spanner of $G$ with at most $n \log_2 n$ edges (compare with Theorem 1).

In Section 3 we generalize the method of Section 2. We define a new notion which combines both the tree-width and the tree-breadth of a graph.

The *k-breadth* of a tree-decomposition $T(G) = (\{X_i | i \in I\}, T = (I, F))$ of a graph $G$ is the minimum integer $r$ such that for each bag $X_i, i \in I$, there is a set of at most $k$ vertices $C_i = \{v_j^i | v_j^i \in V(G), j = 1, \ldots, k\}$ such that for each $u \in X_i$, we have $d_G(u, C_i) \leq r$ (i.e., each bag $X_i$ can be covered with at most $k$ disks of $G$ of radius at most $r$ each; $X_i \subseteq D_r(v_1^i, G) \cup \ldots \cup D_r(v_k^i, G)$). The *k-tree-breadth* of a graph $G$, denoted by $\mathsf{tb}_k(G)$, is the minimum of the $k$-breadth, over all tree-decompositions of $G$. We say that a family of graphs $\mathcal{G}$ is *of bounded k-tree-breadth,* if there is a constant $c$ such that for each graph $G$ from $\mathcal{G}$, $\mathsf{tb}_k(G) \leq c$. Clearly, for every graph $G$, $\mathsf{tb}(G) = \mathsf{tb}_1(G)$, and $\mathsf{tw}(G) \leq k - 1$ if and only if $\mathsf{tb}_k(G) = 0$. Thus, the notions of the tree-width and the tree-breadth are particular cases of the $k$-tree-breadth.

In Section 3, we show that any $n$-vertex graph $G$ with $\mathsf{tb}_k(G) \leq \rho$ has a system of at most $k(1+\log_2 n)$ collective additive tree $(2\rho(1+\log_2 n))$-spanners. In Section 4, we extend a result from [11] and show that if a graph $G$ admits a (multiplicative) $t$-spanner $H$ with $\mathsf{tw}(H) = k - 1$ then its $k$-tree-breadth is at most $\lceil t/2 \rceil$. As a consequence, we obtain that, for every fixed $k$, there is a polynomial time algorithm that, given an $n$-vertex graph $G$ admitting a (multiplicative) $t$-spanner with tree-width at most $k-1$, constructs: i) a system of at most $k(1+\log_2 n)$ collective additive tree $O(t \log n)$-spanners of $G$; ii) an additive $O(t \log n)$-spanner of $G$ with at most $O(kn \log n)$ edges.

All proofs omitted in this extended abstract and a few illustrative figures can be found in the full version of the paper [9].

**Preliminaries.** All graphs occurring in this paper are connected, finite, unweighted, undirected, loopless and without multiple edges. We call $G = (V, E)$ an *n-vertex m-edge graph* if $|V| = n$ and $|E| = m$. A *clique* is a set of pairwise

adjacent vertices of $G$. By $G[S]$ we denote a subgraph of $G$ induced by vertices of $S \subseteq V$. Let also $G \setminus S$ be the graph $G[V \setminus S]$ (which is not necessarily connected). A set $S \subseteq V$ is called a *separator* of $G$ if the graph $G[V \setminus S]$ has more than one connected component, and $S$ is called a *balanced separator* of $G$ if each connected component of $G[V \setminus S]$ has at most $|V|/2$ vertices. A set $C \subseteq V$ is called a *balanced clique-separator* of $G$ if $C$ is both a clique and a balanced separator of $G$. For a vertex $v$ of $G$, the sets $N_G(v) = \{w \in V | vw \in E\}$ and $N_G[v] = N_G(v) \cup \{v\}$ are called the *open neighborhood* and the *closed neighborhood* of $v$, respectively.

In a graph $G$ the *length* of a path from a vertex $v$ to a vertex $u$ is the number of edges in the path. The *distance* $d_G(u, v)$ between vertices $u$ and $v$ is the length of a shortest path connecting $u$ and $v$ in $G$. The *diameter* in $G$ of a set $S \subseteq V$ is $\max_{x,y \in S} d_G(x, y)$ and its *radius* in $G$ is $\min_{x \in V} \max_{y \in S} d_G(x, y)$ (in some papers they are called the *weak diameter* and the *weak radius* to indicate that the distances are measured in $G$ not in $G[S]$). The *disk* of $G$ of radius $k$ centered at vertex $v$ is the set of all vertices at distance at most $k$ to $v$: $D_k(v, G) = \{w \in V | d_G(v, w) \leq k\}$. A disk $D_k(v, G)$ is called a *balanced disk-separator* of $G$ if the set $D_k(v, G)$ is a balanced separator of $G$.

## 2 Collective Additive Tree Spanners and Tree-Breadth

In this section, we show that every $n$-vertex graph $G$ has a system of at most $\log_2 n$ collective additive tree $(2\rho \log_2 n)$-spanners, where $\rho \leq \mathsf{tb}(G)$. We also discuss consequences of this result. Our method is a generalization of techniques used in [13] and [11]. We will assume that $n \geq 4$ since any connected graph with at most 3 vertices has an additive tree 1-spanner.

Note that we do not assume here that a tree-decomposition $T(G)$ of breadth $\rho$ is given for $G$ as part of the input. Our method does not need to know $T(G)$, our algorithm works directly on $G$. For a given graph $G$ and an integer $\rho$, even checking whether $G$ has a tree-decomposition of breadth $\rho$ could be a hard problem. For example, while graphs with tree-length 1 (as they are exactly the chordal graphs) can be recognized in linear time, the problem of determining whether a given graph has tree-length at most $\lambda$ is NP-complete for every fixed $\lambda > 1$ (see [20]). We will need the following results proven in [11].

**Lemma 1 ([11]).** *Every graph $G$ has a balanced disk-separator $D_r(v, G)$ centered at some vertex $v$, where $r \leq \mathsf{tb}(G)$. For an arbitrary graph $G$ with $n$ vertices and $m$ edges a balanced disk-separator $D_r(v, G)$ with minimum $r$ can be found in $O(nm)$ time.*

**Hierarchical Decomposition of a Graph with Bounded Tree-Breadth.** In this subsection, following [11], we show how to decompose a graph with bounded tree-breadth and build a hierarchical decomposition tree for it. This hierarchical decomposition tree is used later for construction of collective additive tree spanners for such a graph.

Let $G = (V, E)$ be an arbitrary connected $n$-vertex $m$-edge graph with a disk-separator $D_r(v, G)$. Also, let $G_1, \ldots, G_q$ be the connected components of

$G[V \setminus D_r(v,G)]$. Denote by $S_i := \{x \in V(G_i) |\ d_G(x, D_r(v,G)) = 1\}$ the neighborhood of $D_r(v,G)$ with respect to $G_i$. Let also $G_i^+$ be the graph obtained from component $G_i$ by adding a vertex $c_i$ (*representative* of $D_r(v,G)$) and making it adjacent to all vertices of $S_i$, i.e., for a vertex $x \in V(G_i)$, $c_i x \in E(G_i^+)$ if and only if there is a vertex $x_D \in D_r(v,G)$ with $xx_D \in E(G)$. In what follows, we will call vertex $c_i$ a *meta vertex representing disk* $D_r(v,G)$ *in graph* $G_i^+$. Given a graph $G$ and its disk-separator $D_r(v,G)$, the graphs $G_1^+, \ldots, G_q^+$ can be constructed in total time $O(m)$. Furthermore, the total number of edges in the graphs $G_1^+, \ldots, G_q^+$ does not exceed the number of edges in $G$, and the total number of vertices (including $q$ meta vertices) in those graphs does not exceed the number of vertices in $G[V \setminus D_r(v,G)]$ plus $q$.

Denote by $G_{/e}$ the graph obtained from $G$ by contracting its edge $e$. Recall that edge $e$ contraction is an operation which removes $e$ from $G$ while simultaneously merging together the two vertices $e$ previously connected. If a contraction results in multiple edges, we delete duplicates of an edge to stay within the class of simple graphs. The operation may be performed on a set of edges by contracting each edge (in any order).

**Lemma 2** ([11]). *For any graph $G$ and its edge $e$, $\mathsf{tb}(G) \le \rho$ implies $\mathsf{tb}(G_{/e}) \le \rho$. Consequently, if $\mathsf{tb}(G) \le \rho$, then $\mathsf{tb}(G_i^+) \le \rho$ for each $i = 1, \ldots, q$.*

Clearly, one can get $G_i^+$ from $G$ by repeatedly contracting (in any order) edges of $G$ that are not incident to vertices of $G_i$. In other words, $G_i^+$ is a minor of $G$. Recall that a graph $G'$ is a *minor* of $G$ if $G'$ can be obtained from $G$ by contracting some edges, deleting some edges, and deleting some isolated vertices. The order in which a sequence of such contractions and deletions is performed on $G$ does not affect the resulting graph $G'$.

Let $G = (V,E)$ be a connected $n$-vertex, $m$-edge graph and assume that $\mathsf{tb}(G) \le \rho$. Lemma 1 guarantees that $G$ has a balanced disk-separator $D_r(v,G)$ with $r \le \rho$, which can be found in $O(nm)$ time by an algorithm that works directly on graph $G$ and does not require construction of a tree-decomposition of $G$ of breadth $\le \rho$. Using these and Lemma 2, we can build a (rooted) *hierarchical tree* $\mathcal{H}(G)$ for $G$ as follows. If $G$ is a connected graph with at most 5 vertices, then $\mathcal{H}(G)$ is one node tree with root node $(V(G), G)$. Otherwise, find a balanced disk-separator $D_r(v,G)$ in $G$ with minimum $r$ (see Lemma 1) and construct the corresponding graphs $G_1^+, G_2^+, \ldots, G_q^+$. For each graph $G_i^+$ $(i = 1, \ldots, q)$ (by Lemma 2, $\mathsf{tb}(G_i^+) \le \rho$), construct a hierarchical tree $\mathcal{H}(G_i^+)$ recursively and build $\mathcal{H}(G)$ by taking the pair $(D_r(v,G), G)$ to be the root and connecting the root of each tree $\mathcal{H}(G_i^+)$ as a child of $(D_r(v,G), G)$.

The depth of this tree $\mathcal{H}(G)$ is the smallest integer $k$ such that $\frac{n}{2^k} + \frac{1}{2^{k-1}} + \ldots + \frac{1}{2} + 1 \le 5$, that is, the depth is at most $\log_2 n - 1$. It is also easy to see that, given a graph $G$ with $n$ vertices and $m$ edges, a hierarchical tree $\mathcal{H}(G)$ can be constructed in $O(nm \log^2 n)$ total time. There are at most $O(\log n)$ levels in $\mathcal{H}(G)$, and one needs to do at most $O(nm \log n)$ operations per level since the total number of edges in the graphs of each level is at most $m$ and the total number of vertices in those graphs can not exceed $O(n \log n)$.

For an internal (i.e., non-leaf) node $Y$ of $\mathcal{H}(G)$, since it is associated with a pair $(D_{r'}(v', G'), G')$, where $r' \leq \rho$, $G'$ is a minor of $G$ and $v'$ is the center of disk $D_{r'}(v', G')$ of $G'$, it will be convenient, in what follows, to denote $G'$ by $G(\downarrow Y)$, $v'$ by $c(Y)$, $r'$ by $r(Y)$, and $D_{r'}(v', G')$ by $Y$ itself. Thus, $(D_{r'}(v', G'), G') = (D_{r(Y)}(c(Y), G(\downarrow Y)), G(\downarrow Y)) = (Y, G(\downarrow Y))$ in these notations, and we identify node $Y$ of $\mathcal{H}(G)$ with the set $Y = D_{r(Y)}(c(Y), G(\downarrow Y))$ and associate with this node also the graph $G(\downarrow Y)$. Each leaf $Y$ of $\mathcal{H}(G)$, since it corresponds to a pair $(V(G'), G')$, we identify with the set $Y = V(G')$ and use, for a convenience, the notation $G(\downarrow Y)$ for $G'$. If now $(Y^0, Y^1, \ldots, Y^h)$ is the path of $\mathcal{H}(G)$ connecting the root $Y^0$ of $\mathcal{H}(G)$ with a node $Y^h$, then the vertex set of the graph $G(\downarrow Y^h)$ consists of some (original) vertices of $G$ plus at most $h$ meta vertices representing the disks $D_{r(Y)}(c(Y^i), G(\downarrow Y^i)) = Y^i$, $i = 0, 1, \ldots, h-1$. Note also that each (original) vertex of $G$ belongs to exactly one node of $\mathcal{H}(G)$.

**Construction of Collective Additive Tree Spanners.** Unfortunately, the class of graphs of bounded tree-breadth is not hereditary, i.e., induced subgraphs of a graph with tree-breath $\rho$ are not necessarily of tree-breadth at most $\rho$ (for example, a cycle of length $\ell$ with one extra vertex adjacent to each vertex of the cycle has tree-breadth 1, but the cycle itself has tree-breadth $\ell/3$). Thus, the method presented in [13], for constructing collective additive tree spanners for hereditary classes of graphs admitting balanced disk-separators, cannot be applied directly to the graphs of bounded tree-breadth. Nevertheless, we will show that, with the help of Lemma 2, the notion of hierarchical tree from previous subsection and a careful analysis of distance changes (see Lemma 3), it is possible to generalize the method of [13] and construct in polynomial time for every $n$-vertex graph $G$ a system of at most $\log_2 n$ collective additive tree $(2\rho \log_2 n)$-spanners, where $\rho \leq \mathsf{tb}(G)$. Unavoidable presence of meta vertices in the graphs resulting from a hierarchical decomposition of the original graph $G$ complicates the construction and the analysis. Recall that, in [13], it was shown that if every induced subgraph of a graph $G$ enjoys a balanced disk-separator with radius $\leq r$, then $G$ admits a system of at most $\log_2 n$ collective additive tree $2r$-spanners.

Let $G = (V, E)$ be a connected $n$-vertex, $m$-edge graph and assume that $\mathsf{tb}(G) \leq \rho$. Let $\mathcal{H}(G)$ be a hierarchical tree of $G$. Consider an arbitrary internal node $Y^h$ of $\mathcal{H}(G)$, and let $(Y^0, Y^1, \ldots, Y^h)$ be the path of $\mathcal{H}(G)$ connecting the root $Y^0$ of $\mathcal{H}(G)$ with $Y^h$. Let $\widehat{G}(\downarrow Y^j)$ be the graph obtained from $G(\downarrow Y^j)$ by removing all its meta vertices (note that $\widehat{G}(\downarrow Y^j)$ may be disconnected).

**Lemma 3.** *For any vertex $z$ from $Y^h \cap V(G)$ there exists an index $i \in \{0, \ldots, h\}$ such that the vertices $z$ and $c(Y^i)$ can be connected in the graph $\widehat{G}(\downarrow Y^i)$ by a path of length at most $\rho(h+1)$. In particular, $d_G(z, c(Y^i)) \leq \rho(h+1)$ holds.*

Consider arbitrary vertices $x$ and $y$ of $G$, and let $S(x)$ and $S(y)$ be the nodes of $\mathcal{H}(G)$ containing $x$ and $y$, respectively. Let also $NCA_{\mathcal{H}(G)}(S(x), S(y))$ be the nearest common ancestor of nodes $S(x)$ and $S(y)$ in $\mathcal{H}(G)$ and $(Y^0, Y^1, \ldots, Y^h)$ be the path of $\mathcal{H}(G)$ connecting the root $Y^0$ of $\mathcal{H}(G)$ with $NCA_{\mathcal{H}(G)}(S(x), S(y)) = Y^h$ (i.e., $Y^0, Y^1, \ldots, Y^h$ are the common ancestors of $S(x)$ and $S(y)$).

**Lemma 4.** *Any path $P^G_{x,y}$ connecting vertices $x$ and $y$ in $G$ contains a vertex from $Y^0 \cup Y^1 \cup \ldots \cup Y^h$.*

Let $SP^G_{x,y}$ be a shortest path of $G$ connecting vertices $x$ and $y$, and let $Y^i$ be the node of the path $(Y^0, Y^1, \ldots, Y^h)$ with the smallest index such that $SP^G_{x,y} \bigcap Y^i \neq \emptyset$ in $G$. The following lemma holds.

**Lemma 5.** *For each $j = 0, \ldots, i$, $d_G(x,y) = d_{G'}(x,y)$ where $G' := \widehat{G}(\downarrow Y^j)$.*

Let now $B^i_1, \ldots, B^i_{p_i}$ be the nodes at depth $i$ of the tree $\mathcal{H}(G)$. For each node $B^i_j$ that is not a leaf of $\mathcal{H}(G)$, consider its (central) vertex $c^i_j := c(B^i_j)$. If $c^i_j$ is an original vertex of $G$ (not a meta vertex created during the construction of $\mathcal{H}(G)$), then define a connected graph $G^i_j$ obtained from $G(\downarrow B^i_j)$ by removing all its meta vertices. If removal of those meta vertices produced few connected components, choose as $G^i_j$ that component which contains the vertex $c^i_j$. Denote by $T^i_j$ a BFS–tree of graph $G^i_j$ rooted at vertex $c^i_j$ of $B^i_j$. If $B^i_j$ is a leaf of $\mathcal{H}(G)$, then $B^i_j$ has at most 5 vertices. In this case, remove all meta vertices from $G(\downarrow B^i_j)$ and for each connected component of the resulting graph construct an additive tree spanner with optimal surplus $\leq 3$. Denote the resulting subtree (forest) by $T^i_j$. The trees $T^i_j$ $(i = 0, 1, \ldots, depth(\mathcal{H}(G)),\ j = 1, 2, \ldots, p_i)$, obtained this way, are called *local subtrees* of $G$. Clearly, the construction of these local subtrees can be incorporated into the procedure of constructing hierarchical tree $\mathcal{H}(G)$ of $G$.

**Lemma 6.** *For any two vertices $x, y \in V(G)$, there exists a local subtree $T$ such that $d_T(x,y) \leq d_G(x,y) + 2\rho \log_2 n - 1$.*

This lemma implies two important results. Let $G$ be a graph with $n$ vertices and $m$ edges having $\mathsf{tb}(G) \leq \rho$. Also, let $\mathcal{H}(G)$ be its hierarchical tree and $\mathcal{LT}(G)$ be the family of all its local subtrees (defined above). Consider a graph $H$ obtained by taking the union of all local subtrees of $G$ (by putting all of them together), i.e., $H := \bigcup\{T^i_j | T^i_j \in \mathcal{LT}(G)\} = (V, \cup\{E(T^i_j) | T^i_j \in \mathcal{LT}(G)\})$. Clearly, $H$ is a spanning subgraph of $G$, constructible in $O(nm \log^2 n)$ total time, and, for any two vertices $x$ and $y$ of $G$, $d_H(x,y) \leq d_G(x,y) + 2\rho \log_2 n - 1$ holds. Also, since for every level $i$ $(i = 0, 1, \ldots, depth(\mathcal{H}(G)))$ of hierarchical tree $\mathcal{H}(G)$, the corresponding local subtrees $T^i_1, \ldots, T^i_{p_i}$ are pairwise vertex-disjoint, their union has at most $n-1$ edges. Therefore, $H$ cannot have more than $(n-1)\log_2 n$ edges in total. Thus, we have proven the following result.

**Theorem 2.** *Every graph $G$ with $n$ vertices and $\mathsf{tb}(G) \leq \rho$ admits an additive $(2\rho \log_2 n)$–spanner with at most $n \log_2 n$ edges. Furthermore, such a sparse additive spanner of $G$ can be constructed in polynomial time.*

Instead of taking the union of all local subtrees of $G$, one can fix $i$ $(i \in \{0, 1, \ldots, depth(\mathcal{H}(G))\})$ and consider separately the union of only local subtrees $T^i_1, \ldots, T^i_{p_i}$, corresponding to the level $i$ of the hierarchical tree $\mathcal{H}(G)$, and then extend in linear $O(m)$ time that forest to a spanning tree $T^i$ of $G$ (using, for

example, a variant of the Kruskal's Spanning Tree algorithm for the unweighted graphs). We call this tree $T^i$ the *spanning tree of $G$ corresponding to the level $i$ of the hierarchical tree $\mathcal{H}(G)$*. In this way we can obtain at most $\log_2 n$ spanning trees for $G$, one for each level $i$ of $\mathcal{H}(G)$. Denote the collection of those spanning trees by $\mathcal{T}(G)$. Thus, we obtain the following theorem.

**Theorem 3.** *Every graph $G$ with $n$ vertices and $\mathsf{tb}(G) \leq \rho$ admits a system $\mathcal{T}(G)$ of at most $\log_2 n$ collective additive tree $(2\rho \log_2 n)$–spanners. Furthermore, such a system of collective additive tree spanners of $G$ can be constructed in polynomial time.*

**Additive Spanners for Graphs Having (Multiplicative) Tree $t$–spanners.** Now we give implications of the above results for the class of tree $t$–spanner admissible graphs. In [11], the following important ("bridging") lemma was proven.

**Lemma 7 ([11]).** *If a graph $G$ admits a tree $t$-spanner then its tree-breadth is at most $\lceil t/2 \rceil$.*

Note that the tree-breadth bounded by $\lceil t/2 \rceil$ provides only a necessary condition for a graph to have a multiplicative tree $t$-spanner. There are (chordal) graphs which have tree-breadth 1 but any multiplicative tree $t$-spanner of them has $t = \Omega(\log n)$ [11]. Furthermore, a cycle on $3n$ vertices has tree-breadth $n$ but admits a system of 2 collective additive tree 0-spanners.

Combining Lemma 7 with Theorem 2 and Theorem 3, we deduce the following results.

**Theorem 4.** *Let $G$ be a graph with $n$ vertices and $m$ edges having a (multiplicative) tree $t$–spanner. Then, $G$ admits an additive $(2\lceil t/2 \rceil \log_2 n)$–spanner with at most $n \log_2 n$ edges constructible in $O(nm \log^2 n)$ time.*

**Theorem 5.** *Let $G$ be a graph with $n$ vertices and $m$ edges having a (multiplicative) tree $t$–spanner. Then, $G$ admits a system $\mathcal{T}(G)$ of at most $\log_2 n$ collective additive tree $(2\lceil t/2 \rceil \log_2 n)$–spanners constructible in $O(nm \log^2 n)$ time.*

## 3 Graphs with Bounded $k$-Tree-Breadth, $k \geq 2$

In this section, we extend the approach of Section 2 and show that any $n$-vertex graph $G$ with $\mathsf{tb}_k(G) \leq \rho$ has a system of at most $k(1+\log_2 n)$ collective additive tree $(2\rho(1+\log_2 n))$-spanners constructible in polynomial time for every fixed $k$.

**Balanced Separators for Graphs with Bounded $k$-Tree-Breadth.** We say that a graph $G = (V, E)$ with $|V| \geq k$ has a *balanced* $\mathbf{D^k_r}$*-separator* if there exists a collection of $k$ disks $D_r(v_1, G), D_r(v_2, G), \ldots, D_r(v_k, G)$ in $G$, centered at (different) vertices $v_1, v_2, \ldots, v_k$ and each of radius $r$, such that the union of those disks $\mathbf{D^k_r} := \bigcup_{i=1}^{k} D_r(v_i, G)$ forms a balanced separator of $G$, i.e., each connected component of $G[V \setminus \mathbf{D^k_r}]$ has at most $|V|/2$ vertices. The following result generalizes Lemma 1.

**Lemma 8.** *Every graph $G$ with at least $k$ vertices and $\mathsf{tb}_k(G) \leq \rho$ has a* balanced $\mathbf{D}^{\mathbf{k}}_{\rho}$-separator. *For an arbitrary graph $G$ with $n \geq k$ vertices and $m$ edges, a balanced $\mathbf{D}^{\mathbf{k}}_{\mathbf{r}}$-separator with the smallest radius $r$ can be found in $O(n^k m)$ time.*

**Collective Additive Tree Spanners of a Graph with Bounded $k$-Tree-Breadth.** Using Lemma 8, we generalize the technique of Section 2 and obtain the following results for the graphs with bounded $k$-tree-breadth ($k \geq 2$). Details can be found in the full version of this extended abstract (see [9]).

**Theorem 6.** *Every graph $G$ with $n$ vertices and $\mathsf{tb}_k(G) \leq \rho$ admits an additive $(2\rho(1+\log_2 n))$–spanner with at most $O(kn \log n)$ edges constructible in polynomial time for every fixed $k$.*

**Theorem 7.** *Every $n$-vertex graph $G$ with $\mathsf{tb}_k(G) \leq \rho$ admits a system $\mathcal{T}(G)$ of at most $k(1+\log_2 n)$ collective additive tree $(2\rho(1+\log_2 n))$-spanners constructible in polynomial time for every fixed $k$.*

## 4 Additive Spanners for Graphs Admitting $t$–Spanners of Bounded Tree-Width

In this section, we show that if a graph $G$ admits a (multiplicative) $t$-spanner $H$ with $\mathsf{tw}(H) = k-1$ then its $k$-tree-breadth is at most $\lceil t/2 \rceil$. As a consequence, we obtain that, for every fixed $k$, there is a polynomial time algorithm that, given an $n$-vertex graph $G$ admitting a (multiplicative) $t$-spanner with tree-width at most $k-1$, constructs a system of at most $k(1+\log_2 n)$ collective additive tree $O(t \log n)$-spanners of $G$.

**$k$-Tree-Breadth of a Graph Admitting a $t$-Spanner of Bounded Tree-width.** Let $H$ be a graph with tree-width $k-1$, and let $T(H) = (\{X_i | i \in I\}, T = (I, F))$ be its tree-decomposition of width $k-1$. For an integer $r \geq 0$, denote by $X_i^{(r)}$, $i \in I$, the set $D_r(X_i, H) := \bigcup_{x \in X_i} D_r(x, H)$. Clearly, $X_i^{(0)} = X_i$ for every $i \in I$. The following important lemmas hold.

**Lemma 9.** *For every integer $r \geq 0$, $T^{(r)}(H) := (\{X_i^{(r)} | i \in I\}, T = (I, F))$ is a tree-decomposition of $H$ with $k$-breadth $\leq r$.*

**Lemma 10.** *If a graph $G$ admits a $t$-spanner with tree-width $k-1$, then $\mathsf{tb}_k(G) \leq \lceil t/2 \rceil$.*

**Consequences.** Now we give two implications of the above results for the class of graphs admitting (multiplicative) $t$–spanners with tree-width $k-1$. They are direct consequences of Lemma 10, Theorem 6 and Theorem 7.

**Theorem 8.** *Let $G$ be a graph with $n$ vertices and $m$ edges having a (multiplicative) $t$–spanner with tree-width $k-1$. Then, $G$ admits an additive $(2\lceil t/2 \rceil(1+\log_2 n))$–spanner with at most $O(kn \log n)$ edges constructible in polynomial time for every fixed $k$.*

**Theorem 9.** *Let $G$ be a graph with $n$ vertices and $m$ edges having a (multiplicative) $t$–spanner with tree-width $k-1$. Then, $G$ admits a system $\mathcal{T}(G)$ of at most $k(1+\log_2 n)$ collective additive tree $(2\lceil t/2\rceil(1+\log_2 n))$–spanners constructible in polynomial time for every fixed $k$.*

## References


1. Bartal, Y.: Probabilistic approximations of metric spaces and its algorithmic applications. In: FOCS 1996, pp. 184–193 (1996)
2. Bartal, Y., Blum, A., Burch, C., Tomkins, A.: A polylog-competitive algorithm for metrical task systems. In: STOC 1997, pp. 711–719 (1997)
3. Cai, L., Corneil, D.G.: Tree spanners. SIAM J. Disc. Math. 8, 359–387 (1995)
4. Charikar, M., Chekuri, C., Goel, A., Guha, S., Plotkin, S.A.: Approximating a finite metric by a small number of tree metrics. In: FOCS 1998, pp. 379–388 (1998)
5. Chew, L.P.: There are planar graphs almost as good as the complete graph. J. of Comp. and Sys. Sci. 39, 205–219 (1989)
6. Diestel, R.: Graph Theory, 2nd edn. Graduate Texts in Math., vol. 173. Springer (2000)
7. Dourisboure, Y., Dragan, F.F., Gavoille, C., Yan, C.: Spanners for bounded tree-length graphs. Theor. Comput. Sci. 383, 34–44 (2007)
8. Dourisboure, Y., Gavoille, C.: Tree-decompositions with bags of small diameter. Disc. Math. 307, 2008–2029 (2007)
9. Dragan, F.F., Abu-Ata, M.: Collective Additive Tree Spanners of Bounded Tree-Breadth Graphs with Generalizations and Consequences (Full version of this extended abstract), CoRR abs/1207.2506 (2012)
10. Dragan, F.F., Fomin, F.V., Golovach, P.A.: Spanners in sparse graphs. J. Comput. Syst. Sci. 77, 1108–1119 (2011)
11. Dragan, F.F., Köhler, E.: An Approximation Algorithm for the Tree $t$-Spanner Problem on Unweighted Graphs via Generalized Chordal Graphs. In: Goldberg, L.A., Jansen, K., Ravi, R., Rolim, J.D.P. (eds.) APPROX/RANDOM 2011. LNCS, vol. 6845, pp. 171–183. Springer, Heidelberg (2011)
12. Dragan, F.F., Yan, C.: Collective Tree Spanners in Graphs with Bounded Parameters. Algorithmica 57, 22–43 (2010)
13. Dragan, F.F., Yan, C., Lomonosov, I.: Collective tree spanners of graphs. SIAM J. Disc. Math. 20, 241–260 (2006)
14. Elkin, M., Emek, Y., Spielman, D.A., Teng, S.H.: Lower-stretch spanning trees. SIAM J. Comput. 38, 608–628 (2008)
15. Elkin, M., Peleg, D.: Approximating $k$-spanner problems for $k \geq 2$. Theor. Comput. Sci. 337, 249–277 (2005)
16. Emek, Y., Peleg, D.: Approximating minimum max-stretch spanning trees on unweighted graphs. SIAM J. Comput. 38, 1761–1781 (2008)
17. Fakcharoenphol, F., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. J. Comput. Syst. Sci. 69, 485–497 (2004)
18. Gupta, A., Kumar, A., Rastogi, R.: Traveling with a pez dispenser (or, routing issues in mpls). SIAM J. Comput. 34, 453–474 (2004)
19. Liestman, A.L., Shermer, T.: Additive graph spanners. Networks 23, 343–364 (1993)
20. Lokshtanov, D.: On the complexity of computing tree-length. Disc. Appl. Math. 158, 820–827 (2010)

21. Peleg, D.: Low Stretch Spanning Trees. In: Diks, K., Rytter, W. (eds.) MFCS 2002. LNCS, vol. 2420, pp. 68–80. Springer, Heidelberg (2002)
22. Peleg, D., Reshef, E.: Low complexity variants of the arrow distributed directory. J. Comput. System Sci. 63, 474–485 (2001)
23. Peleg, D., Schäffer, A.A.: Graph Spanners. J. Graph Theory 13, 99–116 (1989)
24. Peleg, D., Ullman, J.D.: An optimal synchronizer for the hypercube. SIAM J. Comput. 18, 740–747 (1989)
25. Peleg, D., Upfal, E.: A tradeoff between space and efficiency for routing tables (extended abstract). In: STOC 1988, pp. 43–52 (1988)
26. Prisner, E.: Distance Approximating Spanning Trees. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 499–510. Springer, Heidelberg (1997)
27. Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. J. of Algorithms 7, 309–322 (1998)

# Weak Arithmetic Completeness of Object-Oriented First-Order Assertion Networks*

Stijn de Gouw[2,3], Frank de Boer[2,3], Wolfgang Ahrendt[1], and Richard Bubel[4]

[1] Chalmers University, Göteborg, Sweden
[2] CWI, Amsterdam, The Netherlands
[3] Leiden University, The Netherlands
[4] Technische Universität Darmstadt, Germany

**Abstract.** We present a completeness proof of the inductive assertion method for object-oriented programs extended with auxiliary variables. The class of programs considered are assumed to compute over structures which include the standard interpretation of Presburger arithmetic. Further, the assertion language is first-order, i.e., quantification only ranges over basic types like that of the natural numbers, Boolean and Object.

## 1 Introduction

In [5], Cook introduced a general condition of completeness of Hoare logics in terms of the expressibility of the weakest precondition. Harel defined in [9] a general class of (first-order) structures which include the standard interpretation of Peano Arithmetic. For this class standard coding techniques suffice to express the weakest precondition. This is *not* the case for programs with general abstract data structures as studied by Tucker and Zucker in [18]. They prove therefore expressibility of the weakest precondition in a weak second-order language which contains quantification over finite sequences.

In this paper we study arithmetic completeness of inductive assertion networks [7] for proving correctness of object-oriented programs. Our main contribution shows that the inductive assertion method is complete for a class of programs which compute over *weak* arithmetic structures. Such structures include the standard interpretation of Presburger arithmetic. Though multiplication can be simulated in the programming language by repeated addition using a while loop, omitting multiplication limits the expressiveness of the assertion language severely, as can be seen by the following argument. In a Turing complete programming language, any recursively enumerable set is the weakest precondition of some program. But by Presburger's result [16], formulas of Presburger arithmetic define only recursive sets, and hence, cannot express the weakest precondition (nor strongest postcondition) of arbitrary programs.

* This research is partly funded by the EU project FP7-231620 HATS: Highly Adaptable and Trustworthy Software using Formal Models (http://www.hats-project.eu/).

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 207–219, 2013.

We show however that the strongest postcondition *is* expressible using only Presburger arithmetic for object-oriented programs, when appropriately instrumented with auxiliary array variables. In particular we demonstrate that treating arrays as objects allows a direct representation of a computation at the abstraction level of both the programming language and the first-order logic and enables us to express arbitrary properties of the heap in first-order logic. As a practical consequence, any first-order logic theorem prover can be used to prove verification conditions of instrumented object-oriented programs. In contrast to second-order logic (as used by Tucker and Zucker) or recursive predicates (separation logic) which are traditionally used to express heap properties, first-order logic has desirable proof-theoretical properties: first-order logic is the strongest logic satisfying completeness and the Löwenheim-Skolem theorem [10]. Our approach is tool supported by a special version of KeY [4], a state-of-the-art prover for Java. On the theoretical side we show that the above expressiveness result implies completeness: for any valid pre-/postcondition specification of an object-oriented program there is an inductive assertion network of the program extended with *auxiliary variables*.

Finally, using auxiliary variables allows us to restrict the network to *recursive* assertions in case the given pre- and postcondition are recursive. This possibility is of fundamental practical importance as recursive assertions are effectively computable and, hence, we can use them for run-time checking of programs.

*Related work.* In [6] completeness of the inductive assertion method has been studied for recursive programs only and without the use of auxiliary variables. The absence of auxiliary variables made it necessary to resort to an infinite collection of intermediate assertions. Apt showed in [2] that recursive assertions are complete for while programs extended with auxiliary variables. In this paper we combine and extend on the above results by showing that recursive assertions are complete for object-oriented programs extended with auxiliary variables.

Completeness of Hoare logics for object-oriented programs is also formally proven e.g. in [15]. This completeness result however is based on the expressibility of the strongest postcondition in a weak second-order language which contains quantification over finite sequences. In [3] completeness for an object-oriented core language without object creation is proven assuming the standard interpretation of Peano arithmetic for the expressibility of the weakest precondition. We are not aware of any other completeness result based on weak arithmetic structures using only Presburger arithmetic and an assertion language which only contains quantification over basic types., i.e., integer, Boolean and Object.

## 2 The Programming and Specification Language

We introduce now our *core object-oriented language*. The language is strongly typed and contains the primitive types Presburger and Boolean. The only operations provided by Presburger are those of Presburger arithmetic (0, successor and addition). The only operations allowed on Booleans are those of Boolean

algebra. Additionally there are user-defined class types C, predefined class types $T[\,]$ of *unbounded* arrays in which the elements are of type $T$ and a union type Object. Arrays can be dynamically allocated and are indexed by natural numbers. Multi-dimensional arrays are modeled (as in Java) as arrays of arrays. We assume a transitive reflexive subtype relation between types with Object being the supertype of any class type. Our language can be statically type checked.

### 2.1 Syntax

Expressions of our language are side-effect free and generated by the grammar:

$$e ::= u \mid e.x \mid \mathsf{null} \mid e_1 = e_2 \mid \mathsf{if}\ b\ \mathsf{then}\ e\ \mathsf{fi} \mid \mathsf{if}\ b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2\ \mathsf{fi} \mid e_1[e_2] \mid f(e_1, \ldots, e_n) \mid C(e)$$

Variables are indicated by $u$ while $x$ denotes a typical field. The Boolean expression $e_1 = e_2$ denotes the test for equality between the values of $e_1$ and $e_2$. For object expressions we use Java reference semantics, i.e., to be equal $e_1$ and $e_2$ must denote the same object identity. The expression $\mathsf{if}\ b\ \mathsf{then}\ e\ \mathsf{fi}$ has value $e$ if the Boolean expression $b$ is true, otherwise it has an arbitrary value. This expression allows a systematic approach to proving properties about partial functions. A conditional expression is denoted by $\mathsf{if}\ b\ \mathsf{then}\ e_1\ \mathsf{else}\ e_2\ \mathsf{fi}$. The motivation for including it in our core language is that it significantly simplifies treatment of aliasing. If $e_1$ is an expression of type $T[\,]$ and $e_2$ is an expression of type Presburger then $e_1[e_2]$ is an expression of type $T$, also called a subscripted variable. Here $T$ itself can be an array type. For example, if $a$ is an array variable of type Presburger[ ][ ] then the expression $a[0]$ denotes an array of type Presburger[ ]. The function $f(e_1, ..., e_n)$ denotes a Presburger arithmetic or Boolean operation of arity $n$. For class types C the Boolean expression $C(e)$ is true if and only if the dynamic type of $e$ is $C$. Dynamic binding can be simulated in our core language with such expressions. Expressions of a class type can only be compared for equality, dereferenced, accessed as an array if the object is of an array type, or appear as arguments of a class predicate, if-expression, or conditional expression.

The language of statements is generated by the following grammar:

$$s ::= s_1; s_2 \mid \mathsf{if}\ b\ \mathsf{then}\ s_2\ \mathsf{else}\ s_3\ \mathsf{fi} \mid \mathsf{while}\ \ e\ \ \mathsf{do}\ \ s\ \ \mathsf{od} \mid \mathsf{abort} \mid$$
$$e_0.m(e_1, ..., e_n) \mid u := \mathsf{new} \mid u := e \mid e_1[e_2] := e \mid e_1.x := e$$

The abort statement causes a failure. A statement $u :=$ new assigns to the program variable $u$ a newly created object of the declared type (possibly an array type) of $u$. Objects are never destroyed. We assume every statement and expression to be well-typed. A *program* in our language consists of a *main statement* together with sets for variable-, field- and method declarations (respectively Var and $F_{\mathrm{C}}$, $M_{\mathrm{C}}$ for every class C).

Assertions are generated by the following *first-order* language:

$$\phi ::= b \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi \rightarrow \phi_2 \mid \exists l : \phi \mid \forall l : \phi$$

Here, $b$ is a Boolean expression and $l$ is a logical variable of any type.

### 2.2 Semantics

The basic notion underlying the semantics of both the programming language and the assertion language is that of a many-sorted structure of the form

$$(dom(\text{Presburger}), \{\mathsf{true}, \mathsf{false}\}, dom(T_1), \ldots, dom(T_n), I)$$

where $T_i$ for $1 \leq i \leq n$, denotes a class type, array type or some abstract data type and $I$ denotes an interpretation $I$ of the non-logical function and predicate symbols (i.e. arithmetic and logical operations). The non-logical symbols include at least (i) variables declared in Var; (ii) for every class C, its fields declared in $F_{\mathrm{C}}$; (iii) for every class C a unary predicate $C$ of type Object $\to$ Boolean; and (iv) for each array type $T[\,]$ an access function $[\,]_{T[\,]}$ of type Presburger $\to (T[\,] \to T)$. The domains of different class types are assumed to be disjoint. There is no need for a separate sort for Object, semantically this set is simply the union of all the sorts for the class types (which includes array types). It is crucial here that the structure fixes the standard interpretation of both the types Presburger and Boolean, and the arithmetical and logical operations defined on them. The interpretation of the other sorts and operations are user-defined (i.e. not fixed).

We write $M(s)$ instead of $I(s)$ for the interpretation of the non-logical symbol $s$ and $M(T)$ for the sort $dom(T)$ in a structure $M$ of our language.

If $u$ is declared in Var as a variable of type $T$, it is interpreted as an individual of the sort $M(T)$. A field $x \in F_{\mathrm{C}}$ of type $T$ is interpreted as a unary function $M(\mathrm{C}) \to M(T)$. Array access functions $[\,]_{T[\,]}$ are interpreted as binary functions $M(\text{Presburger}) \to (M(T[\,]) \to M(T))$. Thus array indices can be seen as fields.

*Semantics of Expressions and Statements.* The meaning of an expression $e$ of type $T$ is a total function $[\![e]\!]$ that maps a structure $M$ to an individual of $M(T)$. This function is defined by induction on $e$. Here are the main cases:

- $[\![e_1.x]\!](M) = M(x)([\![e_1]\!](M))$.
- $[\![e_1[e_2]]\!](M) = M([\,]_{T[\,]})([\![e_2]\!](M))([\![e_1]\!](M))$
  where $e_1$ has the array type $T[\,]$ and $e_2$ has type Presburger.
- $[\![\mathrm{C}(e_1)]\!](M) = \mathsf{true}$ iff $[\![e_1]\!](M) \in M(\mathrm{C})$

As the meaning function of our semantics is total, some meaning is assigned to the expression $\mathsf{null}.x$. However, in the execution of programs their meaning is given operationally by executing the abort statement.

Statements in our language are deterministic and can fail (abort) or diverge. We define the meaning of a statement in terms of a small-step operational semantics, and use the (quite common) notation

$$\langle s, M\rangle \longrightarrow \langle s', M'\rangle$$

to express that executing $s$ in structure $M$, results after one step in the statement $s'$ and structure $M'$. We use $\longrightarrow^*$ for the reflexive transitive closure of this transition relation. We omit $s'$ if $s$ immediately terminates from $M$. Since calls can appear in statements, the definition of the above transition relation depends in general on the method declarations. Note that throughout execution, assignments to variables and fields change the structure in the interpretation of the

variables and fields respectively. The interpretation of the array access function changes due to assignments to subscripted variables. Moreover during executing, the sorts $dom(\mathrm{C}_i)$ containing instances of $\mathrm{C}_i$ are extended with new objects by object creations $u := \mathsf{new}$. Statements do not affect the sorts Presburger, Boolean, and the interpretation of the other non-logical symbols.

The meaning of normal assignments, conditional statements and while loops is defined in the standard way. Hence, we focus on the semantics of array creation. First define for each type a default value: $init_{\mathrm{Presburger}} = 0$, $init_{\mathrm{Boolean}} = \mathsf{false}$ and $init_{\mathrm{C}} = \mathsf{null}$. For the selection of a new object of class C we use a choice function $\nu$ on a structure $M$ and class C to get a fresh object $\nu(M, \mathrm{C})$ of class C which satisfies $\nu(M, \mathrm{C}) \notin M(T)$ for any type $T$ (in particular, $\nu(M, \mathrm{C}) \notin M(\mathrm{C})$). Clearly, without loss of generality we may assume that $\nu(M, \mathrm{C})$ only depends on $M(\mathrm{C})$ in the sense that this choice function preserves the deterministic nature of our core language (formally: $\nu(M, \mathrm{C}) = \nu(M', \mathrm{C})$ if $M(\mathrm{C}) = M(\mathrm{C})$). Nondeterministic (or random) selection of a fresh object would require reasoning semantically up to a notion of isomorphic models which would unnecessarily complicate proofs.

Let $u$ be of type $T[\,]$. The semantics of an array creation is modeled by:

$$\langle u := \mathsf{new}, M\rangle \longrightarrow M'$$

where $M'$ is changed from $M$ as follows: Let $o$ denote the object identity chosen by $\nu(M, T[\,])$, i.e, $o = \nu(M, T[\,])$ then

1. $M'(T[\,]) = M(T[\,]) \cup \{o\}$.
2. $M'([\,]_{T[\,]})(n)(o) = init_T$ for all $n \in M(\mathrm{Presburger})$.
3. $M'(u) = o$.

The second clause states that all array elements have initially their default value.

The operational semantics of a program is given by the one of its main statement, executed in the *initial structure* $M_0$. In $M_0$, for every class type C no objects other than $\mathsf{null}_{\mathrm{C}}$ exist, and all variables have their default value.

*Semantics of Assertions.* The semantics of assertions is defined by the usual Tarski truth definition. Interestingly, even though we allow quantification over array objects, all assertions are *first-order* formulas (interpreted over arbitrary structures obeying the first-order Presburger and Boolean algebra axioms, including non-standard interpretations). This is because of a subtle difference in meaning between modeling arrays as sequences (not first-order), or as pointers to sequences (first-order [17,12]): In case $s$ ranges over (finite) sequences $\exists s : s[0] = 0$ expresses that there exists a sequence $s$ of natural numbers, of which the first is 0. This sequence itself is not an element of the domain of a structure for our many-sorted dynamic logic language, but rather a sequence of elements of the domain Presburger. In this interpretation the above formula is valid. In this paper we model arrays as pointers to data-structures in the heap (e.g., structure) as in Java. If $a$ is a logical variable of type Presburger[ ] then $\exists a : a[0] = 0$ asserts the existence of an array object (an individual of the sort for Presburger[ ]) in which currently the first element is 0. This formula is not

valid, for it is false in all structures in which no such array exists. Note that the *extensionality* axiom $\forall a, b, n : a[n] = b[n] \rightarrow a = b$ for arrays is also not valid.

## 3 Inductive Assertion Networks

We extend Floyd's inductive assertion method to object-oriented programs. An inductive assertion network is a labelled transition system where transitions are labelled with conditional assignments of (local) variables. A labelled transition may fire if its (pre-)condition is satisfied. The corresponding assignment is executed and the state updated accordingly. Obviously control-flow graphs of imperative programs fall into the class of these transition systems. The labelled transition system is extended to an assertion network by assigning each state a (set of) assertions. An assertion network is called inductive if and only if whenever $M(\phi)$ holds for a structure (state) $M$ and the condition of a transition is satisfied, then the assertion $\phi'$ assigned to the resulting structure holds as well.

We extend Floyd's notion of an inductive assertion network to object-oriented programs: besides basic assignments, transitions can be labelled with object creations and assignments to fields and subscripted variables. This requires a corresponding extension for computing verification conditions, taking for example aliasing into account. Finally we need in general auxiliary variables to describe the object structures in the heap. Because, for instance, first-order logic itself cannot express reachability in linked lists (see Section 5).

As one main feature of our semantics is to model object creation as extension of the underlying structure's domain, the rule for deleting assignments to auxiliary variables as introduced in Owicki and Gries [14] for reasoning about shared variable concurrency is not sound anymore. Clearly we cannot remove the dynamic allocation of the auxiliary variable $u$ even if $u$ does not appear in the assertions (an assignment $u := \mathsf{new}$ in fact may validate an assertion $\exists l : \phi$, where the logical variable is of the same type as $u$). To obtain a complete inductive assertion method we allow method signatures extended with auxiliary formal parameters.

A basic assertion network of a program extended with auxiliary variables associates assertions with each (sub)statement of the program. A (finite) set of verification conditions for this annotated program is then generated fully automatically by means of the weakest precondition calculus defined in [1] extended with a substitution for the (dynamic) creation of arrays. The verification conditions of the pre- and postcondition of a method call are defined in the standard way in terms of the pre- and postcondition of the method body, modeling parameter passing by substitution.

## 4 Expressiveness

In this section we first investigate the expressiveness of auxiliary variables. This leads to the following main result: the set of reachable states at each control point of a general class of instrumented programs is expressible in a first-order

assertion language with equality, unbounded arrays and addition. This forms the basis for the completeness of our object-oriented inductive assertion method.

### 4.1 Instrumentation

Below we show how the computation history of instrumented programs can be stored in auxiliary variables in a canonical manner. The instrumentation must be faithful to the original program:

**Definition 1 (Faithful Instrumentation).** *Given a set of auxiliary variables, an instrumented program is* faithful *to the original program if neither the value of the normal (non-auxiliary) variables nor the termination behavior is affected by the instrumentation.*

Intuitively the instrumentation adds auxiliary array variables to the original program which record only the changes to the values of variables and fields of the program (including those of an array type). In comparison to storing the full state at each computation step, this allows for a fairly simple update mechanism for the auxiliary variables. Faithful instrumentations allow the expression of properties *of the original program* which cannot be expressed in first-order logic formulas directly. We now list the auxiliary variables, along with a description how they are set during the execution of instrumented programs, assuming a unique line number for each (sub)statement:

- A one-dimensional array variable $pc$ of Presburger[ ] to record the history of the program counter. The intention is that if $pc[i] = j$, then line $j$ was executed in the $i$-th step of the computation.
- A variable $|pc|$ of type Presburger containing the number of completed computation steps.
- For each variable $u$ of a type $T$ an array variable $u'$ with content type $T$, and an array variable $u''$ of type Boolean. If in the $i$-th step of the computation the value $v$ is assigned to $u$, then $u'[i] = v$, and $u''[i] = \mathsf{true}$. If the $i$-th step does not involve an assignment to variable $u$, we have $u'[i] = init_T$, where $T$ is the type of $u$, and $u''[i] = \mathsf{false}$, which is the default Boolean value.
- For each field $x$ an array $x'$ of pairs $< o, v >$ (where $o$ is an object identity and $v$ a value)[1], and an array $x''$ of Boolean. In analogy to the two arrays storing the changes to variables, these two arrays store the changes to the field. The extra object identity is needed to identify the object whose field was changed.
- For each array type $T$ occurring in the program, a one-dimensional array variable $Arr'_T$ of Boolean storing the computation steps in which the interpretation of an array object of type $T$ was changed, and a one-dimensional array $Arr_T$ of triples $< o, n, v >$ storing the new values of the changed element in that array ($o$ is an array object, $n$ an array index and $v$ a value).
- A method parameter $loc$ of type Presburger, which stores the line number on which the call was made.

[1] Such a type can be easily defined in our language as a class with two fields.

$$
\begin{array}{ll}
 & pc[|pc|] := j\,; \\
 & u'[|pc|] := e\,; \\
j: & u := e\,; \\
 & u''[|pc|] := \text{true}\,; \\
 & |pc| := |pc| + 1\,;
\end{array}
$$

**Fig. 1.** Instrumentation of the statement $u := e$ on line number $j$

$$
\begin{array}{ll}
 & pc[|pc|] := j\,; \\
 & x'[|pc|] :=< e, e' >\,; \\
j: & e.x := e'\,; \\
 & x''[|pc|] := \text{true}\,; \\
 & |pc| := |pc| + 1\,;
\end{array}
$$

**Fig. 2.** Instrumentation of the statement $e.x := e'$ on line number $j$

As two examples we show how the instrumentation of the basic assignment and method call is performed in Figures 1 and 2. The control structures are simply instrumented by updates to the variable $pc$ to record the flow of control. Additionally in a call we pass the line number as a parameter which is used upon return. Given a line number $j$, by $\text{next}(j)$ we denote the line number of the statement which will be executed in the next step of the computation.

To instrument a program with a main statement $s_{main}$ and method bodies $B_1, \ldots, B_n$, label first each program statement uniquely. Then apply the instrumentation given above to $s_{main}$ to obtain $s'_{main}$ and to each method body $B_i$ to obtain the statement $B'_i$. Next, define a statement init which creates new objects for the auxiliary array variables, and sets $|pc| := 0$. The final instrumented program is given by the main statement $init; s'_{main}$ and method bodies $B'_i$.

**Theorem 1.** *The above instrumentation is faithful to the original program.*

### 4.2 Weak Arithmetic Completeness

*Completeness* of basic inductive assertion networks has been proven in [6,11], provided that suitable intermediate assertions exist in the assertion language. Here we demonstrate how to find such assertions for the class of instrumented object-oriented programs as defined previously.

Recall that programs start executing in a fixed initial structure (see Section 2 on semantics of statements). Hence from a purely semantic viewpoint, the intermediate assertion at location $l$ can simply be chosen as the set of all structures reachable in $l$ from the initial structure. Such reachability predicates are reminiscent of the most general correctness formulae introduced by Gorelick in [8] to show completeness for a Hoare logic for recursive programs.

**Definition 2.** *Let $P$ be a program with statement $s$ on line number $l$. The reachability predicate $\mathbb{R}_l$ denotes the set of states $\{M | \langle P, M_0\rangle \longrightarrow^* \langle s; s', M\rangle\}$, where $M_0$ is a standard model (the initial structure, see Section 2), and $s'$ is the remainder of the program to be executed.*

It remains to show that our first-order assertion language which only assumes the standard interpretation of Presburger arithmetic is expressive enough to define the above reachability predicates *syntactically*. This is indeed the case for

instrumented programs. For such programs, the state-based encoding of the computation allows recovering the computation of the instrumented program in the assertion language without using a Gödel encoding (which relies on the presence of multiplication in the assertion language). Our results are not restricted to the specific instrumentation defined in the previous section. In general any faithful instrumentation which allows recovering the computation in the assertion language can be used.

We now describe how the computation of instrumented programs can be recovered in the assertion language. Given an *uninstrumented* program $P$ and a computation step $i$ (i.e. a number), define an assertion $\mathrm{COMP}_{P,i}$ which completely describes the state change induced by the $i$-th computation step in the instrumented version of $P$. For an assignment $u := e$ with line number $j$ we define $\mathrm{COMP}_{P,i}$ by

$$pc[i] = j \rightarrow (pc[i+1] = \mathrm{next}(j) \wedge u'[i+1] = \mathrm{Val}(e,i) \wedge u''[i+1] = \mathsf{true} \wedge \mathrm{nochange}_j(i))$$

Assignments to fields or subscripted variables can be handled similarly to the variable assignment above. The expression $\mathrm{Val}(e,i)$ stands for the value of the expression $e$ after the $i$-th computation step. The interesting case is when $e$ is a (subscripted) variable or field. We show the case when $e$ is a variable $u$ of type $T$:

$$\begin{aligned}\mathrm{Val}(u,i) = z \leftrightarrow\ & (init_T = z \wedge \forall n \leq |pc| : u''[n] = \mathsf{false}) \vee \\ & (\exists n \leq i : u''[n] \wedge u'[n] = z \wedge \forall k(n < k < |pc|) : u''[n] = \mathsf{false})\end{aligned}$$

The first disjunct asserts that $\mathrm{Val}(u,i)$ is determined by the last assignment to $u$ which occurred before or on computation step $i$. The second disjunct asserts that if there was no such assignment, the variable has retained its initial value. The value of a Boolean condition in a given computation step can also be determined easily using the Val function.

The predicate $\mathrm{nochange}_j(i)$ asserts that only the auxiliary variables representing the l.h.s. of the assignment with line number $j$ are affected by the $i$-th computation step, i.e., all the other auxiliary variables indicate at the $i$-th step that their represented program variables have not changed. For example, for a program variable $u$ of type $T$ this is expressed simply by the assertion $u'[i] = init_T \wedge u''[i] = \mathsf{false}$; and for arrays of type $T$ this is expressed by $\mathrm{Arr}_T[i] = \mathsf{null} \wedge \mathrm{Arr}'_T[i] = \mathsf{false}$. Note that we make use of the initial default values of the auxiliary variables. We denote by $\mathrm{nochange}(i)$ that *all* auxiliary variables indicate at the $i$-th step that their program variables have not changed.

To express the reachability predicate at a location $l$, one must further assert that the current values of the normal (non-auxiliary) variables, fields and array access function are those stored in the auxiliary variables at $l$ (but before the statement at $l$ is executed). Let us abbreviate such an assertion by $\mathrm{aux}(l)$. To see how $\mathrm{aux}(l)$ can be defined in our assertion language, note that for a variable $u$ it simply reduces to the assertion $u = \mathrm{Val}(u,l)$ as defined above. For arrays, a universal quantification ranging over all array indices is necessary. For instance,

$\forall n : a[n] = Val(a[n], l)$ characterizes the full contents of the array $a$ at location $l$. Thus aux($l$) can now be expressed as the (finite) conjunction of such assertions for all variables, fields and arrays. The reachability predicates of instrumented programs can now be readily defined:

**Theorem 2.** *Let $P$ be an arbitrary program, and let $P*$ be the instrumented version of $P$. Then the reachability predicate $\mathbb{R}_l$ of $P*$ is defined by the assertion:*

$$aux(l) \wedge pc[|pc|] = l \wedge \forall 0 \leq i < |pc| : COMP_{P,i} \wedge \text{nochange}$$

*where* nochange *stands for* $\text{nochange}(0) \wedge \forall i > |pc| : pc[i] = 0 \wedge \text{nochange}(i)$.

The next theorem now follows by construction of the reachability predicates:

**Theorem 3.** *Let $P*$ be a program instrumented with auxiliary variables as described above, and let $P'$ be its annotation with at each location $l$ an assertion which defines $\mathbb{R}_l$. A partial correctness formula $\{p\}P^*\{q\}$ is true in the initial model if and only if all generated verification conditions for the assertion network $\{p\}P'\{q\}$ are true.*

The above theorem states that we can derive any true correctness formula in first-order logic. For proving the generated verification conditions we can take as axioms all first-order sentences true in the initial model[2] (and use any off-the-shelve theorem prover for first-order logic). This normally results in ineffective proof systems, since by Gödels incompleteness theorem the axioms are typically not recursively enumerable. However by excluding multiplication from the assertion language, it follows from Presburgers result that the set arithmetical axioms is recursive. Thus if the other types are also interpreted in such a way that their first-order theory is recursive, the truth of the generated verification conditions is decidable.

## 5 Example: Expressing and Verifying Reachability

In the previous section the reachability predicates were defined *uniformly*. In this section we show how to reduce the complexity of the instrumentation significantly by exploiting the structure of a given program and property to prove.

Consider a queue data structure where items of type Presburger can be added to the beginning of the queue and removed from the end of the queue. The queue is backed up by a linked list using a *next* field which points to the next item in the queue. The public interface of such a queue contains of (i) two global variables pointing to its *first* and *last* element, (ii) an *enqueue*($v$) method which adds $v$ to the beginning of the queue and (iii) a *dequeue* method

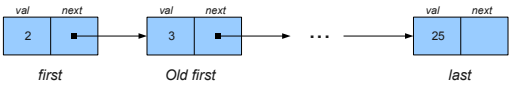



**Fig. 3.** Queue resulting from $first.enqueue(2)$

[2] This set is sometimes called the first-order theory of a structure.

which removes the last item from the queue. Figure 3 visualises the result of the method call $first.enqueue(2)$, where $first$ initially (i.e. before executing the call) points to an item with value 3, and $last$ points to an item with value 25. Let $\text{reach}(f, l, a, n)$ abbreviate the assertion

$$n \geq 0 \wedge a[0] = f \neq null \neq a[n] = l \wedge l.next = null \\ \wedge \forall j(0 \leq j < n) : a[j] \neq null \wedge a[j].next = a[j+1]$$

Intuitively this assertion specifies that an array $a$ stores the linked list, and that $l$ is reachable from $f$ by repeated dereferencing of field $next$.

Using an auxiliary array $b$ to store the new linked list, we can now express that if this reachability property was initially true then it holds again after executing $enqueue(v)$:

$\{\exists a, n : \text{reach}(first, last, a, n)\}$
z := new; z.next := first ; z.val := v; first := z
b := new; b[0] := first ; i := 0;
**while** b[i] $\neq$ last **do** b[i+1] := b[i].next; i := i+1 **od**;
$\{\exists a, n : \text{reach}(first, last, a, n)\}$

Strictly speaking this is a property of this particular instrumented version of $enqueue(v)$: the original version does not even have the auxiliary array $b$. However as the above instrumentation is faithful to the original version, it follows that $last$ is reachable from $first$ in the original program (by repeated dereferencing of $next$), for otherwise $\exists a, n : \text{reach}(first, last, a, n)$ would not hold in *any* faithful instrumentation.

As our semantics are based on *abstract object creation* (not yet created objects play no role in structures, and are not referable in assertions), a corresponding proof theory is needed to verify the above instrumented program. Based on [1], we have extended their approach to support dynamically created arrays. The new rules are fully implemented in a special version of KeY, available at `http://keyaoc.hats-project.eu`. Only one interaction with KeY is required to verify the specified reachability property for method $enqueue(v)$, namely the provision of a loop invariant, everything else was fully automatic.

## 6 Conclusions

*Scope of the Programming Language.* We want to stress that our core language contains all necessary primitive constructs from which more intricate features can be handled by a completely mechanical translation. The features to which this transformational approach applies include failures and *bounded* arrays. Inheritance and dynamic binding have been addressed in [3]. These transformations allow us to treat object creation orthogonally to such features, and thereby indicates our approach scales up to modern languages.

*Expressibility of Weakest Precondition.* General results of Olderog [13] show there is a certain symmetry between the expressibility of strongest postconditions and weakest preconditions. To prove the result, Olderog makes a constant domain assumption which requires a different modeling of object creation than in our case where we support *abstract* object creation. Hence one cannot in general refer to the final values of the variables in assertions evaluated in an initial state. Consequently Olderog's result does not apply here: object creation breaks the symmetry between strongest postconditions and weakest preconditions.

*Recursive Assertions.* Apt et al. [2] prove that even for recursive preconditions and postconditions, the intermediate assertions cannot be chosen recursively for general programs, but only for a class of suitably instrumented programs. Our assertions defining reachability are currently not recursive due to unbounded quantification over array indices (see Section 4.2). However, if we restrict to bounded arrays then we only need *bounded* quantification in the expression of the reachability predicates. The trade-off is a significantly more complicated instrumentation as at each computation step a reallocation of the auxiliary (array) variables becomes necessary.

## References

1. Ahrendt, W., de Boer, F.S., Grabe, I.: Abstract Object Creation in Dynamic Logic. In: Cavalcanti, A., Dams, D. (eds.) FM 2009. LNCS, vol. 5850, pp. 612–627. Springer, Heidelberg (2009)
2. Apt, K.R., Bergstra, J.A., Meertens, L.G.L.T.: Recursive assertions are not enough - or are they? Theor. Comput. Sci. 8, 73–87 (1979)
3. Apt, K.R., de Boer, F.S., Olderog, E.-R., de Gouw, S.: Verification of object-oriented programs: A transformational approach. JCSS 78(3), 823–852 (2012)
4. Beckert, B., Hähnle, R., Schmitt, P. (eds.): Verification of Object-Oriented Software. LNCS (LNAI), vol. 4334. Springer, Heidelberg (2007)
5. Cook, S.A.: Soundness and completeness of an axiom system for program verification. SIAM J. Comput. 7(1), 70–90 (1978)
6. de Bakker, J., Meertens, L.: On the completeness of the inductive assertion method. Journal of Computer and System Sciences 11(3), 323–357 (1975)
7. Floyd, R.W.: Assigning meanings to programs. In: Schwartz, J.T. (ed.) Mathematical Aspects of Computer Science. Proc. of Symposia in Applied Mathematics, vol. 19, pp. 19–32. AMS (1967)
8. Gorelick, G.: A complete axiomatic system for proving assertions about recursive and non-recursive programs. Technical Report 75, Univ. of Toronto (1975)
9. Harel, D.: Arithmetical Completeness in Logics of Programs. In: Ausiello, G., Böhm, C. (eds.) ICALP 1978. LNCS, vol. 62, pp. 268–288. Springer, Heidelberg (1978)
10. Lindström, P.: On extensions of elementary logic. Theoria 35(1), 1–11 (1969)
11. Manna, Z.: Mathematical Theory of Partial Correctness. In: Engeler, E. (ed.) Symposium on Semantics of Algorithmic Languages. Lecture Notes in Mathematics, vol. 188, pp. 252–269. Springer (1971)
12. McCarthy, J.: Towards a mathematical science of computation. In: IFIP, pp. 21–28. North-Holland (1962)

13. Olderog, E.-R.: On the notion of expressiveness and the rule of adaption. Theor. Comput. Sci. 24, 337–347 (1983)
14. Owicki, S.S., Gries, D.: An axiomatic proof technique for parallel programs i. Acta Inf. 6, 319–340 (1976)
15. Pierik, C.: Validation Techniques for Object-Oriented Proof Outlines. PhD thesis, Universiteit Utrecht (2006)
16. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. Comptes Rendus du I congrs de Mathmaticiens des Pays Slaves, 92–101 (1929)
17. Suzuki, N., Jefferson, D.: Verification decidability of presburger array programs. J. ACM 27(1), 191–205 (1980)
18. Tucker, J., Zucker, J.: Program correctness over abstract data types, with error-state semantics. Elsevier Science Inc. (1988)

# Generalising and Unifying SLUR and Unit-Refutation Completeness

Matthew Gwynne[1] and Oliver Kullmann[2]

[1] Computer Science Department, Swansea University
http://cs.swan.ac.uk/~csmg/
[2] Computer Science Department, Swansea University
http://cs.swan.ac.uk/~csoliver

**Abstract.** The class $\mathcal{SLUR}$ (*Single Lookahead Unit Resolution*) was introduced in [22] as an umbrella class for efficient SAT solving. [7,2] extended this class in various ways to hierarchies covering all of CNF (all clause-sets). We introduce a hierarchy $\mathcal{SLUR}_k$ which we argue is the natural "limit" of such approaches.

The second source for our investigations is the class $\mathcal{UC}$ of *unit-refutation complete clause-sets* introduced in [10]. Via the theory of (tree-resolution based) "hardness" of clause-sets as developed in [19,20,1] we obtain a natural generalisation $\mathcal{UC}_k$, containing those clause-sets which are "unit-refutation complete of level $k$", which is the same as having hardness at most $k$. Utilising the strong connections to (tree-)resolution complexity and (nested) input resolution, we develop fundamental methods for the determination of hardness (the level $k$ in $\mathcal{UC}_k$).

A fundamental insight now is that $\mathcal{SLUR}_k = \mathcal{UC}_k$ holds for all $k$. We can thus exploit both streams of intuitions and methods for the investigations of these hierarchies. As an application we can easily show that the hierarchies from [7,2] are strongly subsumed by $\mathcal{SLUR}_k$. We conclude with a discussion of open problems and future directions.

## 1 Introduction

The boolean satisfiability problem, SAT for short, in its core version is the problem of deciding satisfiability of a conjunctive normal form (clause-set); see the handbook [4] for further information. We bring together two previously unconnected streams of research:

**SLUR** The search for classes of clause-sets for which one can decide satisfiability in polynomial time. Especially we consider the SLUR algorithm and class.

**UC** The search for target classes of clause-sets with good knowledge compilation properties, i.e., where the clausal entailment problem can be decided quickly. Especially we consider the class UC of unit-refutation complete clause-sets.

In the year 1995 in [22] the SLUR algorithm was introduced, a simple *incomplete* non-deterministic SAT-decision algorithm, together with the class $\mathcal{SLUR}$ of inputs where it *always* succeeds. $\mathcal{SLUR}$ contains various classes with polynomial-time SAT decision, where previously only rather complicated algorithms were

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 220–232, 2013.

known. The natural question arises, whether $\mathcal{SLUR}$ can be turned into a hierarchy, covering in the limit all clause-sets. In [7,2] the authors finally proved that membership decision of $\mathcal{SLUR}$ is coNP-complete, and presented three hierarchies, $\mathcal{SLUR}(k), \mathcal{SLUR}*(k)$ and CANON$(k)$. It still seemed that none of these hierarchies is the final answer, though they all introduce a certain natural intuition. We now present what seems the natural "limit hierarchy", which we call $\mathcal{SLUR}_k$, and which unifies the two basic intuitions embodied in $\mathcal{SLUR}(k), \mathcal{SLUR}*(k)$ on the one hand and CANON$(k)$ on the other hand.

In the year 1994 in [10] the class $\mathcal{UC}$ was introduced, containing clause-sets $F$ such that *clause entailment*, that is, whether $F \models C$ holds (clause $C$ follows logically from $F$, i.e., $C$ is an implicate of $F$), can be decided by unit-clause propagation. A second development is important here, namely the development of the notion of "hardness" $\mathrm{hd}(F)$ in [19,20,1]. As we show in Theorem 1, $\mathrm{hd}(F) \le k$ is equivalent to the property of $F$, that all implicates of $F$ (i.e., all clauses $C$ with $F \models C$) can be derived by $k$-times nested input resolution from $F$, a generalisation of input resolution as introduced and studied in [19,20]. We obtain that $\mathcal{UC}$ is precisely the class of clause-sets $F$ with $\mathrm{hd}(F) \le 1$ ! It is then natural to define the hierarchy $\mathcal{UC}_k$ via the property $\mathrm{hd}(F) \le k$. The hierarchy CANON$(k)$ is based on resolution trees of height at most $k$, which is a special case of $k$-times nested input resolution, and so we have CANON$(k) \subset \mathcal{UC}_k$.

The hardness-notion provides the *proof-theoretic side* of our investigations. The *algorithmic side* is given by the reductions $\mathrm{r}_k(F)$, which perform certain forced assignments, with $\mathrm{r}_1$ being UCP (unit-clause propagation) as the most well-known case. For unsatisfiable $F$ the hardness $\mathrm{hd}(F)$ is equal to the minimal $k$ such that $\mathrm{r}_k(F)$ detects unsatisfiability of $F$. This yields the basic observation $\mathcal{UC} \subseteq \mathcal{SLUR}$ — and actually we have $\mathcal{UC} = \mathcal{SLUR}$ !

So by replacing the use of $\mathrm{r}_1$ in the SLUR algorithm by $\mathrm{r}_k$ (using a refined, semantic, analysis) we obtain a natural hierarchy $\mathcal{SLUR}_k$, which includes the previous SLUR-hierarchies $\mathcal{SLUR}(k)$ and $\mathcal{SLUR}*(k)$, and where we have $\mathcal{SLUR}_k = \mathcal{UC}_k$. This equality of these two hierarchies is our argument that we have found the "limit hierarchy" for SLUR.

The underlying report of this paper is [15], where all missing proofs can be found, and where examples are provided. Also the anticipated main application of the classes $\mathcal{UC}_k$ as target classes for SAT translation is discussed there.

## 2 Preliminaries

We follow the general notions and notations as outlined in [18]. Based on an infinite set $\mathcal{VA}$ of variables, we form the set $\mathcal{LIT} := \mathcal{VA} \cup \overline{\mathcal{VA}}$ of positive and negative literals, using complementation. A clause $C \subset \mathcal{LIT}$ is a finite set of literals without clashes, i.e., $C \cap \overline{C} = \emptyset$, where for $L \subseteq \mathcal{LIT}$ we set $\overline{L} := \{\overline{x} : x \in L\}$. The set of all clauses is denoted by $\mathcal{CL}$. A clause-set $F \subset \mathcal{CL}$ is a finite set of clauses, the set of all clause-sets is $\mathcal{CLS}$. A special clause is the empty clause $\bot := \emptyset \in \mathcal{CL}$, and a special clause-set is the empty clause-set $\top := \emptyset \in \mathcal{CLS}$. By $\mathrm{lit}(F) := \bigcup F \cup \overline{\bigcup F}$ we denote the set of literals occurring at

least in one polarity in $F$. We use $\mathrm{var} : \mathcal{LIT} \to \mathcal{VA}$ for the underlying variable of a literal, $\mathrm{var}(C) := \{\mathrm{var}(x) : x \in C\} \subset \mathcal{VA}$ for the variables in a clause, and $\mathrm{var}(F) := \bigcup_{C \in F} \mathrm{var}(C)$ for the variables in a clause-set. The number of variables in a clause-set is $n(F) := |\mathrm{var}(F)| \in \mathbb{N}_0$, the number of clauses is $c(F) := |F| \in \mathbb{N}_0$, and the number of literal occurrences is $\ell(F) := \sum_{C \in F} |C| \in \mathbb{N}_0$. The set of Horn clause-sets is $\mathcal{HO} \subset \mathcal{CLS}$, where every clause contains at most one positive literal. A partial assignment $\varphi : V \to \{0,1\}$ maps $V \subset \mathcal{VA}$ to truth-values, the set of all partial assignments is $\mathcal{PASS}$. We construct partial assignments via $\langle v_1 \to \varepsilon_1, \ldots, v_n \to \varepsilon_n \rangle \in \mathcal{PASS}$ for $v_i \in \mathcal{VA}$ and $\varepsilon_i \in \{0,1\}$. We use $\mathrm{var}(\varphi) := V$. For a partial assignment $\varphi \in \mathcal{PASS}$ and a clause-set $F \in \mathcal{CLS}$ the application of $\varphi$ to $F$ is denoted by $\varphi * F \in \mathcal{CLS}$, which results from $F$ by removing all satisfied clauses (containing at least one satisfied literal), and removing all falsified literals from the remaining clauses. A clause-set $F$ is satisfiable (i.e., $F \in \mathcal{SAT} \subset \mathcal{CLS}$) if there exists a partial assignment $\varphi$ with $\varphi * F = \top$, otherwise $F$ is unsatisfiable (i.e., $F \in \mathcal{USAT} := \mathcal{CLS} \setminus \mathcal{SAT}$). Two clauses $C, D \in \mathcal{CL}$ are resolvable if they clash in exactly one literal $x$, that is, $C \cap \overline{D} = \{x\}$, in which case their resolvent is $(C \cup D) \setminus \{x, \overline{x}\}$ (with resolution literal $x$). A resolution tree is a binary tree formed by the resolution operation. We write $\boldsymbol{T : F \vdash C}$ if $T$ is a resolution tree with axioms (the clauses at the leaves) all in $F$ and with derived clause (at the root) $C$. By $\mathbf{Comp^*_R(F)}$ for unsatisfiable $F$ the minimum number of leaves in a tree-resolution-refutation $T : F \vdash \bot$ is denoted. Finally, by $\mathrm{r}_1 : \mathcal{CLS} \to \mathcal{CLS}$ unit-clause propagation is denoted, that is applying $F \rightsquigarrow \langle x \to 1 \rangle * F$ as long as there are unit-clauses $\{x\} \in F$, and reducing $F \rightsquigarrow \{\bot\}$ in case of $\bot \in F$.

## 3 The SLUR Class and Extensions

The SLUR-algorithm and the class $\mathcal{SLUR} \subset \mathcal{CLS}$ have been introduced in [22]. For input $F \in \mathcal{CLS}$ we get an incomplete polynomial-time SAT algorithm, which either returns "SAT" or "UNSAT" (in both cases correctly) or gives up. This algorithm is non-deterministic, and $\mathcal{SLUR}$ is the class of clause-sets where it never gives up (whatever the choices are). Thus SAT-decision for $F \in \mathcal{SLUR}$ can be done in polynomial time, and due to an observation attributed to Truemper in [11], the SLUR-algorithm can actually be implemented such that it runs in linear time. Decision of membership, that is whether $F \in \mathcal{SLUR}$ holds, by definition is in coNP, but only in [7] it was finally shown that this decision problem is coNP-complete. The original motivation was that $\mathcal{SLUR}$ contains several other classes, including renamable Horn, extended Horn, hidden extended Horn, simple extended Horn and CC-balanced clause-sets, where for each class it was known that the SAT problem is solvable in polynomial time, but with in some cases rather complicated proofs, while it is trivial to see that the SLUR-algorithm runs in polynomial time. In [11,12] probabilistic properties of $\mathcal{SLUR}$ have been investigated. In this section we first give a semantic definition of $\mathcal{SLUR}$ in Subsection 3.1. In a nutshell, $\mathcal{SLUR}$ is the class of clause-sets where either UCP (unit-clause propagation aka $\mathrm{r}_1$) creates the empty clause, or where otherwise iteratively making assignments followed by UCP will always yield a satisfying assignment, given that these transitions do not obviously create unsatisfiable

results, i.e., do not create the empty clause. In order to understand this definition clearly, we present a precise mathematical (non-algorithmic) definition, based on the transition relation $F \xrightarrow{\text{SLUR}} F'$ (Definition 2), which represents one non-deterministic step of the SLUR algorithm: If $\mathrm{r}_1$ on input $F \in \mathcal{CLS}$ does not determine unsatisfiability (in which case we have $F \in \mathcal{SLUR}$), then $F \in \mathcal{SLUR}$ iff $\top$ can be reached by this transition relation, while everything else reachable from $F$ is not an end-point of this transition relation. In [7,2] recently three approaches towards generalising $\mathcal{SLUR}$ have been considered, and we discuss them in Subsection 3.2. Our generalisation, called $\mathcal{SLUR}_k$, which we see as the natural completion of these approaches, will be presented in Section 6.

### 3.1 SLUR

The idea of the SLUR-algorithm ("Single Lookahead Unit Resolution") for input $F \in \mathcal{CLS}$ is as follows: First run UCP, that is, reduce $F \rightsquigarrow \mathrm{r}_1(F)$. If now $\bot \in F$ then we determined unsatisfiable. If not, then the algorithm guesses a satisfying assignment for $F$, by repeated transitions $F \xrightarrow{\text{SLUR}} F'$, where $F'$ is obtained by assigning one variable and then performing UCP. The "lookahead" means that for $F' = \{\bot\}$ this transition is not performed. The algorithm might find a satisfying assignment in this way, or it gets stuck, in which case it "gives up". The SLUR class is defined as the class of clause-sets where this algorithm never gives up. The precise details are as follows. First we define the underlying transition relation (one non-failing transition from $F$ to $F'$):

**Definition 1.** *For clause-sets $F, F' \in \mathcal{CLS}$ the relation $\boldsymbol{F \xrightarrow{SLUR} F'}$ holds if there is $x \in \mathrm{lit}(F)$ such that $F' = \mathrm{r}_1(\langle x \to 1\rangle * F)$ and $F' \neq \{\bot\}$. The transitive-reflexive closure is denoted by $\boldsymbol{F \xrightarrow{SLUR}_* F'}$.*

Via the transition-relation $F \xrightarrow{\text{SLUR}} F'$ we can now easily define the class $\mathcal{SLUR}$, which will find a natural generalisation in Definition 11 to $\mathcal{SLUR}_k$ for $k \in \mathbb{N}_0$:

**Definition 2.** *The set of reduced clause-sets reachable from $F \in \mathcal{CLS}$ is denoted by* $\mathbf{slur}(\boldsymbol{F}) := \{F' \in \mathcal{CLS} \mid F \xrightarrow{SLUR}_* F' \wedge \neg\exists\, F'' \in \mathcal{CLS} : F' \xrightarrow{SLUR} F''\}$. *The class of all clause-sets which are either identified by UCP to be unsatisfiable, or where by SLUR-reduction always a satisfying assignment is found, is denoted by* $\boldsymbol{\mathcal{SLUR}} := \{F \in \mathcal{CLS} : \mathrm{r}_1(F) \neq \{\bot\} \Rightarrow \mathrm{slur}(F) = \{\top\}\}$.

### 3.2 Previous Approaches for SLUR Hierarchies

In [7,2] three hierarchies $\mathcal{SLUR}(k), \mathcal{SLUR}*(k)$ $(k \in \mathbb{N})$ and $\mathrm{CANON}(k)$ $(k \in \mathbb{N}_0)$ have been introduced. In Section 4 of [2] it is shown that $\mathcal{SLUR}(k) \subset \mathcal{SLUR}*(k)$ for all $k \in \mathbb{N}$ and so we restrict our attention to $\mathcal{SLUR}*(k)$ and $\mathrm{CANON}(k)$. $\mathrm{CANON}(k)$ is defined to be the set of clause-sets $F$ such that every prime implicate of $F$ can be derived from $F$ by a resolution tree of height at

most $k$. Note that basically by definition (using stability of resolution proofs under application of partial assignments) we get that each CANON($k$) is stable under application of partial assignments and under variable-disjoint union. The $\mathcal{SLUR}*(k)$ hierarchy is derived in [2] from the $\mathcal{SLUR}$ class by extending the reduction $r_1$. We provide an alternative formalisation here, in the same manner as in Section 3.1. The main question is the transition relation $F \rightsquigarrow F'$. The $\mathcal{SLUR}*(k)$-hierarchy provides stronger and stronger witnesses that $F'$ might be satisfiable, by longer and longer assignments (making "$k$ decisions") not yielding the empty clause:

**Definition 3.** *That partial assignment $\varphi \in \mathcal{PASS}$ **makes** $k$ **decisions** for some $k \in \mathbb{N}_0$ w.r.t. $F \in \mathcal{CLS}$ is defined recursively as follows: For $k = 0$ this relation holds if $\varphi * F = \mathrm{r}_1(F)$, while for $k > 0$ this relation holds if either there is $k' < k$ such that $\varphi$ makes $k'$ decision w.r.t. $F$ and $\varphi * F = \top$, or there exists $x \in \mathrm{lit}(F)$ and a partial assignment $\varphi'$ making $k-1$ decision for $\mathrm{r}_1(\langle x \to 1\rangle * F)$, and where $\varphi * F = \varphi' * \mathrm{r}_1(\langle x \to 1\rangle * F)$. Now $\boldsymbol{F \xrightarrow{SLUR*k} F'}$ for $k \geq 1$ by definition holds if there is a partial assignment $\varphi$ making $k$ decision w.r.t. $F$ with $F' = \varphi * F$, where $F' \neq \{\bot\}$. The reflexive-transitive closure is $\xrightarrow{SLUR*k}_*$.*

Finally we can define the hierarchy:

$$\begin{aligned}\mathbf{slur}*(\boldsymbol{k})(\boldsymbol{F}) &:= \{F' \in \mathcal{CLS} \mid F \xrightarrow{\mathrm{SLUR}*k}_* F' \wedge \neg\exists\, F'' : F' \xrightarrow{\mathrm{SLUR}*k} F''\}\\ \boldsymbol{\mathcal{SLUR}}*(\boldsymbol{k}) &:= \{F \in \mathcal{CLS} : \mathrm{slur}*(k)(F) \neq \{F\} \Rightarrow \mathrm{slur}*(k)(F) = \{\top\}\}.\end{aligned}$$

The unsatisfiable elements of $\mathcal{SLUR}*(k)$ are those $F \neq \top$ with $\mathrm{slur}*(k)(F) = \{F\}$. By definition each $\mathcal{SLUR}*(k)$ is stable under application of partial assignments, but not stable under variable-disjoint union, since the number of decision variables is bounded by $k$ (in Lemma 10 we will see that our hierarchy is stable under variable-disjoint union, which is natural since it strengthens the CANON($k$)-hierarchy).

## 4 Generalised Unit-Clause Propagation

In this section we review the approximations of forced assignments, as computed by the hierarchy of reductions $\mathrm{r}_k : \mathcal{CLS} \to \mathcal{CLS}$ from [19,20] for $k \in \mathbb{N}_0$. For further discussions of these reductions, in the context of SAT decision and in their relations to various consistency and width-related notions, see [19,20] and Section 3 in [21]. Fundamental is the notion of a **forced literal** of a clause-set, which are literals which must be set to true in order to satisfy the clause-set. If $x$ is a forced literal for $F$, then the **forced assignment** $\langle x \to 1\rangle * F$ yields a satisfiability-equivalent clause-set. We denote by $\mathbf{r_\infty}(\boldsymbol{F}) \in \mathcal{CLS}$ the result of applying all forced assignments to $F$. Note that $F$ is unsatisfiable iff $\mathrm{r}_\infty(F) = \{\bot\}$. We now present the hierarchy $\mathrm{r}_k : \mathcal{CLS} \to \mathcal{CLS}$, $k \in \mathbb{N}_0$, of reductions ([19]), which achieves approximating $\mathrm{r}_\infty$ by poly-time computable functions.

**Definition 4 ([19]).** *The maps* $\mathbf{r_k} : \mathcal{CLS} \to \mathcal{CLS}$ *for* $k \in \mathbb{N}_0$ *are defined as follows (for* $F \in \mathcal{CLS}$*):*

$$\mathrm{r}_0(F) := \begin{cases} \{\bot\} & \text{if } \bot \in F \\ F & \text{otherwise} \end{cases}$$

$$\mathrm{r}_{k+1}(F) := \begin{cases} \mathrm{r}_{k+1}(\langle x \to 1\rangle * F) & \text{if } \exists\, x \in \mathrm{lit}(F) : \mathrm{r}_k(\langle x \to 0\rangle * F) = \{\bot\} \\ F & \text{otherwise} \end{cases}.$$

$\mathrm{r}_1$ is unit-clause propagation, $\mathrm{r}_2$ is (full) failed literal elimination. In general we call $\mathrm{r}_k$ **generalised unit-clause-propagation of level** $k$. In [19] one finds the following basic observations proven (for $k \in \mathbb{N}_0$ and $F \in \mathcal{CLS}$):

- $\mathrm{r}_k : \mathcal{CLS} \to \mathcal{CLS}$ is well-defined (does not depend on the choices involved).
- $\mathrm{r}_k$ applies only forced assignments.
- $\mathrm{r}_k(F)$ is computable in time $O(\ell(F) \cdot n(F)^{2(k-1)})$ and linear space.

**Definition 5 ([19,20]).** *For* $k \in \mathbb{N}_0$*, clause-sets* $F$ *and clauses* $C$ *the relation* $\boldsymbol{F \models_k C}$ *holds if* $\mathrm{r}_k(\varphi_C * F) = \{\bot\}$*, where* $\varphi_C := \langle x \to 0 : x \in C\rangle$*.*

$F \models_1 C$ iff some subclause of $C$ follows from $F$ via input resolution. In [19] the *levelled height* "$h(T)$" of branching trees $T$ has been introduced, which was further generalised in [20]. It handles satisfiable as well as unsatisfiable clause-sets. Here we will only use the unsatisfiable case. Then this measure reduces to a well-known measure which only considers the structure of the tree. [1] used the term "Horton-Strahler number", which is the oldest source (from 1945).

**Definition 6.** *The* ***Horton-Strahler*** *number* $\mathbf{hs(T)} \in \mathbb{N}_0$ *for a resolution tree* $T$ *is defined as* $\mathrm{hs}(T) := 0$*, if* $T$ *is trivial, while otherwise we have two subtrees* $T_1, T_2$*, and we set* $\mathrm{hs}(T) := \max(\mathrm{hs}(T_1), \mathrm{hs}(T_2))$ *if* $\mathrm{hs}(T_1) \neq \mathrm{hs}(T_2)$*, while in case of* $\mathrm{hs}(T_1) = \mathrm{hs}(T_2)$ *we set* $\mathrm{hs}(T) := \max(\mathrm{hs}(T_1), \mathrm{hs}(T_2)) + 1$*.*

See Sections 4.2, 4.3 in [19] for various characterisations of $\mathrm{hs}(T)$. In [19], Chapter 7 (generalised in [20], Chapter 5), *generalised input resolution* was introduced:

**Definition 7 ([19,20]).** *For a clause-set* $F$ *and a clause* $C$ *the relation* $\boldsymbol{F \vdash_k C}$ *(*$C$ *can be derived from* $F$ *by* $k$***-times nested input resolution****) holds if there exists a resolution tree* $T$ *and* $C' \subseteq C$ *with* $T : F \vdash C'$ *and* $\mathrm{hs}(T) \leq k$*.*

By Parts 1 and 2 of Theorem 7.5 in [19], generalised in Corollary 5.12 in [20]:

**Lemma 1 ([19,20]).** *For clause-sets* $F$*, clauses* $C$ *and* $k \in \mathbb{N}_0$ *we have* $F \models_k C$ *if and only if* $F \vdash_k C$*.*

## 5 Hardness

This section is devoted to the discussion of $\mathrm{hd} : \mathcal{CLS} \to \mathbb{N}_0$. It is the central concept of the paper, from which the hierarchy $\mathcal{UC}_k$ is derived (Definition 10). The basic idea is to start with some measurement $h : \mathcal{USAT} \to \mathbb{N}_0$ of "the

complexity" of unsatisfiable $F$. This measure is extended to arbitrary $F \in \mathcal{CLS}$ by maximising over all "sub-instances" of $F$, that is, over all unsatisfiable $\varphi * F$ for (arbitrary) partial assignments $\varphi$. A first guess for $h : \mathcal{USAT} \to \mathbb{N}_0$ is to take something like the logarithm of the tree-resolution complexity of $F$. However this measure is too fine-grained, and doesn't yield a hierarchy like $\mathcal{UC}_k$. Another approach is algorithmical, measuring how far $F$ is from being refutable by unit-clause propagation. As shown in [19,20], actually these two lines of thought can be brought together by the hardness measure $\mathrm{hd} : \mathcal{USAT} \to \mathbb{N}_0$.

**Definition 8 ([19,20]).** *The* ***hardness*** $\mathrm{hd}(F)$ *of an unsatisfiable* $F \in \mathcal{CLS}$ *is the minimal* $k \in \mathbb{N}_0$ *such that* $\mathrm{r}_k(F) = \{\bot\}$.

As shown in [19], $\mathrm{hd}(F)+1$ is precisely the clause-space complexity of $F$ regarding tree-resolution. From [16] we gain the insight that for $F \in \mathcal{USAT}$ holds $\mathrm{hd}(F) \leq 1$ iff there exists $F' \subseteq F$ which is an unsatisfiable renamable Horn clause-set. By Theorem 7.8 (and Corollary 7.9) in [19] (or, more generally, Theorem 5.14 in [20]) we have for $F \in \mathcal{USAT}$ that $2^{\mathrm{hd}(F)} \leq \mathrm{Comp}^*_{\mathrm{R}}(F) \leq (n(F)+1)^{\mathrm{hd}(F)}$. Lemma 1 yields:

**Lemma 2 ([19,20]).** *For an unsatisfiable clause-set* $F$ *and* $k \in \mathbb{N}_0$ *we have* $\mathrm{hd}(F) \leq k$ *iff* $F \models_k \bot$ *iff* $F \vdash_k \bot$.

By applying partial assignments we can reach all hardness-levels in a clause-set, as the following lemma shows (see [15] for the straightforward proof):

**Lemma 3.** *For an unsatisfiable clause-set* $F$ *and every* $0 \leq k \leq \mathrm{hd}(F)$ *there exists a partial assignment* $\varphi$ *with* $n(\varphi) = k$ *and* $\mathrm{hd}(\varphi * F) = \mathrm{hd}(F) - k$.

The hardness $\mathrm{hd}(F)$ of arbitrary clause-sets can now be defined as the maximum hardness over all unsatisfiable instances obtained by partial assignments.

**Definition 9.** *The* ***hardness*** $\mathbf{hd}(F) \in \mathbb{N}_0$ *for* $F \in \mathcal{CLS}$ *is the minimal* $k \in \mathbb{N}_0$ *such that for all clauses* $C$ *with* $F \models C$ *we have* $F \models_k C$ *(recall Definition 5; by Lemma 7 this is equivalent to* $F \vdash_k C$*).*

In other words, if $F \neq \top$ then $\mathrm{hd}(F)$ is the maximum of $\mathrm{hd}(\varphi * F)$ for partial assignments $\varphi$ such that $\varphi * F \in \mathcal{USAT}$. The measure $\mathrm{hd}(F)$ for satisfiable $F$ apparently was mentioned the first time in the literature in [1], Definition 8, where there in Lemma 9 it was related to another hardness-alternative for satisfiable $F$. Note that one can restrict attention in Definition 9 to prime implicates $C$. Hardness 0 means that all prime clauses are there, i.e., $\mathrm{hd}(F) = 0$ iff $\mathrm{prc}_0(F) \subseteq F$, where $\mathrm{prc}_0(F)$ is the set of prime implicates of $F$.

**Definition 10.** *For* $k \in \mathbb{N}_0$ *let* $\boldsymbol{\mathcal{UC}_k} := \{F \in \mathcal{CLS} : \mathrm{hd}(F) \leq k\}$ *(the class of* ***unit-refutation complete clause-sets of level*** $k$*).*

The class $\mathcal{UC}_1$ has been introduced in [10] for knowledge compilation. Various (resolution-based) algorithms computing for clause-sets $F$ some equivalent set $F' \in \mathcal{UC}_1$ of prime implicates are discussed. Based on the results from [19,20], we can now give a powerful proof-theoretic characterisation for all classes $\mathcal{UC}_k$:

**Theorem 1.** *For $k \in \mathbb{N}_0$ and $F \in \mathcal{CLS}$ holds $F \in \mathcal{UC}_k$ if and only if $\forall C \in \mathrm{prc}_0(F) : F \vdash_k C$. Thus if every $C \in \mathrm{prc}_0(F)$ has a tree-resolution refutation using at most $2^{k+1}-1$ leaves (i.e., $\mathrm{Comp}^*_{\mathrm{R}}(\varphi_C * F) < 2^{k+1}$), then $\mathrm{hd}(F) \le k$.*

*Proof.* The equivalence $F \in \mathcal{UC}_k \Leftrightarrow \forall C \in \mathrm{prc}_0(F) : F \vdash_k C$ follows from Lemma 1. And if $\mathrm{hd}(F) > k$, then there is $C \in \mathrm{prc}_0(F)$ with $F \nvdash_k C$, and then every tree-resolution derivation of $C$ from $F$ needs at least $2^{k+1}$ leaves due to $2^{\mathrm{hd}(\varphi_C * F)} \le \mathrm{Comp}^*_{\mathrm{R}}(\varphi_C * F)$ (as stated before). □

The following basic lemma follows directly by definition:

**Lemma 4.** *If two clause-sets $F$ and $F'$ are variable-disjoint, then we have:*

1. *If $F, F' \in \mathcal{SAT}$, then $\mathrm{hd}(F \cup F') = \max(\mathrm{hd}(F), \mathrm{hd}(F'))$.*
2. *If $F \in \mathcal{SAT}$ and $F' \in \mathcal{USAT}$, then $\mathrm{hd}(F \cup F') = \mathrm{hd}(F')$.*
3. *If $F, F' \in \mathcal{USAT}$, then $\mathrm{hd}(F \cup F') = \min(\mathrm{hd}(F), \mathrm{hd}(F'))$.*

Via full clause-sets $A_n$ (clause-sets such that each clause contains all variables) with $n$ variables and $2^n$ clauses we obtain (unsatisfiable, simplest) examples with $\mathrm{hd}(A_n) = n$, and when removing one clause for $n \ge 1$, then we obtain satisfiable examples $A'_n$ with $\mathrm{hd}(A'_n) = n - 1$ (see [15] for the proof):

**Lemma 5.** *Consider a full clause-set $F$ (each clause contains all variables).*

1. *If $F$ is unsatisfiable then $\mathrm{hd}(F) = n(F)$.*
2. *If $F \neq \top$, then $\mathrm{hd}(F) = n(F) - \min_{C \in \mathrm{prc}_0(F)} |C|$.*
3. *If for $F$ no two clauses are resolvable, then $\mathrm{hd}(F) = 0$.*

The next lemma yields a way of increasing hardness (see [15] for the proof):

**Lemma 6.** *Consider $F \in \mathcal{CLS}$ and $v \in \mathcal{VA} \setminus \mathrm{var}(F)$. Let $F' := \{C \cup \{v\} : C \in F\} \cup \{C \cup \{\overline{v}\} : C \in F\}$. Then $\mathrm{hd}(F') = \mathrm{hd}(F) + 1$.*

### 5.1 Containment and Stability Properties

The following fundamental lemma is obvious from the definition:

**Lemma 7.** *Consider $\mathcal{C} \subseteq \mathcal{CLS}$ stable under partial assignment and $k \in \mathbb{N}_0$ such that for $F \in \mathcal{C} \cap \mathcal{USAT}$ we have $\mathrm{hd}(F) \le k$. Then $\mathrm{hd}(F) \le k$ for all $F \in \mathcal{C}$.*

We apply Lemma 7 to various well-known classes $\mathcal{C}$ (stating in brackets the source for the bound on the unsatisfiable cases).

**Lemma 8.** *Consider $F \in \mathcal{CLS}$.*

1. *For $\varphi \in \mathcal{PASS}$ we have $\mathrm{hd}(\varphi * F) \le \mathrm{hd}(F)$ (by Lemma 3.11 in [19]).*
2. *$\mathrm{hd}(F) \le n(F)$ (by Lemma 3.18 in [19]).*
3. *If $F \in 2\text{-}\mathcal{CLS} = \{F \in \mathcal{CLS} \mid \forall C \in F : |C| \le 2\}$, then $\mathrm{hd}(F) \le 2$ (by Lemma 5.6 in [19]).*
4. *If $F \in \mathcal{HO} = \{F \in \mathcal{CLS} \mid \forall C \in F : |C \cap \mathcal{VA}| \le 1\}$ (Horn clause-sets), then $\mathrm{hd}(F) \le 1$ by (Lemma 5.8 in [19]).*

5. *More generally, if* $F \in \mathcal{QHO}$*, the set of q-Horn clause-sets (see Section 6.10.2 in [8], and [23]), then* $\mathrm{hd}(F) \leq 2$ *(by Lemma 5.12 in [19]).*
6. *Generalising Horn clause-sets to the hierarchy* $\mathcal{HO}_k$ *from [17] (with* $\mathcal{HO}_1 = \mathcal{HO}$*): if* $F \in \mathcal{HO}_k$ *for* $k \in \mathbb{N}$*, then* $\mathrm{hd}(F) \leq k$ *(by Lemma 5.10 in [19]).*

By a standard autarky-argument for 2–$\mathcal{CLS}$ (see [18]) we can sharpen the hardness-upper-bound 2 for satisfiable clause-sets:

**Lemma 9.** *For* $F \in 2\text{–}\mathcal{CLS} \cap \mathcal{SAT}$ *we have* $\mathrm{hd}(F) \leq 1$.

*Proof.* Consider $\varphi \in \mathcal{PASS}$ with unsatisfiable $\varphi * F$. We have $\mathrm{r}_1(\varphi * F) = \{\bot\}$, since otherwise $\mathrm{r}_1(\varphi * F) \subseteq F$, and thus $\mathrm{r}_1(\varphi * F)$ would be satisfiable. □

We have the following stability properties:

**Lemma 10.** *Consider* $k \in \mathbb{N}_0$.

1. $\mathcal{UC}_k$ *is stable under application of partial assignments (with Lemma 8, Part 1; this might reduce hardness).*
2. $\mathcal{UC}_k$ *is stable under variable-disjoint union (with Lemma 4).*
3. $\mathcal{UC}_k$ *is stable under renaming variables and switching polarities.*
4. $\mathcal{UC}_k$ *is stable under subsumption-elimination.*
5. $\mathcal{UC}_k$ *is stable under addition of inferred clauses (this might reduce hardness).*

A fundamental tool, underlying all inclusion relations presented here, are the hierarchies $G_k(\mathcal{U}, \mathcal{S}) \subseteq \mathcal{CLS}$ introduced in [19,20], using oracles $\mathcal{U} \subseteq \mathcal{USAT}$, $\mathcal{S} \subseteq \mathcal{SAT}$ for (un)satisfiability decision. Only the unsatisfiable instances are relevant here (and thus $\mathcal{S}$ is not employed in our context): The $G_k$-hierarchies contain (finally all) satisfiable instances, as does $\mathcal{UC}_k$, but with a different aim, as can be seen from the fact, that for fixed $k$, membership in $G_k(\mathcal{U}, \mathcal{S})$ is decidable in polynomial time, while it is coNP-complete for $\mathcal{UC}_k$. See [15] for more details, and for the relations to the hierarchies presented in [6], which can be understood as special cases. These considerations lead to

**Lemma 11.** *For all* $k \in \mathbb{N}_0$ *we have* $\Pi_k \subset \mathcal{UC}_{k+1}$ *and* $\Upsilon_k \subset \mathcal{UC}_{k+2}$ *for the hierarchies* $\Pi_k, \Upsilon_k$ *introduced in [6].*

### 5.2 Determining Hardness Computationally

By the well-known computation of $\mathrm{prc}_0(F)$ via resolution-closure we obtain:

**Lemma 12.** *Whether for* $F \in \mathcal{CLS}$ *we have* $\mathrm{hd}(F) = 0$ *or not can be decided in polynomial time, namely* $\mathrm{hd}(F) = 0$ *holds if and only if* $F$ *is stable under resolution modulo subsumption (which means that for all resolvable* $C, D \in F$ *with resolvent* $R$ *there exists* $E \in F$ *with* $E \subseteq R$*).*

Thus if the hardness is known to be at most 1, we can compute it efficiently:

**Corollary 1.** *Consider a class* $\mathcal{C} \subseteq \mathcal{CLS}$ *of clause-sets where* $\mathcal{C} \subseteq \mathcal{UC}_1$ *is known. Then for* $F \in \mathcal{C}$ *one can compute* $\mathrm{hd}(F) \in \{0, 1\}$ *in polynomial time.*

Examples for $\mathcal{C}$ are given by $\mathcal{HO} \subset \mathcal{UC}_1$ and in Subsection 3.1. Another example class with known hardness is given by $2\text{–}\mathcal{CLS} \subset \mathcal{UC}_2$ (Lemma 8), and also here we can compute the hardness efficiently (see [15] for the proof):

**Lemma 13.** *For $F \in 2\text{–}\mathcal{CLS}$ the hardness is computable in polynomial time.*

See Theorem 3 for coNP-completeness of computing an upper bound.

## 6 The SLUR Hierarchy

We now define the $\mathcal{SLUR}_k$ hierarchy, generalising $\mathcal{SLUR}$ (recall Subsection 3.1) in a natural way, by replacing $\mathrm{r}_1$ with $\mathrm{r}_k$. In Subsection 6.1 we show $\mathcal{SLUR}_k = \mathcal{UC}_k$, and as application obtain coNP-completeness of membership decision for $\mathcal{UC}_k$ for $k \geq 1$. In Section 6.2 we determine the relations to the previous hierarchies $\mathcal{SLUR}*(k)$ and CANON$(k)$ as discussed in Subsection 3.2.

**Definition 11.** *Consider $k \in \mathbb{N}_0$. For clause-sets $F, F' \in \mathcal{CLS}$ the relation $\boldsymbol{F \xrightarrow{SLUR:k} F'}$ holds if there is $x \in \mathrm{lit}(F)$ such that $F' = \mathrm{r}_k(\langle x \to 1\rangle * F)$ and $F' \neq \{\bot\}$. The transitive-reflexive closure is denoted by $\boldsymbol{F \xrightarrow{SLUR:k}_* F'}$. The set of all fully reduced clause-sets reachable from $F$ is denoted by $\mathbf{slur_k(F)} := \{F' \in \mathcal{CLS} \mid F \xrightarrow{SLUR:k}_* F' \wedge \neg\exists\, F'' \in \mathcal{CLS} : F' \xrightarrow{SLUR:k} F''\}$. Finally the class of all clause-sets which are either identified by $\mathrm{r}_k$ to be unsatisfiable, or where by $k$-SLUR-reduction always a satisfying assignment is found, is denoted by $\boldsymbol{\mathcal{SLUR}_k} := \{F \in \mathcal{CLS} : \mathrm{r}_k(F) \neq \{\bot\} \Rightarrow \mathrm{slur}_k(F) = \{\top\}\}$.*

We have $\mathcal{SLUR}_1 = \mathcal{SLUR}$. Obviously $\top \in \mathrm{slur}_k(F) \Leftrightarrow F \in \mathcal{SAT}$ for $F \in \mathcal{CLS}$ and all $k$. And by definition we get:

**Lemma 14.** *We have for $F \in \mathcal{CLS}$, $k \in \mathbb{N}_0$ and a partial assignment $\varphi$ with $\mathrm{r}_k(\varphi * F) \neq \{\bot\}$ that $F \xrightarrow{SLUR:k}_* \mathrm{r}_k(\varphi * F)$ holds.*

### 6.1 SLUR = UC

For $F \in \mathcal{UC}_k$ there is the following polynomial-time SAT decision: $F$ is unsatisfiable iff $\mathrm{r}_k(F) = \{\bot\}$. And a satisfying assignment can be found for satisfiable $F$ via self-reduction, that is, probing variables, where unsatisfiability again is checked for by means of $\mathrm{r}_k$. For $k = 1$ this means exactly that the nondeterministic "SLUR"-algorithm will not fail. And that implies that $F \in \mathcal{SLUR}$ holds, where $\mathcal{SLUR}$ is the class of clause-sets where that algorithm never fails. So $\mathcal{UC}_1 \subseteq \mathcal{SLUR}$. Now it turns out, that actually this property characterises $\mathcal{UC}_1$, that is, $\mathcal{UC}_1 = \mathcal{SLUR}$ holds, which makes available the results on $\mathcal{SLUR}$. We now show that this equality between $\mathcal{UC}$ and $\mathcal{SLUR}$ holds in full generality for the $\mathcal{UC}_k$ and $\mathcal{SLUR}_k$ hierarchies.

**Theorem 2.** *For all $k \in \mathbb{N}_0$ holds $\mathcal{SLUR}_k = \mathcal{UC}_k$.*

*Proof.* Consider $F \in \mathcal{CLS}$. We have to show $F \in \mathcal{SLUR}_k \Leftrightarrow \mathrm{hd}(F) \leq k$. For $F \in \mathcal{USAT}$ this follows from the definitions, and thus we assume $F \in \mathcal{SAT}$. First consider $F \in \mathcal{SLUR}_k$. Consider a partial assignment $\varphi$ such that $\varphi * F \in \mathcal{USAT}$. We have to show $\mathrm{r}_k(\varphi * F) = \{\bot\}$, and so assume $\mathrm{r}_k(\varphi * F) \neq \{\bot\}$. It follows $F \xrightarrow{\mathrm{SLUR}}_* \mathrm{r}_k(\varphi * F)$ by Lemma 14. In general we have that $F \in \mathcal{SLUR}_k$ together with $F \in \mathcal{SAT}$ and $F \xrightarrow{\mathrm{SLUR}:k}_* F'$ implies $F' \in \mathcal{SAT}$. Whence $\mathrm{r}_k(\varphi * F) \in \mathcal{SAT}$, contradicting $\varphi * F \in \mathcal{USAT}$. Now assume $\mathrm{hd}(F) \leq k$, and we show $F \in \mathcal{SLUR}_k$. For $F \xrightarrow{\mathrm{SLUR}:k}_* F'$ we have $F' \in \mathcal{SAT}$ by Lemma 8, Part 1, and thus $\mathrm{slur}_k(F) = \{\top\}$. □

**Theorem 3.** *For fixed $k \in \mathbb{N}$ the decision whether $\mathrm{hd}(F) \leq k$ (i.e., whether $F \in \mathcal{UC}_k$, or, by Theorem 2, whether $F \in \mathcal{SLUR}_k$) is coNP-complete.*

*Proof.* The decision whether $F \notin \mathcal{SLUR}_k$ is in NP by definition of $\mathcal{SLUR}_k$ (or use Lemma 3). By Theorem 3 in [7] we have that $\mathcal{SLUR}$ is coNP-complete, which by Lemma 6 can be lifted to higher $k$. □

### 6.2 Comparison to the Previous Hierarchies

The alternative hierarchies $\mathcal{SLUR}*(k)$ and CANON$(k)$ (recall Subsection 3.2) extend $\mathrm{r}_1$ in various ways (maintaining linear-time computation for the (non-deterministic) transitions). We give now short proofs that these alternative hierarchies are subsumed by our hierarchy, while already the second level of our hierarchy is (naturally) not contained in any levels of these two hierarchies (naturally, since the time-exponent for deciding whether a (non-deterministic) transition can be done w.r.t. hierarchy $\mathcal{SLUR}_k$ depends on $k$). First we simplify and generalise the main result of [2], that CANON$(1) \subseteq \mathcal{SLUR}$.

**Theorem 4.** *For $k \in \mathbb{N}_0$ we have:* CANON$(k) \subseteq \mathcal{UC}_k$ *and* $\mathcal{UC}_1 \not\subseteq$ CANON$(k)$ *(and thus* CANON$(k) \subset \mathcal{UC}_k$ *for $k \geq 1$).*

*Proof.* By Theorem 1 and the fact, that the Horton-Strahler number of a tree is at most the height, we see CANON$(k) \subseteq \mathcal{UC}_k$. There are formulas in $\mathcal{HO} \cap \mathcal{USAT}$ with arbitrary resolution-height complexity and so $\mathcal{HO} \not\subseteq$ CANON$(k)$. By $\mathcal{HO} \subset \mathcal{UC}_1$ we get $\mathcal{UC}_1 \not\subseteq$ CANON$(k)$. □

Also the other hierarchy $\mathcal{SLUR}*(k)$ is strictly contained in our hierarchy:

**Theorem 5.** *For all $k \in \mathbb{N}_0$ we have $\mathcal{SLUR}*(k) \subset \mathcal{SLUR}_{k+1}$ and $\mathcal{SLUR}_2 \not\subseteq \mathcal{SLUR}*(k)$.*

*Proof.* The inclusion follows most easily by using Lemma 7 together with the simple fact that $\mathrm{slur}*(k)(F) = \{F\}$ for $F \neq \top$ implies $\mathrm{r}_{k+1}(F) = \{\bot\}$. The non-inclusion follows from CANON$(2) \not\subseteq \mathcal{SLUR}*(k)$ (Lemma 13 in [2]), while by Theorem 4 we have CANON$(2) \subseteq \mathcal{SLUR}_2$. □

In [15] we show that $\mathcal{SLUR}*(k)$ and $\mathcal{SLUR}_k$ are incomparable in general.

## 7 Conclusion and Outlook

We brought together two streams of research, one started by [10] in 1994, introducing $\mathcal{UC}$ for knowledge compilation, one started by [22] in 1995, introducing $\mathcal{SLUR}$ for polytime SAT decision. Two natural generalisations, $\mathcal{UC}_k$ and $\mathcal{SLUR}_k$ have been provided, and the (actually surprising) identity $\mathcal{SLUR}_k = \mathcal{UC}_k$ provides both sides of the equation with additional tools. Various basic lemmas have been shown, providing a framework for elegant and powerful proofs. Regarding computational problems, we solved the most basic questions. The next steps for us, which have already been partially accomplished, consist in the following investigations:

1. Complementary to "unit-refutation completeness" there is the notion of "propagation completeness", as investigated in [9,5]. This will be captured and generalised by a corresponding measure $\mathrm{phd} : \mathcal{CLS} \to \mathbb{N}_0$ of **propagation-hardness**.
2. The real power of SAT representations comes with **new variables**. Expressive power and limitations of the "good representations" have to be studied. Relevant here is [3], which shows that for example the satisfiable pigeonhole formulas $\mathrm{PHP}_m^m$ do not have polysize representations of bounded hardness.
3. Applications of representations of bounded hardness to cryptographic problems has to be experimentally evaluated. We consider especially attacking AES/DES, as preliminary discussed in [14,13].
4. The theory started here has to be generalised via the use of oracles as in [19,20] (this is one way of overcoming the principal barriers shown in [3], by employing oracles which can handle pigeonhole formulas).

## References


1. Ansótegui, C., Bonet, M.L., Levy, J., Manyà, F.: Measuring the hardness of SAT instances. In: Fox, D., Gomes, C. (eds.) Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI 2008), pp. 222–228 (2008)
2. Balyo, T., Gurský, Š., Kučera, P., Vlček, V.: On hierarchies over the SLUR class. In: Twelfth International Symposium on Artificial Intelligence and Mathematics, ISAIM 2012 (2012), http://www.cs.uic.edu/bin/view/Isaim2012/AcceptedPapers
3. Bessiere, C., Katsirelos, G., Narodytska, N., Walsh, T.: Circuit complexity and decompositions of global constraints. In: Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009), pp. 412–418 (2009)
4. Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
5. Bordeaux, L., Marques-Silva, J.: Knowledge Compilation with Empowerment. In: Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G. (eds.) SOFSEM 2012. LNCS, vol. 7147, pp. 612–624. Springer, Heidelberg (2012)
6. Čepek, O., Kučera, P.: Known and new classes of generalized Horn formulae with polynomial recognition and SAT testing. Discrete Applied Mathematics 149, 14–52 (2005)

7. Čepek, O., Kučera, P., Vlček, V.: Properties of SLUR Formulae. In: Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G. (eds.) SOFSEM 2012. LNCS, vol. 7147, pp. 177–189. Springer, Heidelberg (2012)
8. Crama, Y., Hammer, P.L.: Boolean Functions: Theory, Algorithms, and Applications, LNCS. Encyclopedia of Mathematics and Its Applications, vol. 142. Cambridge University Press (2011)
9. Darwiche, A., Pipatsrisawat, K.: On the power of clause-learning SAT solvers as resolution engines. Artificial Intelligence 175(2), 512–525 (2011)
10. del Val, A.: Tractable databases: How to make propositional unit resolution complete through compilation. In: Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR 1994), pp. 551–561 (1994)
11. Franco, J.: Relative size of certain polynomial time solvable subclasses of satisfiability. In: Du, D., Gu, J., Pardalos, P.M. (eds.) Satisfiability Problem: Theory and Applications (DIMACS Workshop), March 11-13, 1996. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 35, pp. 211–223. American Mathematical Society (1997)
12. Franco, J., Gelder, A.V.: A perspective on certain polynomial-time solvable classes of satisfiability. Discrete Applied Mathematics 125, 177–214 (2003)
13. Gwynne, M., Kullmann, O.: Towards a better understanding of hardness. In: The Seventeenth International Conference on Principles and Practice of Constraint Programming (CP 2011): Doctoral Program Proceedings, pp. 37–42 (2011), http://people.cs.kuleuven.be/~guido.tack/dp2011/DP_at_CP2011.pdf
14. Gwynne, M., Kullmann, O.: Towards a better understanding of SAT translations. In: Berger, U., Therien, D. (eds.) Logic and Computational Complexity (LCC 2011), as part of LICS 2011 (2011)
15. Gwynne, M., Kullmann, O.: Generalising unit-refutation completeness and SLUR via nested input resolution. Technical Report arXiv:1204.6529v4 [cs.LO], arXiv (2012)
16. Henschen, L.J., Wos, L.: Unit refutations and Horn sets. Journal of the Association for Computing Machinery 21(4), 590–605 (1974)
17. Büning, H.K.: On generalized Horn formulas and $k$-resolution. Theoretical Computer Science 116, 405–413 (1993)
18. Büning, H.K., Kullmann, O.: Minimal unsatisfiability and autarkies. In: Biere, et al. (eds.) [4], ch. 11, pp. 339–401 (2008)
19. Kullmann, O.: Investigating a general hierarchy of polynomially decidable classes of CNF's based on short tree-like resolution proofs. Technical Report TR99-041, Electronic Colloquium on Computational Complexity, ECCC (1999)
20. Kullmann, O.: Upper and lower bounds on the complexity of generalised resolution and generalised constraint satisfaction problems. Annals of Mathematics and Artificial Intelligence 40(3-4), 303–352 (2004)
21. Kullmann, O.: Present and Future of Practical SAT Solving. In: Creignou, N., Kolaitis, P., Vollmer, H. (eds.) Complexity of Constraints. LNCS, vol. 5250, pp. 283–319. Springer, Heidelberg (2008)
22. Schlipf, J.S., Annexstein, F.S., Franco, J.V., Swaminathan, R.P.: On finding solutions for extended Horn formulas. Information Processing Letters 54, 133–137 (1995)
23. van Maaren, H.: A short note on some tractable cases of the satisfiability problem. Information and Computation 158(2), 125–130 (2000)

# On Structural Parameterizations for the 2-Club Problem

Sepp Hartung, Christian Komusiewicz, and André Nichterlein

Institut für Softwaretechnik und Theoretische Informatik,
TU Berlin, Germany
{sepp.hartung,christian.komusiewicz,andre.nichterlein}@tu-berlin.de

**Abstract.** The NP-hard 2-Club problem is, given an undirected graph $G = (V, E)$ and a positive integer $\ell$, to decide whether there is a vertex set of size at least $\ell$ that induces a subgraph of diameter at most two. We make progress towards a systematic classification of the complexity of 2-Club with respect to structural parameterizations of the input graph. Specifically, we show NP-hardness of 2-Club on graphs that become bipartite by deleting one vertex, on graphs that can be covered by three cliques, and on graphs with domination number two and diameter three. Moreover, we present an algorithm that solves 2-Club in $|V|^{f(k)}$ time, where $k$ is the so-called $h$-index of the input graph. By showing $W[1]$-hardness for this parameter, we provide evidence that the above algorithm cannot be improved to a fixed-parameter algorithm. This also implies $W[1]$-hardness with respect to the degeneracy of the input graph. Finally, we show that 2-Club is fixed-parameter tractable with respect to "distance to co-cluster graphs" and "distance to cluster graphs".

## 1 Introduction

In the analysis of social and biological networks, one important task is to find cohesive subnetworks since these could represent communities or functional subnetworks within the large network. There are several graph-theoretic formulations for modeling these cohesiveness demands such as $s$-cliques [1], $s$-plexes [21], and $s$-clubs [15]. In this work, we study the problem of finding large $s$-clubs within the input network. An $s$-club is a vertex set that induces a subgraph of diameter at most $s$. Thus it is a distance-based relaxation of complete graphs, cliques, which are exactly the graphs of diameter one. For constant $s \geq 1$, the problem is defined as follows.

$s$-Club
**Input:** An undirected graph $G = (V, E)$ and an integer $\ell \geq 1$.
**Question:** Is there a vertex set $S \subseteq V$ of size at least $\ell$ such that $G[S]$ has diameter at most $s$?

In this work, we focus on studying the computational complexity of 2-Club. This is motivated by the following two considerations. First, 2-Club is an important special case concerning the applications: For biological networks, 2-clubs

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 233–243, 2013.

and 3-clubs have been identified as the most reasonable diameter-relaxations of cliques [18]. 2-Club also has applications in the analysis of social networks [14]. Consequently, experimental evaluations concentrate on finding 2-clubs and 3-clubs [13]. Second, 2-Club is the most basic variant of $s$-Club that is different from Clique. For example, being a clique is a hereditary graph property, that is, it is closed under vertex deletion. In contrast, being a 2-club is not hereditary, since deleting vertices can increase the diameter of a graph. Hence, it is interesting to spot differences in the computational complexity of the two problems.

In the spirit of multivariate algorithmics [11,17], we aim to describe how structural properties of the input graph determine the computational complexity of 2-Club. We want to determine sharp boundaries between tractable and intractable special cases of 2-Club, and whether some graph properties, especially those motivated by the structure of social and biological networks, can be exploited algorithmically. The structural properties, called structural graph parameters, are usually described by integers; well-known examples of such parameters are the maximum degree or the treewidth of a graph. Our results use the classical framework of NP-hardness as well as the framework of parameterized complexity to show (parameterized) tractability and intractability of 2-Club with respect to the structural graph parameters under consideration.

**Related Work.** For all $s \geq 1$, $s$-Club is NP-complete on graphs of diameter $s+1$ [3]; 2-Club is NP-complete even on split graphs and, thus, also on chordal graphs [3]. For all $s \geq 1$, $s$-Club is NP-hard to approximate within a factor of $n^{1/2-\epsilon}$ [2]; a simple approximation algorithm obtains a factor of $n^{1/2}$ for even $s \geq 2$ and a factor $n^{2/3}$ for odd $s \geq 3$ [2]. Several heuristics [6] and integer linear programming formulations [3, 6] for $s$-Club have been proposed and experimentally evaluated [13]. 1-Club is equivalent to Clique and thus W[1]-hard with respect to $\ell$. In contrast, for $s \geq 2$, $s$-Club is fixed-parameter tractable with respect to $\ell$ [7, 19], with respect to $n - \ell$ [19][1], and also with respect to the parameter treewidth of $G$ [20]. Moreover, $s$-Club does not admit a polynomial-size kernel with respect to $\ell$ (unless NP $\subseteq$ coNP/poly), but admits a so-called *Turing*-kernel with at most $k^2$-vertices for even $s$ and at most $k^3$-vertices for odd $s$ [19]. 2-Club is solvable in polynomial time on bipartite graphs, on trees, and on interval graphs [20]. In companion work [12], we considered different structural parameters: For instance, we presented fixed-parameter algorithms for the parameters "treewidth" and "size of a vertex cover" and polynomial-size kernels for the parameters "feedback edge set" and "cluster editing number". Furthermore, we presented an efficient implementation for 2-Club based on the fixed-parameter algorithm for the dual parameter $n - \ell$.

**Our Contribution.** We make progress towards a systematic classification of the complexity of 2-Club with respect to structural graph parameters. Figure 1

[1] Schäfer et al. [19] actually considered finding an $s$-club of size *exactly* $\ell$. The claimed fixed-parameter tractability with respect to $n-\ell$ however only holds for the problem of finding an $s$-club of size *at least* $\ell$. The other fixed-parameter tractability results hold for both variants.

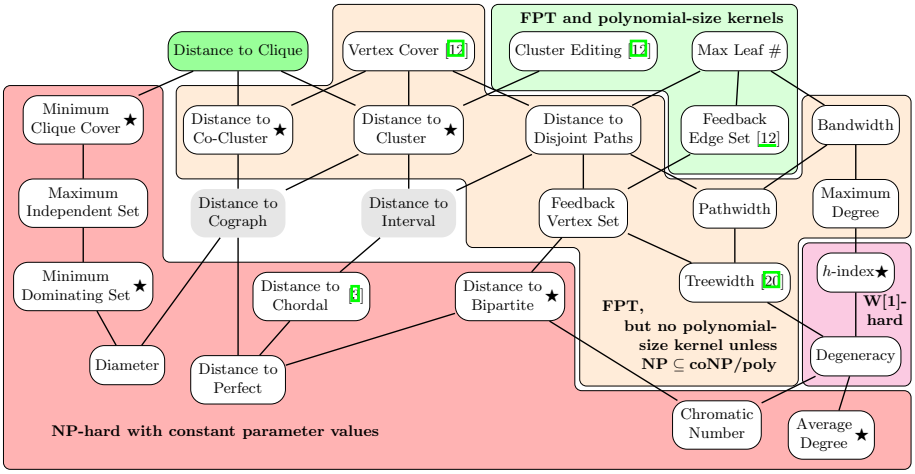



**Fig. 1.** Overview of the relation between structural graph parameters and of our results for 2-Club. An edge from a parameter $\alpha$ to a parameter $\beta$ below of $\alpha$ means that $\beta$ can be upper-bounded in a polynomial (usually linear) function in $\alpha$. The boxes indicate the complexity of 2-Club with respect to the enclosed parameters. 2-Club is FPT with respect to "distance to clique", but it is open whether it admits a polynomial size kernel. The complexity with respect to "distance to interval" and "distance to cograph" is still open. Results obtained in this work are marked with ★. (For the parameters bandwidth and maximum degree, taking the disjoint union of the input graphs is a composition algorithm that proves the non-existence of polynomial-size kernels [5].)

gives an overview of our results and their implications. Therein, for a set of graphs $\Pi$ (for instance the set of bipartite graphs) the parameter *distance to* $\Pi$ measures the number of vertices that have to be deleted in order to obtain a graph that is isomorphic to one in $\Pi$.

In Section 2, we consider the graph parameters minimum clique cover number, minimum dominating set of $G$, and some related graph parameters. We show that 2-Club is NP-hard even if the minimum clique cover number of $G$ is three, that is, the vertices of $G$ can be covered by three cliques. In contrast, we show that if the minimum clique cover number is two, then 2-Club is polynomial-time solvable. Then, we show that 2-Club is NP-hard even if $G$ has a dominating set of size two, that is, there are two vertices $u, v$ in $G$ such that every vertex in $V \setminus \{u, v\}$ is a neighbor of one of the two. This result is tight in the sense that 2-Club is trivially solvable in case $G$ has a dominating set of size one.

In Section 3, we study the parameter distance to bipartite graphs. We show that 2-Club is NP-hard even if the input graph can be transformed into a bipartite graph by deleting only one vertex. This is somewhat surprising since 2-Club is polynomial-time solvable on bipartite graphs [20]. Then, in Section 4, we consider the graph parameter *h-index*: a graph $G$ has *h-index* $k$ if $k$ is the largest number such that $G$ has at least $k$ vertices of degree at least $k$. The study of this parameter is motivated by the fact that the $h$-index is usually small in social

networks (see Section 4 for a more detailed discussion). On the positive side, we show that 2-Club is polynomial-time solvable for constant $k$. On the negative side, we show that 2-Club parameterized by the $h$-index $k$ of the input graph is W[1]-hard. Hence, a running time of $f(k) \cdot n^{O(1)}$ is probably not achievable. This also implies W[1]-hardness with respect to the parameter degeneracy of $G$.

Finally, we describe fixed-parameter algorithms for the parameters distance to cluster and co-cluster graphs. Herein, a *cluster graph* is a vertex-disjoint union of cliques, and a *co-cluster graph* is the complement graph of a cluster graph, that is, it is either an independent set or a complete $p$-partite graph for some $p \leq n$. Interestingly, distance to cluster/co-cluster graph are rare examples for structural graph parameters, that are unrelated to treewidth and still admit a fixed-parameter algorithm (see Figure 1).

**Preliminaries.** We only consider undirected and simple graphs $G = (V, E)$ where $n := |V|$ and $m := |E|$. For a vertex set $S \subseteq V$, let $G[S]$ denote the *subgraph induced by* $S$ and $G - S := G[V \setminus S]$. We use $\text{dist}_G(u, v)$ to denote the *distance between* $u$ *and* $v$ in $G$, that is, the length of a shortest path between $u$ and $v$. For a vertex $v \in V$ and an integer $t \geq 1$, denote by $N_t^G(v) := \{u \in V \setminus \{v\} \mid \text{dist}_G(u, v) \leq t\}$ the set of vertices within distance at most $t$ to $v$. If the graph is clear from the context, we omit the superscript $G$. Moreover, we set $N_t[v] := N_t(v) \cup \{v\}$, $N[v] := N_1[v]$ and $N(v) := N_1(v)$. Two vertices $v$ and $w$ are *twins* if $N(v) \setminus \{w\} = N(w) \setminus \{v\}$ and they are *twins with respect to a vertex set* $X$ if $N(v) \cap X = N(w) \cap X$. The twin relation is an equivalence relation; the corresponding equivalence classes are called *twin classes*. The following observation is easy to see.

**Observation 1.** Let $S$ be a $s$-club in a graph $G = (V, E)$ and let $u, v \in V$ be twins. If $u \in S$ and $|S| > 1$, then $S \cup \{v\}$ is also an $s$-club in $G$.

For the relevant notions of parameterized complexity we refer to [9, 16]. For the parameters distance to cluster/co-cluster graph we assume that a deletion set is provided as an additional input. Note that for both of these parameters there is a polynomial-time constant factor approximation algorithm since cluster graphs and co-cluster graphs are characterized by forbidden induced subgraphs on three vertices. Due to the space restrictions, some proofs are deferred to a long version.

## 2 Clique Cover Number and Domination Number

In this section, we prove that on graphs of diameter at most three, 2-Club is NP-hard even if either the clique cover number is three or the domination number is two. We first show that these bounds are tight. The size of a maximum independent set is at most the size of a minimum clique cover. Moreover, since each maximal independent set is a dominating set, a minimum dominating set is also at most the size of a minimum clique cover.

**Proposition 1.** 2-Club *is polynomial-time solvable on graphs where the size of a maximum independent set is at most two.*

*Proof.* Let $G = (V, E)$ be a graph. If a maximum independent set in $G$ has size one or it has diameter two, then $V$ is a 2-club. Otherwise, if the maximum independent set in $G$ is of size two, then iterate over all possibilities to choose two vertices $v, u \in V$. Denoting by $G'$ the graph that results from deleting $N(v) \cap N(u)$ in $G$, output a maximum size set $N^{G'}[v] \cup (N^{G'}(u) \cap N_2^{G'}(v))$ among all iterations.

We next prove the correctness of the above algorithm. For a maximum size 2-club $S \subset V$ in $G$, there are two vertices $v, u \in V$ such that $v \in S$ and $d_{G[S \cup \{u\}]}(v, u) > 2$, implying that $N(v) \cap N(u) \cap S = \emptyset$. Moreover, $N^{G'}[v]$ and $N^{G'}[u]$ are cliques: Two non-adjacent vertices in $N^{G'}(v)$ (in $N^{G'}(u)$) would form together with $u$ (with $v$) an independent set.

Since $N^{G'}[v]$ is a clique and $v \in S$, $G[S \cup N^{G'}(v)]$ is a 2-club and thus $N^{G'}[v] \subseteq S$ by the maximality of $S$. Moreover, since $\{v, u\}$ is a maximum independent set and thus also a dominating set it remains to specify $N(u) \cap S$. However, since $N^{G'}[u]$ is a clique and each vertex in $S$ has to be adjacent to at least one vertex in $N^{G'}(v)$, it follows that $S = N^{G'}[v] \cup (N^{G'}(u) \cap N_2^{G'}(v))$. □

The following theorem shows that the bound on the maximum independent set size in Proposition 1 is tight.

**Theorem 1.** 2-Club *is NP-hard on graphs with clique cover number three and diameter three.*

*Proof.* We describe a reduction from Clique. Let $(G = (V, E), k)$ be a Clique instance. We construct a graph $G' = (V', E')$ consisting of three disjoint vertex sets, that is, $V' = V_1 \cup V_2 \cup V_E$. Set $V_i$, $i \in \{1, 2\}$, to $V_i = V_i^V \cup V_i^{\text{big}}$, where $V_i^V$ is a copy of $V$ and $V_i^{\text{big}}$ is a set of $n^5$ vertices. Let $u, v \in V$ be two adjacent vertices in $G$ and let $u_1, v_1 \in V_1$, $u_2, v_2 \in V_2$ be the copies of $u$ and $v$ in $G'$. Then add the vertices $e_{uv}$ and $e_{vu}$ to $V_E$ and add the edges $\{v_1, e_{vu}\}, \{u_2, e_{vu}\}, \{u_1, e_{uv}\}, \{v_2, e_{uv}\}$ to $G'$. Furthermore, add for each vertex $v \in V$ the vertices $V_E^v = e_v^1, e_v^2, \ldots, e_v^{n^3}$ to $V_E$ and make $v_1$ and $v_2$ adjacent to all these new vertices. Finally, make the following vertex sets to cliques: $V_1, V_2, V_E$, and $V_1^{\text{big}} \cup V_2^{\text{big}}$. Observe that $G'$ has diameter three and that it has a clique cover number of three.

We now prove that $G$ has a clique of size $k \Leftrightarrow G'$ has a 2-club of size $k' = 2n^5 + kn^3 + 2k + 2\binom{k}{2}$.

"⇒:" Let $S$ be a clique of size $k$ in $G$. Let $S_c$ contain all the copies of the vertices of $S$. Furthermore, let $S_E := \{e_{uv} \mid u_1 \in S_c \wedge v_2 \in S_c\}$ and $S_b := \{e_v^i \mid v_1 \in S_c \wedge 1 \leq i \leq n^3\}$. We now show that $S' := S_c \cup S_E \cup S_b \cup V_1^{\text{big}} \cup V_2^{\text{big}}$ is a 2-club of size $k'$. First, observe that $|V_1^{\text{big}} \cup V_2^{\text{big}}| = 2n^5$ and $|S_c| = 2k$. Hence, $|S_b| = kn^3$ and $|S_E| = 2\binom{k}{2}$. Thus, $S'$ has the desired size. With a straightforward case distinction one can check that $S'$ is indeed a 2-club.

"⇐:" Let $S'$ be a 2-club of size $k'$. Observe that $G'$ consists of $|V'| = 2n^5 + 2n + 2m + n^4$ vertices. Since $k' > 2n^5$ at least one vertex of $V_1^{\text{big}}$ and of $V_2^{\text{big}}$ is in $S$. Since all vertices in $V_1^{\text{big}}$ and in $V_2^{\text{big}}$ are twins, we can assume that all vertices of $V_1^{\text{big}} \cup V_2^{\text{big}}$ are contained in $S'$. Analogously, it follows that at least $k$ sets $V_E^{v^1}, V_E^{v^2}, V_E^{v^3}, \ldots, V_E^{v^k}$ are completely contained in $S'$. Since $S'$ is

a 2-club, the distance from vertices in $V_i^{\text{big}}$ to vertices in $V_E^{v^j}$ is at most two. Hence, for each set $V_E^{v^j}$ in $S'$ the two neighbors $v_1^j$ and $v_2^j$ of vertices in $V_E^{v^j}$ are also contained in $S'$. Since the distance of $v_1^i$ and $v_2^j$ for $v_1^i, v_2^j \in S'$ is also at most two, the vertices $e_{v^i v^j}$ and $e_{v^j v^i}$ are part of $S'$ as well. Consequently, $v^i$ and $v^j$ are adjacent in $G$. Therefore, the vertices $v^1, \ldots, v^k$ form a size-$k$ clique in $G$. □

Since a maximum independent set is also a dominating set, Theorem 1 implies that 2-Club is NP-hard on graphs with domination number three and diameter three. In contrast, for domination number one 2-Club is trivial. The following theorem shows that this cannot be extended.

**Theorem 2.** 2-Club *is NP-hard even on graphs with domination number two and diameter three.*

*Proof.* We present a reduction from Clique. Let $(G = (V, E), k)$ be a Clique instance and assume that $G$ does not contain isolated vertices. We construct the graph $G'$ as follows. First copy all vertices of $V$ into $G'$. In $G'$ the vertex set $V$ will form an independent set. Now, for each edge $\{u, v\} \in E$ add an *edge-vertex* $e_{\{u,v\}}$ to $G'$ and make $e_{\{u,v\}}$ adjacent to $u$ and $v$. Let $V_E$ denote the set of edge-vertices. Next, add a vertex set $C$ of size $n + 2$ to $G'$ and make $C \cup V_E$ a clique. Finally, add a new vertex $v^*$ to $G'$ and make $v^*$ adjacent to all vertices in $V$. Observe that $v^*$ plus an arbitrary vertex from $V_E \cup C$ are a dominating set of $G'$ and that $G'$ has diameter three. We complete the proof by showing that $G$ has a clique of size $k \Leftrightarrow G'$ has a 2-club of size at least $|C| + |V_E| + k$.

"$\Rightarrow$:" Let $K$ be a size-$k$ clique in $G$. Then, $S := K \cup C \cup V_E$ is a size-$|C|+|V_E|+k$ 2-club in $G$: First, each vertex in $C \cup V_E$ has distance two to all other vertices $S$. Second, each pair of vertices $u, v \in K$ is adjacent in $G$ and thus they have the common neighbor $e_{\{u,v\}}$ in $V_E$.

"$\Leftarrow$:" Let $S$ be a 2-club of size $|C| + |V_E| + k$ in $G'$. Since $|C| > |V \cup \{v^*\}|$, it follows that there is at least one vertex $c \in S \cap C$. Since $c$ and $v^*$ have distance three, it follows that $v^* \notin S$. Now since $S$ is a 2-club, each pair of vertices $u, v \in S \cap V$ has at least one common neighbor in $S$. Hence, $V_E$ contains the edge-vertex $e_{\{u,v\}}$. Consequently, $S \cap V$ is a size-$k$ clique in $G$. □

## 3 Distance to Bipartite Graphs

A 2-club in a bipartite graph is a biclique and, thus, 2-Club is polynomial-time solvable on bipartite graphs [20]. However, 2-Club is already NP-hard on graphs that become bipartite by deleting only one vertex.

**Theorem 3.** 2-Club *is NP-hard even on graphs with distance one to bipartite graphs.*

*Proof.* We reduce from the NP-hard Maximum 2-SAT problem: Given a positive integer $k$ and a set $\mathcal{C} := \{C_1, \ldots, C_m\}$ of clauses over a variable set

$X = \{x_1, \ldots, x_n\}$ where each clause $C_i$ contains two literals, the question is whether there is an assignment $\beta$ that satisfies at least $k$ clauses.

Given an instance of Maximum 2-SAT where we assume that each clause occurs only once, we construct an undirected graph $G = (V, E)$. The vertex set $V$ consists of the four disjoint vertex sets $V_{\mathcal{C}}$, $V_F$, $V_X^1$, $V_X^2$, and one additional vertex $v^*$. The construction of the four subsets of $V$ is as follows.

The vertex set $V_{\mathcal{C}}$ contains one vertex $c_i$ for each clause $C_i \in \mathcal{C}$. The vertex set $V_F$ contains for each variable $x \in X$ exactly $n^5$ vertices $x^1 \ldots x^{n^5}$. The vertex set $V_X^1$ contains for each variable $x \in X$ two vertices: $x_t$ which corresponds to assigning true to $x$ and $x_f$ which corresponds to assigning false to $x$. The vertex set $V_X^2$ is constructed similarly, but for every variable $x \in X$ it contains $2 \cdot n^3$ vertices: the vertices $x_t^1, \ldots x_t^{n^3}$ which correspond to assigning true to $x$, and the vertices $x_f^1, \ldots x_f^{n^3}$ which correspond to assigning false to $x$.

Next, we describe the construction of the edge set $E$. The vertex $v^*$ is made adjacent to all vertices in $V_{\mathcal{C}} \cup V_F \cup V_X^1$. Each vertex $c_i \in V_{\mathcal{C}}$ is made adjacent to the two vertices in $V_X^1$ that correspond to the two literals in $C_i$. Each vertex $x^i \in V_F$ is made adjacent to $x_t$ and $x_f$, that is, the two vertices of $V_X^1$ that correspond to the two truth assignments for the variable $x$. Finally, each vertex $x_t^i \in V_X^2$ is made adjacent to all vertices of $V_X^1$ except to the vertex $x_f$. Similarly, each $x_f^i \in V_X^2$ is made adjacent to all vertices of $V_X^1$ except to $x_t$. This completes the construction of $G$ which can clearly be performed in polynomial time. Observe that the removal of $v^*$ makes $G$ bipartite: each of the four vertex sets is an independent set and the vertices of $V_{\mathcal{C}}$, $V_F$, and $V_X^2$ are only adjacent to vertices of $V_X^1$.

The main idea behind the construction is as follows. The size of the 2-club forces the solution to contain the majority of the vertices in $V_F$ and $V_X^2$. As a consequence, for each $x \in X$ exactly one of $x_t$ or $x_f$ is in the 2-club. Hence, the vertices from $V_X^2$ in the 2-club represent a truth assignment. In order to fulfill the bound on the 2-club size, at least $k$ vertices from $V_{\mathcal{C}}$ are in the 2-club; these vertices can only be added if the corresponding clauses are satisfied by the represented truth assignment. It remains to show the following claim (we omit the details).

*Claim.* $(\mathcal{C}, k)$ is a yes-instance of Maximum 2-Sat $\Leftrightarrow$ $G$ has a 2-club of size $n^6 + n^4 + n + k + 1$. □

## 4 Average Degree and $h$-Index

2-Club is fixed-parameter tractable for the parameter maximum degree [19]. In social networks, the degree distribution often follows a power law, implying that there are some high-degree vertices but most vertices have low degree [4]. This suggests considering stronger parameters such as $h$-index, degeneracy, and average degree. Unfortunately, 2-Club is NP-hard even with constant average degree.

**Proposition 2.** *For any constant $\alpha > 2$, 2-Club is NP-hard on connected graphs with average degree at most $\alpha$.*

*Proof.* Let $(G, \ell)$ be an instance of 2-Club where $\Delta$ is the maximum degree of $G$. We can assume that $\ell > \Delta + 2$ since, as shown for instance in the proof of

Theorem 1, 2-Club remains NP-hard in this case. We add a path $P$ to $G$ and an edge from an endpoint $p$ of $P$ to an arbitrary vertex $v \in V$. Since $\ell > \Delta + 2$, any 2-club of size at least $\ell$ contains at least one vertex that is not in $P$. Furthermore, it cannot contain $p$ and $v$ since in this case it is a subset of either $N[v]$ or $N[p]$ which both have size at most $\Delta + 2$ ($v$ has degree at most $\Delta$ in $G$). Hence, the instances are equivalent. Putting at least $\lceil \frac{2m}{\alpha - 2} - n \rceil$ vertices in $P$ ensures that the resulting graph has average degree at most $\alpha$. □

Proposition 2 suggests considering "weaker" parameters such as degeneracy or *h-index* [10] of $G$. Recall that having $h$-index $k$ means that there are at most $k$ vertices with degree greater than $k$. Since social networks have small $h$-index [12], fixed-parameter tractability with respect to the $h$-index would be desirable. Unfortunately, we show that 2-Club is W[1]-hard when parameterized by the $h$-index. Following this result, we show that there is "at least" an algorithm that is polynomial for constant $h$-index.

**Theorem 4.** 2-Club *parameterized by h-index is W[1]-hard.*

Since the reduction in the proof of Theorem 4 is from Multicolored Clique and in the reduction the new parameter is linearly bounded in the old one, the results of Chen et al. [8] imply the following.

**Corollary 1.** 2-Club *cannot be solved in* $n^{o(k)}$*-time on graphs with h-index* $k$ *unless the exponential time hypothesis fails.*

We next prove that there is an XP-algorithm for the parameter $h$-index.

**Theorem 5.** 2-Club *can be solved in* $f(k) \cdot n^{2^k} \cdot nm$ *time where* $k$ *is the h-index of the input graph and* $f$ *solely depends on* $k$.

*Proof.* We give an algorithm that finds a maximum 2-club in $G = (V, E)$ in the claimed running time. Let $X \subseteq V$ be the set of all vertices with degree greater than $k$. By definition, $|X| \leq k$. In a first step, branch into the at most $2^k$ cases to guess a subset $X' \subseteq X$ that is contained in a maximum 2-club $S$ for $G$. In case $X' = \emptyset$, one can apply the fixed-parameter algorithm for the parameter maximum degree. In each other branch, proceed as follows. First, delete all vertices from $X \setminus X'$ and while there are vertices that have distance greater than two to any vertex in $X'$, delete them. Denote the resulting graph by $G' = (V', E')$. We next describe how to find a maximum 2-club in $G' = (V', E')$ that contains $X'$.

Partition all vertices in $V' \setminus X'$ into the at most $2^k$ twin classes $T_1, \ldots, T_p$ with respect to $X'$. Two twin classes $T$ and $T'$ are in *conflict* if $N(T) \cap N(T') \cap X' = \emptyset$. Now, the crucial observation is that, if $T$ and $T'$ are in conflict, then all vertices in $(T \cup T') \cap S$ are contained in the same connected component of $G'[S \setminus X']$. Then, since all vertices in $T \cap S$ have in $G'[S \setminus X]$ distance at most two to all vertices in $T' \cap S$, it follows that all vertices in $T \cap S$ have pairwise distance at most four in $G'[S \setminus X']$.

Now, branch into the $O(n^{2^k})$ cases to choose for each twin class $T$ a *center* $c$, that is, a vertex from $T \cap S$. Clearly, if $T \cap S = \emptyset$, then there is no center $c$ and

we delete all vertices in $T$. Consider a remaining twin class $T$ that is in conflict to any other twin class. By the observation above, $T \cap S$ is contained in one connected component of $G'[S \setminus X']$ and in this component all vertices in $T \cap S$ have pairwise distance at most four. Moreover, the graph $G[V' \setminus X']$ has maximum degree at most $k$. Thus for the center $c$ of $T$ one can guess $N_4^S(c) := N_4(c) \cap S$ by branching into at most $2^{k^4}$ cases. Guess the set $N_4^S(c)$ for all centers $c$ where the corresponding twin class is in conflict to at least one other twin class and fix them to be contained in the desired 2-club $S$. Delete all vertices in $T$ guessed to be not contained in $N_4^S(c)$.

Let $\tilde{S}$ be the set of vertices guessed to be contained in $S$. Next, while there is a vertex $v \in V' \setminus \tilde{S}$ that has distance greater than two to any vertex in $\tilde{S}$, delete $v$. Afterwards, check whether all vertices in $\tilde{S}$ have pairwise distance at most two. (If this check fails, then this branch cannot lead to any solution.) We next prove that the remaining graph is a 2-club.

In the remaining graph, each pair of vertices in $V' \setminus \tilde{S}$ has distance at most two and thus the graph is a 2-club: Suppose that two remaining vertices $v, w \in V' \setminus \tilde{S}$ have distance greater than two. Let $T$ and $T'$ be the twin classes with the corresponding centers $c, c'$ such that $v \in T$ and $w \in T'$. In case $T = T'$, it follows that $N(T) \cap X' = \emptyset$ (since $X' \subseteq \tilde{S}$). However, since $T$ cannot be in conflict with any other twin class (otherwise $v, w \in N_4^S(c) \subseteq \tilde{S}$), it follows that $S$ only contains the twin class $T$. This implies that $v$ and $w$ have distance greater than two to all vertices in $X'$ (note that $X' \neq \emptyset$), a contradiction. In case $T \neq T'$, since $N(v) \cap N(w) = \emptyset$ it follows that $T$ is in conflict to $T'$, implying that $v \in N_4^S(c)$ and $w \in N_4^S(c')$, a contradiction to $v, w \in V' \setminus \tilde{S}$. □

## 5 Distance to Cluster and Co-cluster Graphs

We now present a simple fixed-parameter algorithm for 2-Club parameterized by distance to co-cluster graphs. The algorithm is based on the fact that each co-cluster graph is either an independent set or a 2-club.

**Theorem 6.** 2-Club *is solvable in* $O(2^k \cdot 2^{2^k} \cdot nm)$ *time where* $k$ *denotes the distance to co-cluster graphs.*

*Proof.* Let $(G, X, \ell)$ be an 2-Club instance where $X$ with $|X| = k$ and $G - X$ is a co-cluster graph. Note that the co-cluster graph $G - X$ is either a connected graph or an independent set. In the case that $G - X$ is an independent set, the set $X$ is a vertex cover and we thus apply the algorithm we gave in companion work [12] to solve the instance in $O(2^k \cdot 2^{2^k} \cdot nm)$ time.

Hence, assume that $G - X$ is connected. Since $G - X$ is the complement of a cluster graph, this implies that $G - X$ is a 2-club. Thus, if $\ell \leq n - k$, then we can trivially answer yes. Hence, assume that $\ell > n - k$ or, equivalently, $k > n - \ell$. Schäfer et al. [19] showed that 2-club can be solved in $O(2^{n-\ell} nm)$ (simply choose a vertex pair having distance at least three and branch into the two cases of deleting one of them). Since $k > n - \ell$ it follows that 2-club can be solved in $O(2^k nm)$ time in this case. □

Next, we present a fixed-parameter algorithm for the parameter distance to cluster graphs.

**Theorem 7.** 2-Club *is solvable in* $O(2^k \cdot 3^{2^k} \cdot nm)$ *time where* $k$ *denotes distance to cluster graphs.*

## 6 Conclusion

Although the complexity status of 2-Club is resolved for most of the parameters in the complexity landscape shown in Figure 1, some open questions remain. What is the complexity of 2-Club parameterized by "distance to interval graphs" or "distance to cographs"? The latter parameter seems particularly interesting since every induced subgraph of a cograph has diameter two. Hence, the distance to cographs measures the distance from this trivial special case. In contrast to the parameter $h$-index, it is open whether 2-Club parameterized by the degeneracy is in XP or NP-hard on graphs with constant degeneracy. Finally, it would be interesting to see which results carry over to 3-Club [13, 18].

## References

[1] Alba, R.: A graph-theoretic definition of a sociometric clique. J. Math. Sociol. 3(1), 113–126 (1973)
[2] Asahiro, Y., Miyano, E., Samizo, K.: Approximating Maximum Diameter-Bounded Subgraphs. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 615–626. Springer, Heidelberg (2010)
[3] Balasundaram, B., Butenko, S., Trukhanovzu, S.: Novel approaches for analyzing biological networks. J. Comb. Optim. 10(1), 23–39 (2005)
[4] Barabási, A., Albert, R.: Emergence of scaling in random networks. Science 286(5439), 509 (1999)
[5] Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. J. Comput. System Sci. 75(8), 423–434 (2009)
[6] Bourjolly, J.-M., Laporte, G., Pesant, G.: An exact algorithm for the maximum $k$-club problem in an undirected graph. European J. Oper. Res. 138(1), 21–28 (2002)
[7] Chang, M.S., Hung, L.J., Lin, C.R., Su, P.C.: Finding large $k$-clubs in undirected graphs. In: Proc. 28th Workshop on Combinatorial Mathematics and Computation Theory (2011)
[8] Chen, J., Chor, B., Fellows, M., Huang, X., Juedes, D.W., Kanj, I.A., Xia, G.: Tight lower bounds for certain parameterized NP-hard problems. Inf. Comput. 201(2), 216–231 (2005)
[9] Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer (1999)
[10] Eppstein, D., Spiro, E.S.: The $h$-Index of a Graph and Its Application to Dynamic Subgraph Statistics. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) WADS 2009. LNCS, vol. 5664, pp. 278–289. Springer, Heidelberg (2009)
[11] Fellows, M.: Towards Fully Multivariate Algorithmics: Some New Results and Directions in Parameter Ecology. In: Fiala, J., Kratochvíl, J., Miller, M. (eds.) IWOCA 2009. LNCS, vol. 5874, pp. 2–10. Springer, Heidelberg (2009)

[12] Hartung, S., Komusiewicz, C., Nichterlein, A.: Parameterized Algorithmics and Computational Experiments for Finding 2-Clubs. In: Thilikos, D.M., Woeginger, G.J. (eds.) IPEC 2012. LNCS, vol. 7535, pp. 231–241. Springer, Heidelberg (2012)
[13] Mahdavi, F., Balasundaram, B.: On inclusionwise maximal and maximum cardinality $k$-clubs in graphs. Discrete Optim. (to appear, 2012)
[14] Memon, N., Larsen, H.L.: Structural Analysis and Mathematical Methods for Destabilizing Terrorist Networks Using Investigative Data Mining. In: Li, X., Zaïane, O.R., Li, Z.-h. (eds.) ADMA 2006. LNCS (LNAI), vol. 4093, pp. 1037–1048. Springer, Heidelberg (2006)
[15] Mokken, R.J.: Cliques, Clubs and Clans. Quality and Quantity 13, 161–173 (1979)
[16] Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press (2006)
[17] Niedermeier, R.: Reflections on multivariate algorithmics and problem parameterization. In: Proc. 27th STACS. LIPIcs, vol. 5, pp. 17–32. Schloss Dagstuhl–Leibniz-Zentrum für Informatik (2010)
[18] Pasupuleti, S.: Detection of Protein Complexes in Protein Interaction Networks Using n-Clubs. In: Marchiori, E., Moore, J.H. (eds.) EvoBIO 2008. LNCS, vol. 4973, pp. 153–164. Springer, Heidelberg (2008)
[19] Schäfer, A., Komusiewicz, C., Moser, H., Niedermeier, R.: Parameterized computational complexity of finding small-diameter subgraphs. Optim. Lett. 6(5) (2012)
[20] Schäfer, A.: Exact algorithms for s-club finding and related problems. Diploma thesis, Friedrich-Schiller-Universität Jena (2009)
[21] Seidman, S.B., Foster, B.L.: A graph-theoretic generalization of the clique concept. J. Math. Sociol. 6, 139–154 (1978)

# On Languages of One-Dimensional Overlapping Tiles

David Janin*

Université de Bordeaux, LaBRI UMR 5800,
351, Cours de la libération, F-33405 Talence
janin@labri.fr

**Abstract.** A one-dimensional tile with overlaps is a standard finite word that carries some more information that is used to say when the concatenation of two tiles is legal. Known since the mid 70's in the rich mathematical field of inverse monoid theory, this model of tiles with the associated partial product have yet not been much studied in theoretical computer science despite some implicit appearances in studies of two-way automata in the 80's.

In this paper, after giving an explicit description of McAlister monoid, we define and study several classical classes of languages of tiles: from recognizable languages (REC) definable by morphism into finite monoids up to languages definable in monadic second order logic (MSO).

We show that the class of MSO definable languages of tiles is both *simple*: these languages are finite sums of Cartesian products of rational languages, and *robust*: the class is closed under product, iterated product (star), inverse and projection on context tiles. A equivalent notion of *extended* regular expression is deduced from these results.

The much smaller class of REC recognizable languages of tiles is then studied. We describe few examples and we prove that these languages are tightly linked with covers of periodic bi-infinite words.

## 1 Introduction

### Background

In this paper, we study languages of one-dimensional discrete overlapping tiles. These tiles already appear in the 70's as elements of McAlister's monoid in the rich mathematical field of inverse monoid theory (see 9.4. in [7]). In particular, though sometimes implicitly, they are used for studying the structure of (zigzag) covers of finite, infinite or bi-infinite words [1]. McAlister's monoid also appears in studies of the structure of tiling (in the usual sense with no overlaps) of the $d$-dimensional Euclidian space $\mathbb{R}^d$ [6,1].

In a field more closely related with computer science, overlapping tiles also appear decades ago in study of two-way automata on words (they are called word bisections in [11]). But there the underlying monoid structure is left at most implicit. Only recently, it has been shown that one can relevantly defined two-way

* Partially funded by French ANR, projet ContInt 2012: INEDIT.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 244–256, 2013.

word automata semantics by mapping partial runs to elements of McAslister's Monoid [3].

Oddly enough to be mentioned, our interest in studying languages of positive tiles came from application perspectives in computational music theory [5]. In particular, tiles and their continuous variants, can be used to describe advanced synchronization mechanisms between musical patterns [2].

Our purpose here is to provide the computer science flavored basis for the systematic study of languages of tiles. In this paper, we especially focus our attention on characterizing two classical classes of languages that are defined on tiles. The class REC and the class MSO.

**Outline**

We first define the monoid of overlapping tiles and prove it isomorphic to the inverse monoid of McAlister. Three classes of languages of tiles are then considered:

- the class $REC$ of languages definable as pre-images of monoid morphisms into finite monoids,
- the class $REG$ (resp. $REG_C$) of languages definable by means of finite Kleene's regular expressions (Kleene 's regular expression extended with some projection operators),
- the class $MSO$ of languages of tiles definable by means of formula of monadic second order logic.

The largest class $MSO$ is shown to be both *simple*: these languages are finite sums of Cartesian products of rational languages of words (Theorem 4), and *robust*: this class is closed under sum, product, star, inverse and context projection operators (Theorem 5). As a consequence of robustness, we show that $REG_C \subseteq MSO$ (corollary 6). As a consequence of simplicity, we show that $MSO \subseteq REG_C$ (corollary 7).

The class $REC$ is studied in the last part. Compared to the class $MSO$, it really collapses. A simple example shows that it is strictly included into $MSO$ (Lemma 8). Still a non trivial example of an onto morphism from the monoid of tiles to a finite monoid is provided (Theorem 11). It illustrates the fact, generalized in Theorem 12, that recognizable languages of tiles are necessarily sort of languages of covers of finitely many bi-infinite words.

For the picture to be complete, let us mention that the intermediate class $REG$ is characterized in a compagnon paper [3] by means of finite state two-way automata. It is strictly included in the class $REG_C$ as witnessed (with a simple pumping lemma argument) by the language of idempotent tiles. In other words $REC \subset\subset REG \subset REG_C = MSO$.

**Preliminaries**

*The free monoid.* Given a finite alphabet $A$, $A^*$ denotes the free monoid generated by $A$, 1 denotes the neutral element. The concatenation of two words $u$ and $v$ is denoted by $uv$.

*Prefix and suffix lattices.* $\leq_p$ stands for the prefix order over $A^*$, $\leq_s$ for the suffix order. $\vee_p$ (resp. $\vee_s$) denotes the joint operator for the prefix (resp. suffix) order: thus for all words $u$ and $v$, $u \vee_p v$ (resp. $u \vee_s v$) is the least word whose both $u$ and $v$ are prefixes (resp. suffixes).

The extended monoid $A^* + \{0\}$ (with $0u = u0 = 0$ for every word $u$), ordered by $\leq_p$ (extended with $u \leq_p 0$ for every word $u$), is a lattice; in particular, $u \vee_p v = 0$ whenever neither $u$ is a prefix of $v$, nor $v$ is a prefix of $u$. Symmetric properties hold in the suffix lattice.

*Syntactic inverses.* Given $\overline{A}$ a disjoint copy of $A$, $u \mapsto \overline{u}$ denotes the mapping from $(A + \overline{A})^*$ to itself inductively defined by $\overline{1} = 1$, for every letter $a \in A$, $\overline{a}$ is the copy of $a$ in $\overline{A}$ and $\overline{\overline{a}} = a$, for every word $u \in (A + \overline{A})^*$, $\overline{au} = \overline{u}.\overline{a}$. The mapping $u \mapsto \overline{u}$ is involutive ($\overline{\overline{u}} = u$ for every word $u$); it is an antimorphism of the free monoid $(A + \overline{A})^*$, i.e. for all words $u$ and $v \in (A + \overline{A})^*$, $\overline{uv} = \overline{v}.\overline{u}$.

*Free group.* The free group $FG(A)$ generated by $A$ is the quotient of $(A+\overline{A})^*$ by the least congruence $\simeq$ such that, for every letter $a \in A$, $a\overline{a} \simeq 1$ and $\overline{a}a \simeq 1$.

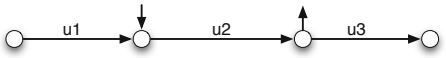
## 2 The Monoid of Overlapping Tiles

We give in this section a description of the monoid of overlapping tiles. It is shown to be isomorphic to McAlister's monoid [8].

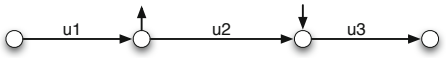
**Positive, Negative and Context Tiles**

A *tile* over the alphabet $A$ is a triple of words $u = (u_1, u_2, u_3) \in A^* \times (A^* + \overline{A}^*) \times A^*$ such that, if $u_2 \in \overline{A}^*$, its inverse $\overline{u_2}$ is a suffix of $u_1$ and a prefix of $u_3$. When $u_2 \in A^*$ we say that $u$ is a *positive tile*. When $u_2 \in \overline{A}^*$ we say that $u$ is a *negative tile*. When $u_2 = 1$, i.e. when $u$ is both positive and negative, we say that $u$ is a *context tile*.

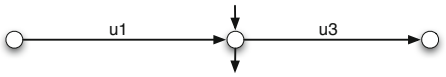
A positive tile $u = (u_1, u_2, u_3)$ is conveniently drawn as a (linear, unidirectional and left to right) Munn's birooted word tree [10]:




where the dangling input arrow (marking the beginning of the root) appears on the left of the dangling output arrow (marking the end of the root). A negative tile of the form $u = (u_1u_2, \overline{u_2}, u_2u_3) \in A^* \times \overline{A}^* \times A^*$ is also drawn as a birooted word tree




where the dangling input arrow appears on the right of the dangling output arrow. A context tile of the form $u = (u_1, 1, u_3) \in A^* \times 1 \times A^*$ is then drawn as follows:

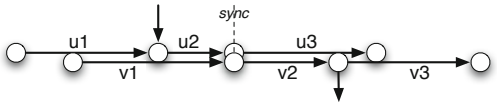



The *domain* of a tile $u = (u_1, u_2, u_3)$ is the reduced form of $u_1u_2u_3$ (always a word of $A^*$). Its *root* is the word $u_2$.

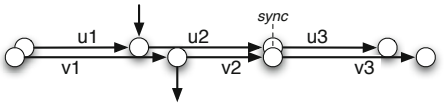
Sets $T_A$, $T_A^+$, $T_A^-$ and $C_A$ will respectively denote the set of tiles, the set of positive tiles, the set of negative tiles and the set of context tiles over $A$.

**A Product of Tiles**

Intuitively, the sequential product of two tiles is their superposition in such a way that the end of the root of the first tile coincides with the beginning of the root of the second tile; the superposition requires pattern-matching conditions to the left and to the right of the synchronization point. When both tiles are positive, this can be drawn as follows:




The product can be extended to arbitrary tiles, as illustrated by the following figure (positive $u$ and negative $v$):




Formally, we extend the set $T_A$ with a zero tile to obtain $T_A^0 = T_A + \{0\}$. The sequential product of two non-zero tiles $u = (u_1, u_2, u_3)$ and $v = (v_1, v_2, v_3)$ is defined as

$$u.v = ((u_1u_2 \vee_s v_1)\overline{u_2}, u_2v_2, \overline{v_2}(u_3 \vee_p v_2v_3))$$

when both $u_1u_2 \vee_s v_1 \neq 0$ and $u_3 \vee_p v_2v_3 \neq 0$, and $u.v = 0$ otherwise, where, in this formula, words in $(A + \overline{A})^*$ are implicitly considered as reduced elements of $FG(A)$. We also let $u.0 = 0.u = 0$ for every $u \in T_A^0$.

*Example 1.* Let $a$, $b$, $c$ and $d \in A$ be distinct letters. Then $(a, b, c).(b, c, d) = (a, bc, d)$ whereas $(a, b, c).(a, c, d) = 0$. In the latter case, the left matching constraint is violated because $a \neq b$.

Set $T_A^0$ equipped with the above sequential product is a monoid. But the proof of that fact is postponed to the proof that it is even isomorphic to the monoid of McAlister.

**The Inverse Monoid of McAlister**

McAlister's monoid is defined as the quotient of the free inverse monoid $FIM(A)$ by the ideal $\perp$ of all tiles which are non unidirectional or not linear.

More precisely, following Munn's result [10], elements of $FIM(A)$ are seen as birooted word trees, i.e. pairs $(u, P)$ where $P$ is a non empty finite and prefix-closed subset of (reduced elements of) the free group $FG(A)$ generated by $A$, with $u \in P$. The product of two birooted trees $(u, P)$ and $(v, Q)$ is defined by $(u, P).(v, Q) = (uv, P \cup uQ)$.

A birooted word tree $(u, P)$ is said *unidirectional* when $P \subseteq A^* + \overline{A}^*$, and *linear* when both $P \cap A^*$ and $P \cap \overline{A}^*$ are totally ordered by the prefix order.

It is straightforward that the set $\perp$ of non-unidirectional or non-linear birooted trees is an ideal.

The monoid of McAlister is then defined as the Rees quotient $FIM(A)/\perp$, i.e. the monoid obtained from $FIM(A)$ by merging into a single zero all elements of $\perp$. In that monoid, given two linear and unidirectional birooted word trees $(u, P)$ and $(v, Q)$, the product of these two tiles is defined to be $(uv, P \cup uQ)$ as in $FIM(A)$ when the resulting birooted tree is linear and unidirectional, and 0 otherwise.

**Theorem 1.** *The set $T_A^0$ equipped with the sequential product is a monoid isomorphic to the monoid of McAlister.*

*Proof.* (sketch of) For all non zero tile $u = (u_1, u_2, u_3) \in T_A$ let $t_u = (u_2, P_u)$ be the resulting birooted tree defined by $P_u = \{x \in \overline{A}^* : x \leq_p \overline{u}_1\} \cup \{x \in A^* : x \leq_p u_2u_3\}$.

We observe that $t_u$ is a well-defined unidirectional and linear birooted tree. Indeed, when $u$ is a positive tile, we have $u_2 \leq u_2u_3$ hence $u_2 \in P_u$. When $u$ is a negative tile, i.e. with $u_2 \in \overline{A}^*$ we have both $u_1u_2$ and $u_2u_3 \in A^*$ and thus $u_2 \leq_p \overline{u}_1$ hence $u_2 \in P_u$.

We conclude then by showing that the mapping $\varphi : T_A^0 \to FIM(A)/\perp$ defined by $\varphi(0) = 0$ and for any non-zero tile $u = (u_1, u_2, u_3) \in T_A$, $\varphi(u) = (u_2, P_u)$ is an isomorphism.

First, it is easy to check that it is a bijection. In fact, given a linear and unidirectional tile $(u, P)$ one check that $\varphi^{-1}((u, P))$ is the tile $(u_1, u_2, u_3)$ defined by $u_2 = u$, $u_1 = \bigvee_s \overline{P \cap \overline{A}^*}$ and $u_3 = \overline{u}_2 \bigvee_p P \cap A^*$.

It remains to show $\varphi$ preserves products. For this, it is enough to check that for any two non zero tiles $u = (u_1, u_2, u_3)$ and $v = (v_1, v_2, v_3) \in T_A$, one indeed has $t_{u.v} = (u_2v_2, P_u \cup u_2P_v)$ which essentially follows from definitions. □

**Corollary 2.** *Monoid $T_A^0$ is an inverse monoid with (pseudo) inverses given by $0^{-1} = 0$ and for every non zero tile $u = (u_1, u_2, u_3) \in T_A$, $u^{-1} = (u_1u_2, \overline{u}_2, u_2u_3)$.*

*Proof.* (sketch of) For every $u = (u_1, u_2, u_3) \in T_A$, given $t_u = (u_2, P_u)$, this amounts to check that $t_{u^{-1}} = (t_u)^{-1} = (\overline{u}_2, u_2P_u)$ and thus this just amounts to check that $u_2P_u = P_{u^{-1}}$. □

An immediate property worth being mentioned:

**Lemma 3.** *The mapping $u \mapsto (1, u, 1)$ from $A^*$ to $T_A$ is a one-to-one morphism and the monoid $T_A^0$ is finitely generated from the image of the set of letters $A$ and their inverses.*

In other words, the free monoid $A^*$ can be seen as a submonoid of $T_A^0$. In the remainder of the text we may use the same notation for words of $A^*$ and their images in $T_A^0$.

**Remark.** One can also observe that positive (resp. negative) tiles extended with 0 form a submonoid $T_A^{+0}$ (resp. $T_A^{-0}$) of $T_A^0$. However, this submonoid *is not* finitely generated. One need extra operators.

More precisely, for every $u = (u_1, u_2, u_3) \in T_A^+$, let $u_L = u^{-1}u = (u_1u_2, 1, u_3)$, the canonical *left context tile* associated to tile $u$ and let $u_R = uu^{-1} = (u_1, 1, u_2u_3)$, the canonical *right context tile* associated to tile $u$.

By construction, we have $u_Ru = uu_L = u$ for every $u \in T_A$. Moreover, submonoid $T_A^{+0}$ (resp. $T_A^{-0}$) is finitely generated by (embeddings of) $A$ (resp. $\overline{A}$), product and left and right context operators.

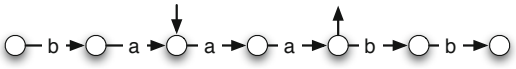
## 3 MSO-Definable and Regular Languages of Tiles

We consider in this section the class MSO of languages of tiles definable by means of monadic second order formulae.

**MSO Definability**

We need FO-models for positive tiles. For this, we use a typical encoding of words into FO-structures that amounts to encode each letter $a \in A$ as a relation between elements of the domain. This way, there is no need of end markers and the empty word is simply modeled by the structure with singleton domain and empty relations. Then we raise models of words to models of tiles just by marking (as pictured in birooted trees) entry and exit points.

For instance, the triple $u = (ba, aa, bb)$ is modeled as indicated by the following picture




where, as before, a dangling input arrow marks the entry point and a dangling output arrow marks the exit point.

The model of a tile $u$ is denoted by $t_u$. The associated domain of its underlying FO-structure is written $dom(t_u)$, the entry point in written $in(t_u)$ and the exit point is written $out(t_u)$.

A language $L \subseteq T_A$ is MSO definable when there is a MSO formula of the form $\varphi_L(U, x, y)$ where $U$ is a set variable and $x$ and $y$ are two FO-variables such that, for all $t \in T_A$, $t \in L$ if and only if $t \models \varphi_L(dom(t), in(t), out(t))$.

**A Word Congruence for Languages of Tiles**

We aim at achieving a simple characterization of MSO definable language of tiles. For this purpose, we first define a notion of congruence relation over $A^*$ which is defined for all language of tiles. It occurs that this congruence is of finite index if and only if the language of tile is definable in MSO.

Given a language $L \subseteq T_A$ on non zero tiles, we define the *word* congruence $\simeq_L$ associated to $L$ that tells when two words can be replaced one by the other in any tile without altering membership in $L$.

Formally, $\simeq_L$ is the least relation over words such that, for all $u_0$ and $v_0 \in A^*$, $u_0 \simeq_L v_0$ when for all $w_0$, $w_2$, $w_3$ and $w_4 \in A^*$, if $u = (w_1u_0w_2, w_3, w_4)$ and $v = (w_1v_0w_2, w_3, w_4)$, or if $u = (w_1, w_2u_0w_3, w_4)$ and $v = (w_1, w_2v_0w_3, w_4)$, or

if $u = (w_1, w_2, w_3u_3w_4)$ and $v = (w_0, w_2, w_3v_3w_4)$ then $u \in L \Leftrightarrow v \in L$ and $u^{-1} \in L \Leftrightarrow v^{-1} \in L$.

Relation $\simeq_L$ is a congruence on words. For every for $u \in A^*$, let $[u]_L$ be the equivalence class of $u$ defined by $[u]_L = \{v \in A^* : u \simeq_L v\}$.

**Theorem 4.** *For every language $L \subseteq T_A$ of non zero tiles:*

$$L = \Sigma_{(u_1,u_2,u_3)\in L\cap T_A^+}[u_1]_L \times [u_2]_L \times [u_3]_L \\ +\Sigma_{(u_1,u_2,u_3)^{-1}\in L\cap T_A^-}([u_1]_L \times [u_2]_L \times [u_3]_L)^{-1}$$

*Moreover, $L$ is definable in MSO if and only if relation $\simeq_L$ is of finite index.*

*Proof.* The first statement is an immediate consequence of the definition of $\simeq_L$. Moreover, if $\simeq_L$ is of finite index, then this sum is finite, and any language of the form $[w]_L \subseteq A^*$ with $w \in A^*$ is rational henceforth MSO definable. It follows that $L$ is thus definable in MSO.

Conversely, assume $L$ is definable in MSO. Given $L^+ = L \cap T_A^+$ and $L^- = L \cap T_A^-$, we observe that both $L^+$ and $(L^-)^{-1} \subseteq A^* \times A^* \times A^*$ can be encoded into languages of words $M^+$ and $M^- \subseteq A_L^*A_C^*A_R^*$ where $A_L$, $A_C$ and $A_R$ are three disjoint copies of the alphabet $A$ for encoding the left, center and right elements of each tiles.

Now, since $L$ is definable in MSO, so are $L^+$ and $L^-$ and thus, their encoding $M^+$ and $M^-$ are also definable in MSO. By Büchi theorem, this means that they are regular and thus their syntactic congruences $\simeq_{M^+}$ and $\simeq_{M^-}$ are of finite index. This implies that $\simeq_L$ is also of finite index. Indeed, for all word $u$ and $v \in A^*$, we have $u \simeq_L v$ if and only if $u_X \simeq_{M^+} v_X$ and $u_X \simeq_{M^-} v_X$ for $X$ being $L$, $C$ or $R$ and with $w_X$ denoting the renaming of any word $w \in A^*$ in the copy alphabet $A_X$. □

**Extended Regular Languages**

For every langage $L$ and $M \subseteq T_A$ of *non zero* tiles, let $L + M = L \cup M$, $L.M = \{u.v \in T_A : u \in L, v \in M, u.v \neq 0\}$, $L^* = \bigcup_{k\geq 0} L^k$, $L^{-1} = \{u^{-1} \in T_A : u \in L\}$ and $L^C = \{u \in T_A : u \in L \cap C_A\}$ (called context projection).

**Theorem 5.** *For every languages $L$ and $M \subseteq T_A$ on non zero tiles, if $L$ and $M$ are MSO definable then so are $L + M$, $L.M$, $L^*$, $L^{-1}$ and $L^C$.*

*Proof.* Let $\varphi_L(U, x, y)$ and $\varphi_M(U, x, y)$ be two formulae defining respectively the language of tiles $L$ and $M$. We assume that these formulae also check that both $x$ and $y$ belongs to $U$ that, moreover, $U$ is connected.

Case of $L + M$: take $\psi(U, x, y)$ defined by $\varphi_L(U, x, y) \vee \varphi_M(U, x, y)$.

Case of $L.M$: take $\psi(U, x, y)$ stating that there exist two sets $X$ and $Y$ such that $U = X \cup Y$ and there is $z$ such that both $\varphi_L(X, x, z)$ and $\varphi_M(Y, z, y)$ hold.

Case of $L^*$: in order to define $\varphi(U, x, y)$, the main idea is to consider the reflexive transitive closure $R^+(x, y)$ of the binary relation $R(x_1, x_2)$ defined by

$\exists X \varphi_L(X, x_1, x_2)$; one must take care, however, that set $U$ is completely covered by (sub)tiles' domains; this is equivalent to the fact, as domains necessarily overlap, that each extremity (left most or right most element) of the domain $U$ belongs to one of these sets $X$ at least. This is easily encoded by a disjunction of the three possible cases: extremities are reached in a single intermediate tile, left extremity is reached first or right extremity is reached first.

Case of $L^{-1}$: take $\psi(U, x, y)$ defined by $\varphi_L(U, y, x)$.

Case of $L^C$: take $\psi(U, x, y)$ defined by $(x = y) \wedge \varphi_L(U, x, y)$. □

Let then $REG$ (resp. $REG_C$) be the class of regular (resp. extended regular) languages of tiles that is the class of languages definable by means of finite sets of non zero tiles, sum, product, iterated product (or star), inverse and context projection.

Since finite tile languages are indeed definable in MSO we can state the following corollary of Theorem 5:

**Corollary 6.** *Extended regular languages of tiles are MSO definable.*

Moreover, since for every regular languages $L$, $M$ and $R \subseteq A^*$, one has $L \times C \times R = (L^{-1}.L)^C.M.(R.R^{-1})^C$ where, on the right side, words are embedded into tiles as in Lemma 3, we can also state immediate corollary of Theorem 4:

**Corollary 7.** *MSO definable languages of tiles are extended regular.*

In other words, $REG_C = MSO$.

**Remark.** Observe that, in the above proof, we use the fact that for every languages $L \subseteq A^*$ of (embeddings of) words into tiles, $(L.L^{-1})^C = \{uu^{-1} : u \in L\}$ and $(L^{-1}.L)^C = \{u^{-1}u : u \in L\}$. This property is not true for arbitrary languages of $T_A$.

## 4 Recognizable Languages of Tiles

In this section, we consider languages of non zero tiles that are (up to adding the zero tile) recognizable in the algebraic sense. Although the theory of tiles can be seen as part of the theory of inverse monoid, the results we obtain rather differ from the former studies of languages of words recognized by finite inverse monoids [9] or free inverse monoid languages [12]. Morphisms from $T_A^0$ (or even $T_A^{+0}$) to arbitrary monoids turns out to be even more constrained than morphisms from $A^*$ to finite inverse monoids.

### A Non Recognizable Language of Tiles

The next result, negative, tells us that rather simple (MSO definable) languages of tiles are not recognizable.

**Lemma 8.** *Language* $L = \{(ba^m, a^n, a^p) \in T_A : m, n, p \in I\!N\}$ *with* $a$ *and* $b$ *two distinct letters, is not recognizable.*

*Proof.* We prove that the syntactical congruence $\simeq_L$ (and not the word congruence defined above) associated to $L$ in $T_A^0$ is of infinite index.

For all $m \in \mathbb{N}$, let $u_m$ be the tile $u_m = (ba^m, 1, 1)$. It is an easy observation that for all $m$ and $n \in \mathbb{N}$, $u_m \simeq_L u_n$ if and only if $m = n$ hence the claim. Indeed, for all $k \in \mathbb{N}$ let $v_k = (a^k, 1, 1)$. We have for any given $m \in \mathbb{N}$, $u_m v_k \in L$ if and only if $k \leq m$. Now if for some $m$ and $n \in \mathbb{N}$ one has $u_m \simeq_L u_n$ then for all $k \in \mathbb{N}$, $u_m v_k \in L$ if and only if $u_n v_k \in L$. It follows that, for all $k$, $k \leq m$ if and only if $k \leq n$, hence $m = n$. □

Since $L = b^{-1}(a^{-1})^* ba^*$, it belongs to the class *REG*. Since recognizable languages are definable by regular expressions [3], we have the strict inclusion:

**Corollary 9.** *REC* $\subset$ *REG*

**A (Non-Trivial) Recognizable Language**

Before studying in the next section recognizable languages in full generality, we provide in this section a non trivial example of such a language. It illustrates the main characteristic of all recognizable languages of tiles : a strong link with tiles's cover of periodic bi-infinite words.

Building such an example essentially amounts to provide a (onto) monoid morphism from $T_A^0$ onto some non-trivial finite (inverse) monoid $M$. Here, the main idea is to type tiles, by means of a monoid morphism, according to their capacity to cover the bi-infinite word ${}^{\omega}(ab)(ab)^{\omega}$ with $a$ and $b$ two distinct letters.

In order to do so, let $M = \{0, 1, (a, 1, b), (b, 1, a), (b, a, b), (a, b, a)\}$ with product $\odot$ defined as expected for 0 and 1 and defined according to the following product table:

| $\odot$ | $(a,1,b)$ | $(b,1,a)$ | $(b,a,b)$ | $(a,b,a)$ |
|---|---|---|---|---|
| $(a,1,b)$ | $(a,1,b)$ | 0 | 0 | $(a,b,a)$ |
| $(b,1,a)$ | 0 | $(b,1,a)$ | $(b,a,b)$ | 0 |
| $(b,a,b)$ | $(b,a,b)$ | 0 | 0 | $(b,1,a)$ |
| $(a,b,a)$ | 0 | $(a,b,a)$ | $(a,1,b)$ | 0 |

**Lemma 10.** *Monoid* $(M, \odot)$ *is an inverse monoid.*

*Proof.* We easily check that product $\odot$ is associative hence $M$ is a monoid. Given $E(M) = \{0, 1, (a, 1, b), (b, 1, a)\}$ the set of idempotents of $S$, the commutation of idempotents immediately follows from unique non trivial case $(a, 1, b) \odot (b, 1, a) = (b, 1, a) \odot (a, 1, b) = 0$. Last, we check that $(a, b, a) \odot (b, a, b) \odot (a, b, a) = (a, b, a)$ and $(b, a, b) \odot (a, b, a) \odot (b, a, b) = (b, a, b)$. It follows that $(a, b, a)^{-1} = (b, a, b)$ and $(b, a, b)^{-1} = (a, b, a)$. All other element is idempotent and thus self-inverse. □

The expected monoid morphism $\varphi : T_A^0 \to M$ is then defined by $\varphi(0) = 0$, $\varphi(1) = 1$ and for all $(u, v, w) \in T_A^0$ such that $uvw \neq 1$, $\varphi(u, v, w) = 0$ when $uvw$ is not a factor of $(ab)^{\omega}$ and, otherwise, when $u$ is a positive tile:

1. $\varphi(u,v,w) = (a,1,b)$ when $|v|$ is even with $a \leq_s u$,$b \leq_p v$, $a \leq_s v$ or $b \leq_p w$,
2. $\varphi(u,v,w) = (b,1,a)$ when $|v|$ is even with $b \leq_s u$,$a \leq_p v$, $b \leq_s v$ or $a \leq_p w$,
3. $\varphi(u,v,w) = (b,a,b)$ when $|v|$ is odd with $a \leq_p v$,
4. $\varphi(u,v,w) = (a,b,a)$ when $|v|$ is odd with $b \leq_p v$,

and $\varphi(u,v,w) = (\varphi(uv,\overline{v},vw))^{-1}$ when $(u,v,w)$ is a negative tile.

**Theorem 11.** *The mapping $\varphi : T_A^0 \to M$ is a onto morphism.*

*Proof.* This follows from the fact that, for all $u$ and $v \in T_A^0$, $\varphi(u) \odot \varphi(v) = \varphi(uv) = \varphi(\varphi(u)\varphi(v))$. □

Given $L_S = (ab)^* + b(ab)^*$, given $L_C = (ab)^*$, given $L_P = (ab)^* + (ab)^*a$, this theorem says, in particular, that the non trivial tile language $L_S \times L_C \times L_P - 1$ is recognizable since it equals $\varphi^{-1}((b,1,a))$.

**More on Recognizable Languages of Tiles**

We conclude our study by showing that recognizable languages of tiles are essentially generalization of the example described above.

We say that $L \subseteq T_A$ is recognizable when there is a monoid morphism $\varphi : T_A^0 \to M$ with finite monoid $M$ such that $L = \varphi^{-1}(\varphi(L)) - 0$.

Since we can always restrict $M$ to $\varphi(T_A^0)$ and $\varphi(0)$ is a zero in the submonoid $\varphi(T_A^0)$, we assume, without loss of generality, that $M = \varphi(T_a^0)$ with $\varphi(0) = 0$. Now, by complement, understanding the structure of languages of non zero tiles recognizable by $\varphi$ amounts to understand the structure of languages of the form $\varphi^{-1}(s)$ for all non-zero element $s \in S$.

Moreover, we can restrict our attention to recognizable languages of positive tiles. It can indeed by shown that $\varphi(T_A^0)$ is necessarily an inverse monoid and, for every $s \in M - 0$, $\varphi^{-1}(s) = \varphi^{-1}(s) \cap T_A^+ + (\varphi^{-1}(s^{-1}) \cap T_A^+)^{-1}$.

In the theorem below, we even consider the more difficult case of a onto monoid morphism $\varphi : T_A^{+0} \to M$ from the submonoid of positive tiles to a finite monoid $M$. However, in order to avoid undesirable effect due to the fact that, contrary to $T_A^0$, $T_A^{+0}$ is not finitely generated, we assume that $\varphi$ is *safe* in the sense, for every $u \in T_A^+$, if $\varphi(u) \neq 0$ then both $\varphi_L(u) = \varphi(u^{-1}u) \neq 0$ and $\varphi_R(u) = \varphi(uu^{-1}) \neq 0$. One can check that an onto morphism from all tiles $T_A^0$ to a monoid $M$ is always safe.

**Theorem 12.** *Let $\varphi : T_A^{+0} \to M$ be a safe monoid morphism. Let $s \in M$ be a non-zero element. Let $L_s = \varphi^{-1}(s)$. Then either $L_s$ is finite with $L_s \subseteq \{u \in T_A^+ : v \leq u\}$ for some (non zero) $v \in T_A^+$, or there are two finite words $x \in A^*$ and $y \in A^+$ and some integer $k \geq 0$ such that, $L_s$ is a* co-finite subset *of one of the following set of non-zero tiles:*

1. *$S(^\omega(xy)) \times x \times P(w)$ for some finite $w <_p (yx)^\omega$,*
2. *$S(w) \times x \times P((yx)^\omega)$ for some finite $w <_s {}^\omega(xy)$,*
3. *$S(^\omega(xy)) \times x \times P((xy)^\omega)$,*
4. *or $S(^\omega(xy)) \times x(yx)^k(yx)^* \times P((yx)^\omega)$,*

*with, for all* $w \in A^* + {}^\omega A$, $S(w) = \{z \in A^* : z \leq_s w\}$, *i.e. the set of suffix of* $w$, *and for all* $w \in A^* + A^\omega$, $P(w) = \{z \in A^* : z \leq_p w\}$, *i.e. the set of prefix of* $w$.

*In particular, domains of tiles of* $L_s$ *are factors of the bi-infinite periodic word* ${}^\omega(xy)x(yx)^\omega$.

*Proof.* The rest of this section is dedicated to the proof of this theorem. In order to do so, we first prove several closure properties of $L_s$.

**Lemma 13.** *For all* $u = (u_1, u_2, u_3) \in L_s$ *and* $v = (v_1, v_2, v_3) \in L_s$, $(u_1 \vee_s v_1, u_2, u_3) \in L_s$ *and* $(u_1, u_2, u_3 \vee_p v_3) \in L_s$.

*Proof.* We know that $(v_1)_L v = v$ hence $(v_1)u \in L_s$ since $\varphi((v_1)_L v) = \varphi_L(v_1)\varphi(v) = \varphi_L(v_1)\varphi(u) = \varphi((v_1)_L u)$. Moreover, $0 \notin L_s$ hence $(v_1)_L u = (u_1 \vee_s v_1, u_2, u_3)$ with $u_1 \vee_S v_1 \neq 0$. Symmetrical arguments prove the prefix case. □

**Lemma 14.** *For all* $u = (u_1, u_2, u_3)$ *and* $v = (v_1, v_2, v_3) \in L_s$ *such that* $|u_2| \leq |v_2|$, *either* $u_2 = v_2$ *or there exists* $x \in A^*$, $y \in A^+$ *and* $k \geq 0$ *such that* $u_2 = x(yx)^k$, $v_2 = x(yx)^{k+1}$. *In that latter case, there is* $u' \in L_s$ *such that* $u'((xy)_C)^* \subseteq L_s$.

*Proof.* Let $u$ and $v$ as above. We have $(v_2)_R v (v_2)_L = v$ hence, by a similar argument as in Lemma 13, $(v_2)_R u (v_2)_L \in L_s$. Since $|u_2 \leq |v_2|$ this means that $u_2 \leq_p v_2$ and $u_2 \leq_s v_2$, i.e. *roots of elements of* $L_s$ *are totally by both prefix and suffix.*

In the case $u_2 \neq v_2$ this means $v_2 = wu_2 = u_2 w'$ for $w$ and $w' \in A^+$. Let then $k \geq 0$ be the greatest integer such that $|w^k| < |u_2|$.

When $k = 0$, this means $w = u_2 y$ for some $y \in A^*$ and we take $x = u_2$. Otherwise, by a simple inductive argument over $k$, this means $u_2 = w^k x$ for some $x \in A^*$ with $|x| < |w|$. In that latter case, we have $v_2 = w^{k+1}x = w^k x w'$. Since $|w'| = |w|$ it follows that $w' = yx$ for some $y \in A^+$ and thus $w = xy$. In all cases, $u_2 = x(yx)^k$ and $v_2 = x(yx)^{k+1}$.

By applying Lemma 13, with $v_1' = u_1 \vee_s v_1$, $v_3' = u_3 \vee_p v_3$ and $v' = (v_1', v_2, v_3')$, we have $v' \in L_s$. But we known that $(v_2 v_3')_L v' = v'$ hence we also have $(v_2 v_3')_L u \in L_s$. By applying product definition, this means that $u' = (v_1', u_2, yxv_3') \in L_s$ hence $v' = u'(yx)_C \in L_s$ hence, by an immediate pumping argument, $u'((yx)_C)^* \subseteq L_s$. □

Lemma 14 already proves the finite case of Theorem 12. We assume now $L_s$ is infinite.

In the case a single root appears in elements of $L_s$, for all $u = (u_1, x, u_3) \in L_s$ we have $(u_1)_L u (u_3)_L = u$ hence, because $s \neq 0$, $\varphi_L(u_1) \neq 0$ and $\varphi_L(u_3) \neq 0$.

By *safety assumption*, this means that $\varphi_C(u_1) \neq 0$ and $\varphi_C(u_3) \neq 0$. Since $M$ is *finite*, while $L_s$ is finite, this means that two distinct constraints have same images by $\varphi_C$. Applying Lemma 14 this implies all domains of tiles of $L_s$ are factors of the same periodic bi-infinite word of the form ${}^\omega(xy)x(yx)^\omega$ for some $x$ and $y \in A^*$ with $xy \neq 0$.

Depending on the case elements of $L_s$ have infinitely many left constraints, right constraints or both left and right constraints, we conclude by showing

that for all $v_1 \leq_s {}^{\omega}(xy)$ and $v_3 \leq_p (yx)^{\omega}$ there is $v \in L_s$ such that either $(v_1)_L v$, $v(v_3)_R$ or $(v_1)_L v(v_3)_R \in L_s$ hence, given a given fixed $u = (u_1, x, u_3) \in L_S$, either $(v_1)_L u$, $u(v_3)_R$ or $(v_1)_L u(v_3)_R \in L_s$. This proves the co-finiteness inclusion in case 1, 2 and 3 in Theorem 12.

In the case two distinct roots (hence infinitely many) appears in tiles of $L_s$. Applying Lemma 14 there is $x \in A^*$, $y \in A^+$ and $k \geq 0$ with $u = (u_1, u_2, xyu_3)$ such that $u((xy)_C)^* \subseteq L_s$. By choosing properly the initial $u$ and $v$ in Lemma 14, we may assume that $u_2 = x(yx)^k$ is the least root of elements of $L_s$ and that $y(yx)^{k+1}$ is the second least. Then we claim that all roots of elements of $L_s$ belongs to $x(yx)^k(yx)^*$.

Indeed, let then $(v_1, v_2, v_3) \in L_s$ and let $m \geq 0$ be the unique integer such that $v_2 = x(yx)^{m+k}x'$ with $x' <_p x$ hence $x = x'y'$ for some $y' \in A^+$. By an argument similar with the argument in the proof of Lemma 14, we can show that there is $u' = (v_1', x(yx)^{m+k}, x'v_3') \in L_s$ with $v' = (v_1', v_2, v_3') \in L_s$ henceforth with $v' = u'x'_C \in L_s$. But this also means that $ux'_C = (u_1, u_2x', y'yu_3) \in L_s$ which, by minimality of $u_2xy$ as second least roots, forces $x'$ to be equal to 1. This proves the inclusion stated in case 4 in Theorem 12.

Co-finiteness of the inclusion follows from the finiteness of $M$ with arguments similar to the arguments for case 3 above. □

**Remark.** As a conclusion, let us mention that, for languages of positive tiles, a weakening of the notion of recognizability by means of premorphisms instead of morphisms has been shown to capture MSO [4]. However, how such a more expressive notion can be relevantly applied to languages of arbitrary tiles is still an open problem.

## References


1. de Almeida, F.S.: Algebraic Aspects of Tiling Semigroups. PhD thesis, Universidade de Lisboa, Faculdade de Ciências Departamento de Matemática (2010)
2. Berthaut, F., Janin, D., Martin, B.: Advanced synchronization of audio or symbolic musical patterns. In: Sixth IEEE International Conference on Semantic Computing, pp. 202–209. IEEE Society Press (2012)
3. Dicky, A., Janin, D.: Two-way automata and regular languages of overlapping tiles. Technical Report RR-1463-12, LaBRI, Université de Bordeaux (2012)
4. Janin, D.: Quasi-recognizable vs MSO Definable Languages of One-Dimensional Overlapping Tiles. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) MFCS 2012. LNCS, vol. 7464, pp. 516–528. Springer, Heidelberg (2012)
5. Janin, D.: Vers une modélisation combinatoire des structures rythmiques simples de la musique. Revue Francophone d'Informatique Musicale (RFIM), 2 (to appear, 2012)
6. Kellendonk, J.: The local structure of tilings and their integer group of coinvariants. Comm. Math. Phys. 187, 115–157 (1997)
7. Lawson, M.V.: Inverse Semigroups: The theory of partial symmetries. World Scientific (1998)
8. Lawson, M.V.: McAlister semigroups. Journal of Algebra 202(1), 276–294 (1998)

9. Margolis, S.W., Pin, J.-E.: Languages and Inverse Semigroups. In: Paredaens, J. (ed.) ICALP 1984. LNCS, vol. 172, pp. 337–346. Springer, Heidelberg (1984)
10. Munn, W.D.: Free inverse semigroups. Proceeedings of the London Mathematical Society 29(3), 385–404 (1974)
11. Pécuchet, J.-P.: Automates boustrophedon, semi-groupe de birget et monoide inversif libre. ITA 19(1), 71–100 (1985)
12. Silva, P.V.: On free inverse monoid languages. ITA 30(4), 349–378 (1996)

# An Exact Algorithm to Check the Existence of (Elementary) Paths and a Generalisation of the Cut Problem in Graphs with Forbidden Transitions

Mamadou Moustapha Kanté*, Christian Laforest**, and Benjamin Momège***

Clermont-Université, Université Blaise Pascal, LIMOS, CNRS, France
{mamadou.kante,laforest,momege}@isima.fr

**Abstract.** A *graph with forbidden transitions* is a pair $(G, F_G)$ where $G := (V_G, E_G)$ is a graph and $F_G$ is a subset of the set $\{(\{y, x\}, \{x, z\}) \in E_G^2\}$. A *path* in a graph with forbidden transitions $(G, F_G)$ is a path in $G$ such that each pair $(\{y, x\}, \{x, z\})$ of consecutive edges does not belong to $F_G$. It is shown in [S. Szeider, Finding paths in graphs avoiding forbidden transitions, DAM 126] that the problem of deciding the existence of a path between two vertices in a graph with forbidden transitions is NP-complete. We give an exact exponential time algorithm that decides in time $O(2^n \cdot n^5 \cdot \log(n))$ whether there exists a path between two vertices of a given $n$-vertex graph with forbidden transitions. We also investigate a natural extension of the *minimum cut* problem: we give a polynomial time algorithm that computes a set of forbidden transitions of minimum size that disconnects two given vertices (while in a minimum cut problem we are seeking for a minimum number of edges that disconnect the two vertices). The polynomial time algorithm for that second problem is obtained via a reduction to a standard minimum cut problem in an associated *allowed line graph*.

## 1 Introduction

Algorithms manipulating graphs are often used to solve concrete situations in many applied fields. Finding a path between two given points/vertices is a fundamental basic tool that often serves as subroutine in many more complex algorithms and software (for example in flows (improving paths between a source and a sink), in scheduling (notion of constraint and critical path), in network for routing operations, etc.). Several well-known polynomial time algorithms are able to do this task: DFS, BFS (to find shortest paths in unweighted graphs), Dijkstra (for weighted graphs). They are widely available in software packages

* M.M. Kanté is supported by the French Agency for Research under the DORSO project.
** Ch. Laforest is supported by the French Agency for Research under the DEFIS program TODO, ANR-09- EMER-010.
*** B. Momège has a PhD grant from CNRS and région Auvergne.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 257–267, 2013.

(like Maple and Mathematica[1] for example) and are taught in most of first level computer science or engineering courses all around the world (see a reference book on algorithms like [6]).

A path (simple or elementary) $P$ in a graph $G$ is just a list of consecutive (incident) edges (or arcs) of $G$ between a given first vertex $s$ and a final vertex $t$. To construct such a path $P$ given $G$, $s$ and $t$, the classical algorithms use all the potentiality of the graph: namely, when a given intermediate vertex $u$ is reached, there is no restrictions on the following vertex that can be reached: *any* neighbour of $u$. Of course, BFS for example does not explore an already explored (and marked) vertex $w$ but, as $w$ is a neighbour of $u$, this possibility is taken into account among all possibilities.

This is a strong hypothesis in several real applications. Indeed, in some concrete networks, it is *not* possible, coming from a point $a$ towards a point $b$ to continue towards point $c$. For example in several large streets of cities, it is forbidden to turn left at point $b$ (towards point $c$) and to cross a road (if one come from point $a$ preceding $b$). Many such transits are forbidden in all the countries; several other restrictions exist (no "U" turn for example).

All these concrete limitations are due to the system modelled by graphs (routes, systems of production, etc.) in which all the paths are *not* possible because they have a local transition that is not allowed.

These *forbidden* transitions must be added into the knowledge of the graph. Then the algorithms that are supposed to construct paths between two vertices must take them into account. The hard part of the work starts at this point. Indeed, unlike the classical situation where there are several algorithms mentioned above, adding forbidden transitions strongly increases the complexity of the situation: In [8], Szeider shows that knowing whether there exists a path between two nodes avoiding forbidden transitions is NP-Complete while it is polynomial without these constraints (see [6]).

Several other studies have been done on graphs with forbidden transitions. For example, in [3] the authors want to find an Eulerian path in a graph representing biological data (from DNA sequencing) where all transitions between these biological elements are not allowed. Later, in [4], this practical problem serves as a motivation for a graph theoretic study. The authors prove, among other results, that finding an Eulerian path is NP-Complete when the graph contains forbidden transitions. In [7] the problem of finding two-factors[2] is considered. The author gives conditions on the type of transitions under which deciding whether there is a two-factor avoiding forbidden transitions in $G$ is polynomial or NP-complete. In [1,9] the authors investigate a more general form of transitions; they propose polynomial time algorithms to find a shortest path avoiding forbidden *subpaths*. However, their paths are not elementary (we will call *walks* these objects later). This is a huge difference with the study of [8] where Szeider consider elementary paths.

[1] See `http://www.maplesoft.com/` and `http://www.wolfram.com/`

[2] A subgraph such that for any vertex its in-degree and its out-degree is exactly one.

**Summary.** In Section 2 of our paper, we give preliminary definitions, notations and concepts. In Section 3 we propose an exponential time algorithm based on inclusion-exclusion principle having a complexity of $O(2^n \cdot n^5 \cdot \log(n))$ (where $n$ is the number of vertices) to decide if the graph contains or not a (elementary) path between two given vertices avoiding forbidden transitions. In Section 4 we investigate an equivalent problem to the minimum cut problem: we propose a polynomial time algorithm to compute the minimum number of allowed transitions to transform into forbidden ones to disconnect a given vertex $s$ to a given vertex $t$ (instead of cutting edges as in the original well-known cut problem).

## 2 Preliminaries

In this paper we consider only simple graphs. If $A$ and $B$ are two sets, $A \backslash B$ denotes the set $\{x \in A \mid x \notin B\}$. The size of a set $A$ is denoted by $|A|$.

We refer to [5] for graph terminology not defined in this paper. The vertex-set of a graph $G$ (directed or not) is denoted by $V_G$ and its edge-set (or arc-set if it is directed) by $E_G$. An edge between two vertices $x$ and $y$ in an undirected graph $G$ is denoted by $\{x, y\}$ and an arc from $x \in V_G$ to $y \in V_G$ in a directed graph $G$ is denoted by $(x, y)$. For a vertex $x \in V_G$ we let $E_G(x)$ be the set of edges or arcs incident with $x$; the *degree* of $x$, defined as $|E_G(x)|$, is denoted by $d_G(x)$.

If $\mathcal{C}$ is a class of graphs, we denote by $\mathcal{C}^{\text{ind}}$ the class of graphs $\{H \mid H$ is an induced subgraph of some graph $G \in \mathcal{C}\}$. We denote by $P_3$, $K_3$, $2K_2$, $P_4$ and $L_4$ the undirected graphs depicted in Figure 1.

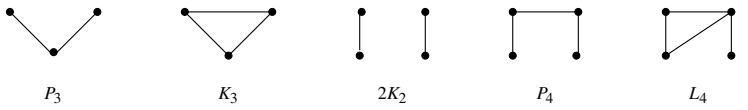



**Fig. 1.**

In the **undirected case**, a *graph with forbidden transitions* is a pair $(G, F_G)$ where $G$ is a graph and $F_G$ is a subset of the set $T_G := \{(\{y, x\}, \{x, z\}) \in E_G^2\}$ of *transitions* of $G$. Transitions of $F_G$ are called the *forbidden transitions* and the transitions of $A_G := T_G \setminus F_G$ are called the *allowed transitions*. We will also denote a transition $(\{y, x\}, \{x, z\})$ by the triplet $(y, x, z)$. If for all the transitions $(\{y, x\}, \{x, z\}) \in F_G$ we have $(\{y, x\}, \{x, z\}) \in F_G \Leftrightarrow (\{x, z\}, \{y, x\}) \in F_G$ the forbidden transitions are called *symmetric* and in this case we define for each vertex $x$ a *transition graph* $A_x$ with $V_{A_x} := E_G(x)$ and $E_{A_x} := \{\{e_i, e_j\} \subseteq V_{A_x} \mid (e_i, e_j) \in A_G\}$. The set $\mathcal{A}_G := \{A_x \mid x \in V_G\}$ is called the system of allowed transitions.

A *walk* between $s$ and $t$ or an $(s, t)$-walk in $(G, F_G)$ is a sequence of vertices $(s = x_1, x_2, \ldots, x_k = t)$ such that $\{x_i, x_{i+1}\} \in E_G$ for every $1 \leq i \leq k-1$ and $(\{x_{i-1}, x_i\}, \{x_i, x_{i+1}\}) \in A_G$ for every $2 \leq i \leq k-1$. Such a walk is called

a walk on $k$ vertices and it may also be represented by the sequence of edges $(\{x_1, x_2\}, \ldots, \{x_i, x_{i+1}\}, \ldots \{x_{k-1}, x_k\})$.

In the **directed case**, a *directed graph with forbidden transitions* is a pair $(G, F_G)$ where $G$ is a graph and $F_G$ is a subset of the set $T_G := \{((y, x), (x, z)) \in E_G^2\}$ of *transitions* of $G$. Transitions of $F_G$ are called the *forbidden transitions* and the transitions of $A_G := T_G \setminus F_G$ are called the *allowed transitions*. We will also denote a transition $((y, x), (x, z))$ by the triplet $(y, x, z)$. A *walk* from $s$ to $t$ or an $(s, t)$-walk in $(G, F_G)$ is a sequence of vertices $(s = x_1, x_2, \ldots, x_k = t)$ such that $(x_i, x_{i+1}) \in E_G$ for every $1 \leq i \leq k-1$ and $((x_{i-1}, x_i), (x_i, x_{i+1}))$ is a transition of $A_G$ for every $2 \leq i \leq k-1$. Such a walk is called a walk on $k$ vertices and it may also be represented by the sequence of arcs $((x_1, x_2), \ldots, (x_i, x_{i+1}), \ldots, (x_{k-1}, x_k))$.

In the two cases, a *shortest* $(s, t)$-walk is a walk for which $k$ is minimum. A *path* is a walk where each vertex appears once. If there is an $(s, t)$-walk in $G$ we say that $G$ is $(s, t)$-connected; $s$ and $t$ are called disconnected if there is no $(s, t)$-walk.

We can apply these definitions to a "classic" graph $G$ by noting that $G$ is a graph with forbidden transitions where $F_G = \emptyset$.

The *path problem* in graphs with forbidden transitions consists in, given a graph with forbidden transitions $(G, F_G)$, and two vertices $s$ and $t$, asking for the existence of an $(s, t)$-path.

Until the end of this section we consider only undirected graphs with symmetric forbidden transitions.

**Theorem 1 ([8]).** *Let $\mathcal{C}$ be a class of graphs closed under isomorphism and let $\mathcal{G}(\mathcal{C})$ be the set of graphs with symmetric forbidden transitions $(G, F_G)$ with $\mathcal{A}_G \subseteq \mathcal{C}$. The path problem is* NP*-complete in $\mathcal{G}(\mathcal{C})$ if $\mathcal{C}^{\text{ind}}$ contains at least one of the sets $\{K_3, 2K_2\}$, $\{K_3, 2K_2\}$, $\{P_4\}$, $\{L_4\}$. In all other cases the path problem is solvable in linear time, and a path can be constructed in linear time.*

**Remark 2.** *One can even prove that Theorem 1 is still true if we consider bipartite graphs with symmetric forbidden transitions. Indeed, let $(G, F_G)$ be a graph with symmetric forbidden transitions, and let $(G', F_{G'})$ where*

$$
\begin{aligned}
V_{G'} &:= V_G \cup E_G, \\
E_{G'} &:= \{\{x, \{x, y\}\} \mid \{x, y\} \in E_G\}, \\
F_{G'} &:= \{(\{\{x, y\}, y\}, \{y, \{y, z\}\}) \mid (\{x, y\}, \{y, z\}) \in F_G\}.
\end{aligned}
$$

*One easily proves that $P := (x_1, \ldots, x_k)$ is a path in $(G, F_G)$ if and only if $(x_1, \{x_1, x_2\}, x_2, \ldots, x_{k-1}, \{x_{k-1}, x_k\}, x_k)$ is a path in $(G', F_{G'})$.*

An easy corollary of Theorem 1 is the following.

**Corollary 3.** *Let $(G, F_G)$ be a graph with symmetric forbidden transitions such that, for every vertex $x$ of $G$, the transition graph $A_x$ is a complete graph whenever $d_G(x) \geq 4$. Then, for every two vertices $x$ and $y$, one can construct in linear time a path between $x$ and $y$, if one exists.*

*Proof.* Let $x$ be a vertex of $G$. If $d_G(x) \geq 4$, then $A_x$ is a complete graph, and each graph in the set $\{P_3,\ P_4,\ L_4,\ 2K_2\}$ is not an induced subgraph of $A_x$. If $d_G(x) \leq 3$, then $P_4,\ L_4$ and $2K_2$ cannot be induced subgraphs of $A_x$. Therefore, any of these sets $\{K_3, 2K_2\},\ \{K_3, 2K_2\},\ \{P_4\},\ \{L_4\}$ is included in $\mathcal{A}_G^{\text{ind}}$. And by Theorem 1, one can find a path, if it exists, between every two vertices of $G$ in linear time. □

## 3 Exact Exponential Time Algorithm

Our algorithm will count the number of paths between two vertices using the principle of inclusion-exclusion as in [2]. Let us introduce some notations.

Let $(G, F_G)$ be a graph (directed or not) with forbidden transitions and let $s$ and $t$ be two vertices of $G$. For every positive integer $\ell$, we denote by $W_\ell(s,t)$ the set of $(s,t)$-walks on $\ell$ vertices, and by $P_\ell(s,t)$ the set of paths in $W_\ell(s,t)$. For $Y \subseteq V_G$, we denote by $W_{\ell,Y}(s,t)$ the walks in $W_\ell(s,t)$ that do not intersect $Y$ and similarly for $P_{\ell,Y}(s,t)$.

We propose Algorithm 1 to verify the existence of an $(s,t)$-path.

**Algorithm 1.** PathProblem$(G, F_G)$

**Data**: A graph with forbidden transitions $(G, F_G)$ and two vertices $s$ and $t$.
**Result**: Does there exist an $(s,t)$-path in $(G, F_G)$? If yes, how many vertices the shortest path contain?

```
begin
1   Let n be |V_G|
2   for ℓ ← 1 to n do
3       R := 0
4       foreach A ⊆ V_G with |A| ≥ n − ℓ do
5           R := R + C(|A|, n−ℓ) · (−1)^{|A|−(n−ℓ)} · |W_{ℓ,A}(s,t)|
        end
6       if R ≥ 1 then
7           return (YES, ℓ)
        end
    end
8   return NO
end
```

**Theorem 4.** *Algorithm 1 is correct and runs in time $O(2^n \cdot n^5 \cdot \log(n))$ for every $n$-vertex graph with forbidden transitions.*

The rest of this section is devoted to the proof of Theorem 4. So, we assume that we are given an $n$-vertex graph with forbidden transitions $(G, F_G)$ and two vertices $s$ and $t$.

**Lemma 5.** *Let $\ell$ be a fixed positive integer. Then*

$$|P_\ell(s,t)| = \sum_{\substack{Y \subseteq V_G \\ |Y|=n-\ell}} \sum_{X \subseteq V_G \setminus Y} (-1)^{|X|} \cdot |W_{\ell,Y\cup X}(s,t)|.$$

*Proof.* Since a path on $\ell$ vertices is a walk that goes through $\ell$ vertices and avoids $n-\ell$ other vertices, we have that

$$|P_\ell(s,t)| = \sum_{\substack{Y \subseteq V_G \\ |Y|=n-\ell}} |P_{\ell,Y}(s,t)|. \tag{1}$$

A walk on $\ell$ vertices that is not a path is a walk that repeats at least one vertex. Then,

$$|P_{\ell,Y}(s,t)| = |W_{\ell,Y}(s,t)| - \left| \bigcup_{x \in V_G \setminus Y} W_{\ell,Y\cup\{x\}}(s,t) \right|.$$

And, by the inclusion-exclusion principle

$$\left| \bigcup_{x \in V_G \setminus Y} W_{\ell,Y\cup\{x\}}(s,t) \right| = \sum_{\substack{X \subseteq V_G \setminus Y \\ X \neq \emptyset}} (-1)^{|X|} \cdot |W_{\ell,Y\cup X}(s,t)|.$$

Therefore,

$$|P_{\ell,Y}(s,t)| = \sum_{X \subseteq V_G \setminus Y} (-1)^{|X|} \cdot |W_{\ell,Y\cup X}(s,t)|.$$

By Eq. (1), we can conclude. □

As a corollary of Lemma 5, we get the following which proves the correctness of the algorithm.

**Corollary 6.** *Let $\ell$ be a fixed positive integer. Then,*

$$|P_\ell(s,t)| = \sum_{\substack{A \subseteq V_G \\ |A| \geq n-\ell}} \binom{|A|}{n-\ell} \cdot (-1)^{|A|-(n-\ell)} \cdot |W_{\ell,A}(s,t)|.$$

*Proof.* Choosing a subset $Y$ of $V_G$ of size $n-\ell$ and then choosing a subset of $V_G \setminus Y$, is similar to choosing a subset $A$ of $V_G$ of size at least $n-\ell$, and then choosing a subset $Y$ of $A$ of size $n-\ell$. Hence,

$$\sum_{\substack{Y \subseteq V_G \\ |Y|=n-\ell}} \sum_{X \subseteq V_G \setminus Y} (-1)^{|X|} \cdot |W_{\ell,Y\cup X}(s,t)|$$

$$= \sum_{\substack{A \subseteq V_G \\ |A| \geq n-\ell}} \binom{|A|}{n-\ell} \cdot (-1)^{|A|-(n-\ell)} \cdot |W_{\ell,A}(s,t)|.$$

By Lemma 5, we can conclude. □

It remains now to bound the time complexity of Algorithm 1.

**Lemma 7.** *Let $\ell$ be a fixed positive integer and let $Y$ be a subset of $V_G$. Then one can compute $|W_{\ell,Y}(s,t)|$ in time $O(n^4 \cdot \log(n))$.*

*Proof.* We can assume that $\ell \geq 3$ since the statement is clear for $\ell \leq 2$. If $\{s,t\} \cap Y \neq \emptyset$, then $|W_{\ell,Y}(s,t)| = 0$. So, assume that $s$ and $t$ do not belong to $Y$. Let us use the notation $W_{\ell,Y}(s,x,t)$ to denote walks in $W_{\ell,Y}(s,t)$ having $x$ as penultimate vertex. Then,

$$|W_{\ell,Y}(s,t)| = \sum_{x \in V_G \setminus Y} |W_{\ell,Y}(s,x,t)|.$$

If we know the multiset $\{|W_{\ell,Y}(s,x,t)| \mid x \in V_G \setminus Y\}$, then we can compute $|W_{\ell,Y}(s,t)|$ in time $O(n)$. We will prove that the multiset $\{|W_{\ell,Y}(s,x_0,x_1)| \mid (x_0,x_1) \in (V_G \setminus Y)^2\}$ can be computed in time $O(n^4 \cdot \log(n))$. If $\ell = 3$, then $|W_{\ell,Y}(s,x_0,x_1)| = 1 \Leftrightarrow (s,x_0,x_1) \in A_G$. Assume now that $\ell \geq 4$. Then

$$|W_{\ell,Y}(s,x_0,x_1)| = \sum_{\substack{y \in V_G \setminus Y \\ (y,x_0,x_1) \in A_G}} |W_{\ell-1,Y}(s,y,x_0)|. \tag{2}$$

We will compute the multiset $\{|W_{\ell,Y}(s,x_0,x_1)| \mid (x_0,x_1) \in (V_G \setminus Y)^2\}$ by dynamic programming using Eq. (2). We first compute $\{|W_{3,Y}(s,x_0,x_1)| \mid (x_0,x_1) \in (V_G \setminus Y)^2\}$ in time $O(n^2 \cdot \log(n))$ (there are $O(n^2)$ possible transitions and a search can be done in time $O(\log(n))$ if $T_G$ is lexicographically ordered). Now, if $\{|W_{\ell-1,Y}(s,x_0,x_1)| \mid (x_0,x_1) \in (V_G \setminus Y)^2\}$ is known, one can compute $|W_{\ell,Y}(s,x_0,x_1)|$, for some pair $(x_0,x_1) \in (V_G \setminus Y)^2$, in time $O(n \cdot \log(n))$ (by using Eq. (2)), and then one can compute the multiset $\{|W_{\ell,Y}(s,x_0,x_1)| \mid (x_0,x_1) \in (V_G \setminus Y)^2\}$ from $\{|W_{\ell-1,Y}(s,x_0,x_1)| \mid (x_0,x_1) \in (V_G \setminus Y)^2\}$ in time $O(n^3 \cdot \log(n))$ and from $\{|W_{3,Y}(s,x_0,x_1)| \mid (x_0,x_1) \in (V_G \setminus Y)^2\}$ in time $O(n^4 \cdot \log(n))$. This finishes the proof. □

*Proof (Proof of Theorem 4).* By Corollary 6, Algorithm 1 is correct. And, by Corollary 6 and Lemma 7, the time complexity is bounded by

$$\sum_{\ell=1}^{n} \sum_{i=n-\ell}^{n} \binom{n}{i} \cdot O(n^4 \cdot \log(n)) \leq O(2^n \cdot n^5 \cdot \log(n)).$$

This concludes the proof. □

## 4 Generalisation of the Minimum Cut Problem

Paths and walks are related to notions of connectivity. A classical problem in graph theory is to cut a minimum number of edges to disconnect two given vertices $s$ and $t$. In our context, there is another way to disconnect $s$ and $t$: we can just transform some allowed transitions into forbidden ones. In this section

we propose a polynomial time algorithm to find a minimum number of such transitions that must be turned into forbidden.

Let $(G, F_G)$ be a **directed** graph with forbidden transitions, and $s$, $t$ two non-adjacent vertices of $G$. We first define a new graph associated to $(G, F_G)$, and $s$, $t$ that we will use in the rest of this section.

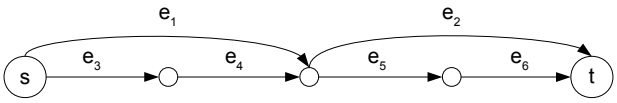
**Definition 1.** *We denote by $G^*_{s,t}$ the directed graph defined by:*

$$V_{G^*_{s,t}} := E_G \cup \{s'\} \cup \{t'\},$$
$$E_{G^*_{s,t}} := A_G \cup \{(s', (s, v)) \mid (s, v) \in E_G\} \cup \{((v, t), t') \mid (v, t) \in E_G\}$$

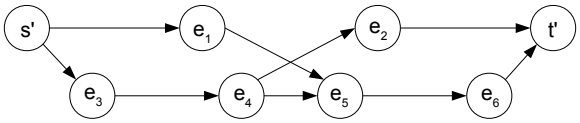
*In the following we simply denote $G^*_{s,t}$ by $G^*$.*

**Remark 8.** *We call the subgraph of $G^*$ induced by $E_G$ the* allowed line graph *of $(G, F_G)$. It admits as vertex-set $E_G$ and as edge-set $A_G$.*

For example for the following graph with forbidden transition $(G, F_G)$ with $F_G = \{(e_1, e_2)\}$:




we obtain the following graph $G^*$:




The proof of the following proposition is straightforward.

**Proposition 9.** *The function*

$$f : \begin{cases} \{(s,t)\text{-walks in } G\} \to \{(s',t')\text{-walks in } G^*\} \\ (e_1 \in E_G, ..., e_k \in E_G) \mapsto (s', e_1 \in V_{G^*}, ..., e_k \in V_{G^*}, t') \end{cases}$$

*is well defined and bijective.*

So we have the following immediate corollary.

**Corollary 10.** *$G$ is $(s,t)$-connected if and only if $G^*$ is $(s',t')$-connected.*

**Remark 11.** *We see that $f$ is a bijection between the shortest $(s,t)$-walks in $G$ and the shortest $(s',t')$-walks in $G^*$ and therefore to obtain in polynomial time a shortest $(s,t)$-walk in $G$ (if it exists) we can find a shortest path in $G^*$ (if it exists) and return its fibre under $f$.*

**Lemma 12.** *There exists an $(s', t')$-cut (*i.e. *a set of arcs disconnecting $s'$ and $t'$) in $G^*$ having no outgoing arc of $s'$ and no incoming arc of $t'$. By definition the size of a cut is its number of arcs.*

*Proof.* Since $s$ and $t$ are not adjacent in $G$ an $(s, t)$-walk in $G$ takes at least one transition. By making these transitions forbidden we obtain a new graph which is not $(s, t)$-connected. Hence by removing the corresponding arcs in $G^*$ we obtain a graph which is not $(s', t')$-connected by Corollary 10. Finally these arcs form a cut and as they correspond to transitions of $G$, there is no outgoing arc of $s'$ or incoming arc of $t'$. □

**Lemma 13.** *In $G^*$, the arcs of a minimum $(s', t')$-cut having no outgoing arc of $s'$ and no incoming arc of $t'$, correspond to a minimum set of allowed transitions in $G$ sufficient to forbid to disconnect $s$ and $t$, and reciprocally.*

*Proof.* $G^*$ deprived of these arcs is no longer $(s', t')$-connected and so $G$ deprived of the corresponding transitions is also no longer $(s, t)$-connected according to Corollary 10 and reciprocally. Moreover as the size of this cut is equal to the size of the set of corresponding transitions, if one is minimum, the other is also minimum. □

**Theorem 14.** *Let $m$ be the number of arcs of $G^*$. Assign to each arc of $G^*$ a capacity equal to one except for the outgoing arcs of $s'$ and the incoming arcs of $t'$ for which it is taken equal to $m$. The arcs of an $(s', t')$-cut of minimum capacity correspond to a minimum set of allowed transitions in $G$ sufficient to forbid to disconnect $s$ and $t$.*

*Proof.* According to Lemma 12, there exists an $(s', t')$-cut in $G^*$ having no outgoing arc of $s'$ and no incoming arc of $t'$. This cut has less than $m$ arcs (each having a capacity equal to one) and thus admits a capacity less than $m$. Thus, the minimum capacity of an $(s', t')$-cut is strictly less than $m$, and therefore an $(s', t')$-cut of minimum capacity contains no outgoing arc of $s'$ or incoming arc of $t'$ because its capacity would be greater than or equal to $m$. Now for such a cut, the capacity is equal to its size and if its size is minimum, it corresponds to a minimum set of allowed transitions in $G$ sufficient to forbid to disconnect $s$ and $t$ by Lemma 13. □

We are now able to give an algorithm that takes as input a graph with forbidden transitions $(G, F_G)$, and $s, t \in V_G$ two non-adjacent vertices, and returns a minimum set of allowed transitions sufficient to forbid to disconnect $s$ and $t$.

**Theorem 15.** *Algorithm 2 is correct and runs in time polynomial in the size of $G$.*

*Proof.* The correctness of Algorithm 2 follows from Theorem 14. For the time complexity, the steps 1, 2, and 4 run in time $O(|E_G|^2)$ and step 3 can be done using a polynomial time algorithm which computes an $(s', t')$-cut of minimum capacity in $G^*$ as the Edmonds-Karp algorithm. □

**Algorithm 2.** GeneralisedCutProblem$(G, F_G)$

**Data**: A directed graph with forbidden transitions $(G, F_G)$, and two non-adjacent vertices $s$ and $t$.
**Result**: A minimum set of allowed transitions sufficient to forbid to disconnect $s$ and $t$.
**begin**
1 Construct the graph $G^*$;
2 Assign to each arc of $G^*$ a capacity equal to one except for the outgoing arcs of $s'$ and the incoming arcs of $t'$ for which it is taken equal to $|E_G|$;
3 Compute an $(s', t')$-cut of minimum capacity in $G^*$ with these capacities;
4 **return** *Transitions of $G$ corresponding to the arcs of the cut.*
**end**

Now if $(G, F_G)$ is an **undirected** graph with forbidden transitions and $s$, $t$ two non-adjacent vertices of $V_G$, we introduce the directed graph with forbidden transitions $G_d$ defined formally as follows:

$$\begin{aligned} V_{G_d} &:= V_G, \\ E_{G_d} &:= \{(v,w),(w,v) \mid \{v,w\} \in E_G\}, \\ F_{G_d} &:= \{((v,w),(w,x)) \mid (\{v,w\},\{w,x\}) \in F_G\}. \end{aligned}$$

As $G$ and $G_d$ have the same walks (sequences of vertices) just apply the algorithm 2 to $G_d$ to obtain a minimum set of allowed transitions sufficient to forbid to disconnect $s$ and $t$ in $G$.

**Remark 16.** *To obtain (if it exists) in polynomial time a shortest $(s,t)$-walk in an undirected graph with forbidden transitions $G$ we can find (if it exists) a shortest path in $G_d^*$ and return its fibre under $f$.*

## 5 Conclusion and Perspectives

Including forbidden transitions into graphs strongly extend the capacities to model real situations but also increases the complexity of *a priori* simple tasks such as finding elementary (and shortest) paths. When the total number $k$ of forbidden transitions in $G$ is low one can apply a simple branch and bound algorithm to find (if there is one) such a path. Up to $k = O(\log n)$, this elementary algorithm remains polynomial. However, as $k$ can be much larger than $n$, we proposed in Section 3 another method with time complexity $O(2^n \cdot poly(n))$.

In Section 4 we transposed a classical cutting problem: instead of cutting a minimum number of edges to disconnect two given vertices $s$ and $t$, we proposed a polynomial time algorithm to turn a minimum number of allowed transitions into forbidden ones in such a way that there is no more walk between $s$ and $t$. This is another measure of connectivity between $s$ and $t$ and can be used for example in preliminary studies to prevent to disconnect two parts of a city when

there are temporary traffic restrictions due for example to a punctual event or due to works in streets.

In conclusion, we can see graphs with forbidden transitions as graphs with "reduced connection capabilities". Due to the potential applications, we plan to further investigate this subject. We hope to decrease the $O(2^n \cdot poly(n))$ complexity of the exact algorithm. Our result on the equivalent cutting problem in Section 4 motivates us to try to generalise the flow problem in graphs with forbidden transitions. In a more general way we intend to generalise classical algorithmic problems into graphs with forbidden transitions.

## References

1. Ahmed, M., Lubiw, A.: Shortest paths avoiding forbidden subpaths. In: Albers, S., Marion, J.-Y. (eds.) STACS. LIPIcs, vol. 3, pp. 63–74. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2009)
2. Bax, E.T.: Inclusion and exclusion algorithm for the hamiltonian path problem. Inf. Process. Lett. 47(4), 203–207 (1993)
3. Błażewicz, J., Kasprzak, M.: Computational complexity of isothermic dna sequencing by hybridization. Discrete Applied Mathematics 154(5), 718–729 (2006)
4. Błażewicz, J., Kasprzak, M., Leroy-Beaulieu, B., de Werra, D.: Finding hamiltonian circuits in quasi-adjoint graphs. Discrete Applied Mathematics 156(13), 2573–2580 (2008)
5. Bondy, J.A., Murty, U.S.R.: Graph Theory. Springer London Ltd. (2010)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press (2009)
7. Dvořák, Z.: Two-factors in orientated graphs with forbidden transitions. Discrete Mathematics 309(1), 104–112 (2009)
8. Szeider, S.: Finding paths in graphs avoiding forbidden transitions. Discrete Applied Mathematics 126(2-3), 261–273 (2003)
9. Villeneuve, D., Desaulniers, G.: The shortest path problem with forbidden paths. European Journal of Operational Research 165(1), 97–107 (2005)

# Polynomial Time Algorithms for Computing a Minimum Hull Set in Distance-Hereditary and Chordal Graphs*

Mamadou Moustapha Kanté and Lhouari Nourine

Clermont-Université, Université Blaise Pascal, LIMOS, CNRS, France
{mamadou.kante,nourine}@isima.fr

**Abstract.** We give linear and polynomial time algorithms for computing a minimum hull-set in distance-hereditary and chordal graphs respectively. Prior to our result a polynomial time algorithm was only known for sub-classes of considered graph classes. Our techniques allow us to give at the same time a linear time algorithm for computing a minimum geodetic set in distance-hereditary graphs.

## 1 Introduction

In this paper we consider convexity related to shortest paths in graphs. For an undirected graph $G$ and a subset $X$ of the vertex-set of $G$, let $I[X]$ be the set of vertices that lie in a shortest path between two vertices of $X$. The *closure* of $X$ is the smallest $X' \supseteq X$ such that $I[X'] = X'$. The *hull number* [16] of a graph $G$ is defined as the minimum $k$ such that there exists a set $X$ of size $k$ whose closure is the set of vertices of $G$. The notion of hull number has been introduced in [16] and has attracted many interest in the years [1,3,12,13]. Computing the minimum hull number of graphs is NP-complete [1,12] and polynomial time algorithms have been proposed for some graph classes: proper interval graphs, cographs, split graphs [12], cobipartite graphs [1].

In this paper we give polynomial time algorithms for computing a minimum hull set in distance-hereditary and chordal graphs. The computational complexity of the hull number of chordal graphs has been left open since [12] and to our knowledge no polynomial time algorithm for the computation of the hull number of distance-hereditary graphs is known. Surprisingly, the related notion *geodetic number* has been proven to be NP-complete in chordal graphs [14] and a polynomial time algorithm for interval graphs is open (polynomial time algorithms for split graphs and proper interval graphs are given in [14] and [15]).

Our linear time (in the size of the graph) algorithm for distance-hereditary graphs can be summarised as follows. We will first give a *monadic second-order* formula $HullSet(X)$ that holds in a connected distance-hereditary graph $G$ if and only if $X$ is a hull set of $G$. We will then use Courcelle et al.'s theorem

* M.M. Kanté is supported by the DORSO project, and L. Nourine by the DAG project, both of "Agence Nationale Pour la Recherche".

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 268–279, 2013.

stating that monadic second-order properties can be solved in linear time in distance-hereditary graphs (see [7]). In order to write the formula $HullSet(X)$ we will express in monadic second-order logic the property "$z$ is in a shortest path between $x$ and $y$" and we will use for that the well-known notion of *split decomposition* [8] and a characterisation of the property "being in a shortest path" by means of split decomposition.

The algorithm for chordal graphs is based upon the notion of *functional dependencies* which are constraints used in relational database design and specially in normalisation process [5].

**Summary.** Section 2 is devoted to preliminaries and basic facts. A linear time algorithm for distance-hereditary graphs, based on logical tools, is given in Section 3. Section 4 is devoted to the polynomial time algorithm for chordal graphs. We finish by some concluding remarks and open questions in Section 5.

## 2 Preliminaries

**Graphs.** If $A$ and $B$ are two sets, $A \backslash B$ denotes the set $\{x \in A \mid x \notin B\}$. The power set of a set $V$ is denoted by $2^V$. The size of a set $A$ is denoted by $|A|$.

We refer to [10] for our graph terminology. A *graph* $G$ is a pair $(V_G, E_G)$ where $V_G$ is the set of vertices and $E_G \subseteq (V_G \times V_G) \setminus (\{(x,x) \mid x \in V_G\})$ is the set of edges. A graph $G$ is said to be *undirected* if $(x,y) \in E_G$ implies $(y,x) \in E_G$; an edge $(x,y)$ is hence written $xy$ (equivalently $yx$). For a graph $G$, we denote by $G[X]$, called the subgraph of $G$ induced by $X \subseteq V_G$, the graph $(X, E_G \cap (X \times X))$. The *size* of a graph $G$, denoted by $\|G\|$, is defined as $|V_G| + |E_G|$.

A *path of length* $k$ in a graph $G$ from the vertex $x$ to the vertex $y$ is a sequence $(x = x_0, x_1, \ldots, x_k = y)$ such that the set $\{(x_i, x_{i+1}) \mid 0 \leq i \leq k-1\}$ is a subset of $E_G$; this path is said *chordless* if $(x_i, x_j) \notin E_G$ whenever $j > i+1$ or $j < i-1$. It is worth noticing that whenever $G$ is undirected, if $P := (x_0, \ldots, x_k)$ is a path from $x$ to $y$, $(x_k, \ldots, x_0)$ is also a path from $y$ to $x$ and we will say in this case that $P$ is between $x$ and $y$. A graph $G$ is said *strongly connected* if for every pair $(x,y)$ of vertices there exists a path from $x$ to $y$ and also a path from $y$ to $x$. A *strongly connected component* of a graph $G$ is a maximal strongly connected induced subgraph of $G$. Strongly connected undirected graphs are simply said to be *connected* and their strongly connected components are called *connected components*.

The *distance* between two vertices $x$ and $y$ in an undirected graph $G$, denoted by $d_G(x,y)$, is the minimum $k$ such that there exists a path of length $k$ between $x$ and $y$; if no such path exists then $d_G(x,y) = \infty$ (this does happen if $G$ is not connected). Any path between two vertices $x$ and $y$ of length $d_G(x,y)$ is called a *shortest path* and is by definition a chordless path.

An undirected graph $G$ is said *complete* if $E_G = (V_G \times V_G) \setminus (\{(x,x) \mid x \in V_G\})$ (it is denoted $K_n$ if it has $n$ vertices), and it is called a *cycle of length* $n$ if its edge-set is the set $\{x_i x_{i+1} \mid 1 \leq i \leq n-1\} \cup \{x_1 x_n\}$ with $(x_1, \ldots, x_n)$ an ordering of its vertex-set. An undirected graph $G$ is called *distance-hereditary* if for every

two vertices $x$ and $y$ of $G$ all the chordless paths between $x$ and $y$ have the same length, and it is called *chordal* if it has no induced cycle of length greater than or equal to 4. A vertex $x$ of an undirected graph $G$ is called *simplicial* if $G[\{y \mid xy \in E_G\}]$ is a complete graph. A *tree* is an acyclic connected undirected graph and a *star* is a tree with a distinguished vertex adjacent to the other vertices (it is denoted $S_n$ if it has $n$ vertices).

**Betweenness Relations.** Let $V$ be a finite set. A *betweenness relation on* $V$ is a ternary relation $B \subseteq V^3$ such that for every $x, y, z \in V$, we have $B(x, z, y)$ holds if and only if $B(y, z, x)$ holds; $z$ is said to be *between* $x$ and $y$. Several betweenness relations, particularly in graphs, are studied in the literature (see [4,20]).

Let $B$ be a betweenness relation on a finite set $V$. For every subset $X$ of $V$, we let $B^o(X)$ be $\bigcup_{x,y \in X}\{z \in V \mid B(x, z, y)\}$. A subset $X$ of $V$ is said $B$*-convex* if $B^o(X) = X$ and its $B$*-convex hull*, denoted by $B^+(X)$, is the smallest $B$-convex set that contains $X$. A subset $X$ of $V$ is a $B$*-hull set* (resp. $B$*-geodetic set*) if $B^+(X) = V$ (resp. $B^o(X) = V$).

In this paper we deal with the following betweenness relation. For every undirected graph $G$, we define the betweenness relation $\mathcal{SP}_G$ on $V_G$ where $\mathcal{SP}_G(x, z, y)$ holds if and only if $z$ is in a shortest path between $x$ and $y$. We are interested in computing the *hull number* of an undirected graph $G$ defined as the size of a minimum $\mathcal{SP}_G$-hull set of $V_G$ [16]. The computation of the hull number of a graph is NP-complete [12] and polynomial time algorithms exist for some graph classes (see for instance [1,12,13]). We give polynomial time algorithms for distance-hereditary and chordal graphs. Our techniques will allow us to derive a linear time algorithm for computing the *geodetic number* of a distance-hereditary graph $G$, defined as the $\mathcal{SP}_G$-geodetic set of $V_G$ [14].

**Fact 1.** *A $\mathcal{SP}_G$-hull set of a non connected undirected graph $G$ is the union of $\mathcal{SP}_G$-hull sets of its connected components. Similarly for $\mathcal{SP}_G$-geodetic sets.*

*Proof.* This follows from the fact that for every triple $x, y, z$ of $V_G$, $z$ is in a shortest path between $x$ and $y$ if and only if $d_G(x, y) = d_G(x, z) + d_G(z, y)$.

**Fact 2.** *Any $\mathcal{SP}_G$-hull set (or $\mathcal{SP}_G$-geodetic set) of an undirected graph $G$ must contain the set of simplicial vertices of $G$.*

**Proposition 3.** *Let $G$ be a connected undirected graph and let $x$ be a vertex of $G$. Then, we can compute all the sets $\mathcal{SP}^o_G(\{x, y\})$, for all vertices $y \in V_G \setminus \{x\}$, in time $O(\|G\| + \sum_{y \in V_G \setminus \{x\}} \left|\mathcal{SP}^o_G(\{x, y\})\right|)$.*

**Monadic Second-Order Logic.** We refer to [7] for more information. A *relational signature* is a finite set $\mathcal{R} := \{R, S, T, \ldots\}$ of relation symbols, each of which given with an arity $ar(R) \geq 1$. A *relational $\mathcal{R}$-structure* $\mathfrak{A}$ is a tuple $(A, (R_{\mathfrak{A}})_{R \in \mathcal{R}})$ with $R_{\mathfrak{A}} \subseteq A^{ar(R)}$ for every $R \in \mathcal{R}$ and $A$ is called its domain. Examples of relational structures are graphs that can be seen as relational

$\{edg\}$-structures with $ar(edg) = 2$, *i.e.*, a graph $G$ is seen as the relational $\{edg\}$-structure $(V_G, edg_G)$ with $V_G$ its set of vertices and $edg_G(x, y)$ holds if and only if $(x, y) \in E_G$.

We will use lower case variables $x, y, z, \ldots$ (resp. upper case variables $X, Y, Z, \ldots$) to denote elements of domains (resp. subsets of domains) of relational structures. Let $\mathcal{R}$ be a relational signature. The *atomic formulas over* $\mathcal{R}$ are $x = y$, $x \in X$ and $R(x_1, \ldots, x_{ar(R)})$ for $R \in \mathcal{R}$. The set $MS_{\mathcal{R}}$ of *monadic second-order formulas over* $\mathcal{R}$ is the set of formulas formed from atomic formulas over $\mathcal{R}$ with Boolean connectives $\wedge$, $\vee$, $\neg$, $\Longrightarrow$, $\Longleftrightarrow$, *element quantifications* $\exists x$ and $\forall x$, and *set quantifications* $\exists X$ and $\forall X$. An occurrence of a variable which is not under the scope of a quantifier is called a *free variable*. We will write $\varphi(x_1, \ldots, x_m, Y_1, \ldots, Y_q)$ to express that the formula $\varphi$ has $x_1, \ldots, x_m, Y_1, \ldots, Y_q$ as free variables and $\mathfrak{A} \models \varphi(a_1, \ldots, a_m, Z_1, \ldots, Z_q)$ to say that $\varphi(a_1, \ldots, a_m, Z_1, \ldots, Z_q)$ holds in $\mathfrak{A}$ when substituting $(a_1, \ldots, a_m) \in A^m$ to element variables $(x_1, \ldots, x_m)$ and $(Z_1, \ldots, Z_q) \in (2^A)^q$ to set variables $(Y_1, \ldots, Y_q)$ in $\varphi(x_1, \ldots, x_m, Y_1, \ldots, Y_q)$. The following is an example of a formula expressing that two vertices $x$ and $y$ are connected by a path

$$\forall X(x \in X \ \wedge \ \forall z, t(z \in X \ \wedge \ edg(z, t) \implies t \in X) \Longrightarrow y \in X).$$

If $\varphi(x_1, \ldots, x_m, Y_1, \ldots, Y_q)$ is a formula in $MS_{\mathcal{R}}$, we let $opt_\varphi$, with $opt \in \{\min, \max\}$, be the problem that consists in, given a relational $\mathcal{R}$-structure $\mathfrak{A}$, to finding a tuple $(Z_1, \ldots, Z_q)$ of $(2^A)^q$ such that

$$\sum_{1 \leq i \leq q} |Z_i| = opt \left\{ \sum_{1 \leq i \leq q} |W_j| \mid \mathfrak{A} \models \varphi(a_1, \ldots, a_m, W_1, \ldots, W_q) \right\}.$$

Many optimisation graph problems, *e.g.*, minimum dominating set, maximum clique, ..., correspond to $opt_\varphi$ for some $MS_{\{edg\}}$ formula $\varphi$.

Let $A$ be a finite set. An *$A$-coloured graph* is a graph with its edges and vertices labelled with elements in $A$. Let $\mathcal{R}_A$ be the relational signature $\{(edg_a)_{a \in A}, (nlab_a)_{a \in A}\}$ with $ar(edg_a) = 2$ and $ar(nlab_a) = 1$ for every $a \in A$. Every $A$-coloured graph $G$ can be represented by the relational $\mathcal{R}_A$-structure $(V_G, (edg_{aG})_{a \in A}, (nlab_{aG})_{a \in A})$ where $edg_{aG}(x, y)$ holds if and only if $(x, y) \in E_G$ is labelled with $a \in A$ and $nlab_{aG}(x)$ holds if and only if $x \in V_G$ is labelled with $a \in A$.

*Clique-width* is a graph complexity measure introduced by Courcelle et al. and that is important in complexity theory (we refer to [7] for more information on clique-width). We recall the following important theorem that is the base of our algorithm for distance-hereditary graphs.

**Theorem 4.** *[7] Let $A$ be a fixed finite set and let $k$ be a fixed constant. For every $MS_{\mathcal{R}_A}$ formula $\varphi(x_1, \ldots, x_m, Y_1, \ldots, Y_q)$, $opt_\varphi$, for $opt \in \{\min, \max\}$, can be solved in time $O(f(k, |A|, \varphi) \cdot |V_G|)$, for some function $f$, in any $A$-coloured graph of clique-width at most $k$, provided the clique-width expression is given.*

## 3 Distance-Hereditary Graphs

We will first use the well-known notion of *split-decomposition* defined by Cunningham and Edmonds [8] to associate with every distance-hereditary graph $G$ an $A$-coloured graph $\mathcal{S}_G$ for some finite set $A$. In a second step, we will prove that there exists a formula $Bet_{\mathcal{SP}}(x,z,y)$ in $MS_{\mathcal{R}_A}$ that holds in $\mathcal{S}_G$ if and only if $z$ is in a shortest path in $G$ between $x$ and $y$. These two constructions combined with Theorem 4 and the next proposition will give rise to the linear time algorithm.

**Proposition 5.** *Let $B$ be a betweenness relation on a finite set $V$ and assume there exists a relational $\mathcal{R}$-structure $\mathfrak{A}$ with $V \subseteq A$, for some relational signature $\mathcal{R}$ that contains a relation $nlab_{\mathbf{V}}$ representing $V$. Assume also that there exists an $MS_{\mathcal{R}}$ formula $Bet(x,z,y)$ that holds in $\mathfrak{A}$ if and only if $B(x,z,y)$ holds. Then, there exist $MS_{\mathcal{R}}$ formulas $HullSet(X)$ and $GeodeticSet(X)$ expressing that $X$ is a $B$-hull set and a $B$-geodetic set respectively.*

### 3.1 Split Decomposition

We will follow [6] (see also [18]) for our definitions. Two bipartitions $\{X_1, X_2\}$ and $\{Y_1, Y_2\}$ of a set $V$ *overlap* if $X_i \cap Y_j \neq \emptyset$ for all $i,j \in \{1,2\}$. A *split* in a connected undirected graph $G$ is a bipartition $\{X,Y\}$ of the vertex set $V_G$ such that $|X|, |Y| \geq 2$, and there exist $X_1 \subseteq X$ and $Y_1 \subseteq Y$ such that $E_G = E_{G[X]} \cup E_{G[Y]} \cup X_1 \times Y_1$ (see Figure 1). A split $\{X,Y\}$ is *strong* if there is no other split $\{X',Y'\}$ such that $\{X,Y\}$ and $\{X',Y'\}$ overlap. Notice that not all graphs have a split. Those that do not have a split are called *prime*.

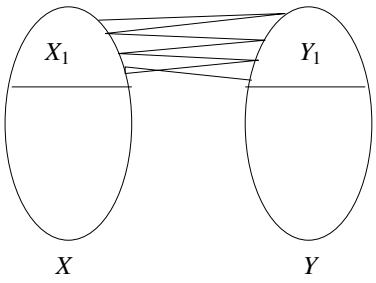



**Fig. 1.** A schematic view of a split

If $\{X,Y\}$ is a split of an undirected graph $G$, then we let $\mathcal{G}[X]$ and $\mathcal{G}[Y]$ be respectively $(X \cup \{h_X\}, E_{G[X]} \cup \{xh_X \mid x \in X \text{ and } \exists y \in Y,\ xy \in E_G\})$ and $(Y \cup \{h_Y\}, E_{G[Y]} \cup \{yh_Y \mid y \in Y \text{ and } \exists x \in X,\ xy \in E_G\})$ where $h_X$ and $h_Y$ are new vertices. The vertices $h_X$ and $h_Y$ are called *neighbour markers* of $\mathcal{G}[X]$

and $\mathcal{G}[Y]$. Notice that given $\mathcal{G}[X]$ and $\mathcal{G}[Y]$ with neighbour markers $h_X$ and $h_Y$ distinguished, we can reconstruct $G$ as follows

$$V_G = (V_{\mathcal{G}[X]} \cup V_{\mathcal{G}[Y]}) \setminus \{h_X, h_Y\},$$
$$E_G = (E_{\mathcal{G}[X]} \cup E_{\mathcal{G}[Y]}) \setminus (\{xh_X \in E_{\mathcal{G}[X]}\} \cup \{xh_Y \in E_{\mathcal{G}[Y]}) \cup \{xy \mid x \in V_{\mathcal{G}[X]},\ y \in V_{\mathcal{G}[Y]} \text{ and } xh_X \in E_{\mathcal{G}[X]},\ yh_Y \in E_{\mathcal{G}[Y]}\}.$$

**Fact 6.** *Let $\{X, Y\}$ be a split in a connected undirected graph $G$ and let $P := (x_0, \ldots, x_k)$ be a path in $G$.*

1. *If $x_0 \in X$ and $x_k \in Y$, then $P$ is a shortest path if and only if there exists $x_i \in P$ such that $(x_0, \ldots, x_i, h_X)$ and $(h_Y, x_{i+1}, \ldots, x_k)$ are shortest paths in $\mathcal{G}[X]$ and $\mathcal{G}[Y]$ respectively.*
2. *If $x_0, x_k \in X$, then $P$ is a shortest path if and only if either $P$ is a shortest path in $\mathcal{G}[X]$ or $(x_0, \ldots, x_{i-1}, h_X, x_{i+1}, \ldots, x_k)$ is a shortest path in $\mathcal{G}[X]$ with $x_i \in Y$. Similarly for $x_0, x_k \in Y$.*

A *decomposition* of a connected undirected graph $G$ is defined inductively as follows: $\{G\}$ is the only decomposition of size 1. If $\{G_1, \ldots, G_n\}$ is a decomposition of size $n$ of $G$ and $G_i$ has a split $\{X, Y\}$, then $\{G_1, \ldots, G_{i-1}, \mathcal{G}_i[X], \mathcal{G}_i[Y], G_{i+1}, \ldots, G_n\}$ is a decomposition of size $n+1$. Notice that the decomposition process must terminate because the new graphs $\mathcal{G}_i[X]$ and $\mathcal{G}_i[Y]$ are smaller than $G_i$. The graphs $G_i$ of a decomposition are called *blocks*. If two blocks have neighbour markers, we call them *neighbour blocks*.

A decomposition is *canonical* if and only if: (i) each block is either prime (called *prime block*), or is isomorphic to $K_n$ (called *clique block*) or to a $S_n$ (called *star block*) for $n \geq 3$, (ii) no two clique blocks are neighbour, and (iii) if two star blocks are neighbour, then either their markers are both centres or both not centres.

**Theorem 7 ([8,9]).** *Every connected undirected graph has a unique canonical decomposition, up to isomorphism. It can be obtained by iterated splitting relative to strong splits. This canonical decomposition can be computed in time $O(\|G\|)$ for every undirected graph $G$.*

The canonical decomposition of a connected undirected graph $G$ constructed in Theorem 7 is called *split-decomposition* and we will denote it by $\mathcal{D}_G$.

### 3.2 Definition of $Bet_{\mathcal{SP}}(x, z, y)$ and the Linear Time Algorithm

We let $A$ be the set $\{s, \epsilon, \mathbf{V}, \mathbf{M}\}$. For every connected undirected graph $G$ we associate the $A$-coloured graph $\mathcal{S}_G$ where $V_{\mathcal{S}_G} = \bigcup_{G_i \in \mathcal{D}_G} V_{G_i}$, $E_{\mathcal{S}_G} = (\bigcup_{G_i \in \mathcal{D}_G} E_{G_i}) \cup \{xy \mid x, y \text{ are neighbour markers}\}$, a vertex $x$ is labelled $\mathbf{V}$ if and only if $x \in V_G$, otherwise it is labelled $\mathbf{M}$, and an edge $xy$ is labelled $s$ if and only if $xy \in E_{G_i}$ for some $G_i \in \mathcal{D}_G$, otherwise it is labelled $\epsilon$. Figure 2 gives an example of the graph $\mathcal{S}_G$.

A path $(x_0, x_1, \ldots, x_k)$ in $\mathcal{S}_G$ is said *alternating* if, for every $1 \leq i \leq k-1$, the labels of the edges $x_{i-1}x_i$ and $x_ix_{i+1}$ are different.

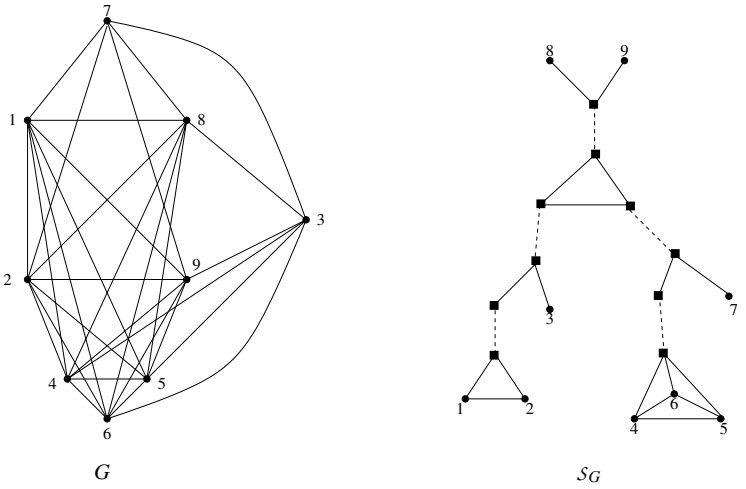



**Fig. 2.** Solid lines are edges labelled by $s$ and the dashed ones are labelled by $\epsilon$. The vertices labelled by **V** are those with a number and those labelled by **M** are the rest.

**Lemma 8.** *[6] Let $G$ be a connected undirected graph. Then, $xy \in E_G$ if and only if there exists an alternating path between $x$ and $y$ in $\mathcal{S}_G$. This alternating path is moreover unique.*

The following gives a characterisation of the property "$z$ is in a shortest path in $G$ between $x$ and $y$" with respect to shortest paths in $\mathcal{S}_G$.

**Proposition 9.** *Let $G$ be a connected undirected graph and let $x, y, z$ be vertices of $G$. Then, $\mathcal{SP}_G(x, z, y)$ holds if and only if*

*(i) $\mathcal{SP}_{\mathcal{S}_G}(x, z, y)$ holds, or*
*(ii) there exists a marker $h$ such that $\mathcal{SP}_{\mathcal{S}_G}(x, h, y)$ holds and there exists an alternating path between $h$ and $z$ in $\mathcal{S}_G$ starting with an edge labelled by $\epsilon$.*

We now return to distance-hereditary graphs. We will use the following theorem that characterises distance-hereditary graphs (among the several ones).

**Theorem 10 ([2]).** *A connected undirected graph is a distance-hereditary graph if and only if each block of its split decomposition is either a clique or a star block.*

As a corollary we get the following (the proof is an easy induction on the size of the split decomposition).

**Corollary 11.** *Let $G$ be a connected distance-hereditary graph. Then, a sequence $P := (x_0, \ldots, x_k)$ is a shortest path in $\mathcal{S}_G$ if and only if $P$ is a chordless path in $\mathcal{S}_G$. Moreover, a shortest path between two vertices in $\mathcal{S}_G$ is unique.*

We can therefore prove the following.

**Proposition 12.** *There exists an $MS_{\mathcal{R}_A}$ formula $Bet_{\mathcal{SP}}(x, z, y)$ that expresses that $z$ is in a shortest path between $x$ and $y$ in connected distance-hereditary graphs.*

By Corollary 11 the graph obtained from $\mathcal{S}_G$ by forgetting the labels of the vertices and of the edges is a distance-hereditary graph, and then has clique-width at most 3 (see [19]). We can in fact prove that the $A$-coloured graph $\mathcal{S}_G$ has clique-width at most 3 as stated below (its proof is an easy induction).

**Proposition 13.** *For every connected distance-hereditary graph $G$, the $A$-coloured graph $\mathcal{S}_G$ has clique-width at most 3. Moreover, a clique-width expression of $\mathcal{S}_G$ can be computed in time $O(\|\mathcal{S}_G\|)$.*

**Theorem 14.** *For every distance-hereditary graph $G$ one can compute in time $O(\|G\|)$ a minimum $\mathcal{SP}_G$-hull set and a minimum $\mathcal{SP}_G$-geodetic set of $G$.*

*Proof.* From Fact 1 it is enough to prove the theorem for connected distance-hereditary graphs. So assume that $G$ is connected. By Theorem 10 one can compute in time $O(\|G\|)$ the split decomposition $\mathcal{D}_G$ of $G$. By [18, Lemma 2.2] we have $\|\mathcal{S}_G\| = O(\|G\|)$, and therefore one can compute $\mathcal{S}_G$ in time $O(\|G\|)$. By Theorem 4 and Propositions 5, 12 and 13, one can compute a minimum $\mathcal{SP}_G$-hull set and a minimum $\mathcal{SP}_G$-geodetic set of $G$ in time $O(\|\mathcal{S}_G\|) = O(\|G\|)$. □

## 4 Chordal Graphs

The algorithm for chordal graphs is based on the notion of *functional dependencies*, borrowed from database community. Before introducing them, let us recall some properties of chordal graphs.

**Lemma 15.** *[11] Every chordal graph has at least two simplicial vertices.*

A *perfect elimination ordering* of a graph $G$ is an ordering $\sigma := (x_1, \ldots, x_n)$ of $V_G$ such that for every $1 \leq i \leq n$, the vertex $x_i$ is simplicial in $G[\{x_i, \ldots, x_n\}]$. The following is a well-known result.

**Theorem 16.** *[17,21] A graph $G$ is chordal if and only if it has a perfect elimination ordering. Moreover, a perfect elimination ordering can be computed in time $O(\|G\|)$.*

We now define *functional dependencies*. Let $V$ be a finite set. A *functional dependency* on $V$ is a pair $(X, y)$, often written $X \to y$, where $X \subseteq V$ is called the *premise*, and $y \in V$ is called the *conclusion*. If $X \to z$ is a functional dependency with $X = \{x, y\}$ (or $X = \{x\}$), we will write $xy \to z$ (or $x \to z$) instead of $\{x, y\} \to z$ (or $\{x\} \to z$). A *model* for $X \to y$ is a subset $F$ of $V$ such that $y \in F$ whenever $X \subseteq F$.

A set $\Sigma$ of functional dependencies on $V$ is called an *implicational system on* $V$. If an implicational system is composed only of functional dependencies with premises of size 1 we call it a *unit-premise implicational system*. A set $X \subseteq V$ is

said $\Sigma$-*closed* if it is a model for each functional dependency in $\Sigma$. The $\Sigma$-*closure* of a set $X$, noted $\Sigma(X)$, is the smallest $\Sigma$-closed set that contains $X$. A *superkey* for $\Sigma$ is a subset $K$ of $V$ such that $\Sigma(K) = V$ and a *key* is an inclusionwise minimal superkey.

If $\Sigma$ is an implicational system on a finite set $V$, we let $\mathscr{G}(\Sigma)$, called the *dependence graph* of $\Sigma$, be the directed graph $(V', E')$ with

$$V' := V \cup \{P_X \mid X \text{ is a premise in } \Sigma\},$$
$$E' := \bigcup_{X \to y \in \Sigma} \Big( \big( \{(x, P_X) \mid x \in X\} \big) \cup \{(P_X, y)\} \Big).$$

One observes that $\mathscr{G}(\Sigma)$ has size $O\left(|V| + \sum_{X \to y \in \Sigma} \big(|X| + 1\big)\right)$ and can be computed in time $O\left(|V| + \sum_{X \to y \in \Sigma} \big(|X| + 1\big)\right)$.

**Proposition 17.** *Let $\Sigma$ be a unit-premise implicational system on a finite set $V$. Then a minimum key of $\Sigma$ can be computed in time $O(|V| + |\Sigma|)$.*

We now borrow some ideas from [22]. Let $\Sigma$ be an implicational system on a finite set $V$. A strongly connected component $S$ of $\mathscr{G}(\Sigma)$ is called a *source component* if for all $x, y \in V_{\mathscr{G}(\Sigma)}$, if $(x, y) \in E_{\mathscr{G}(\Sigma)}$ and $y \in V_S$, then $x \in V_S$. That means that all edges between a vertex in $S$ and a vertex in $V_{\mathscr{G}(\Sigma)} \setminus S$ is always from $S$.

Let $R$ be a subset of $V$. We let $\Sigma^R$, called the *restriction of $\Sigma$ to $R$*, be the implicational system on $R$ defined as $\{X \to y \in \Sigma \mid X \cup \{y\} \subseteq R\}$. A *contraction* of $\Sigma$ to $R$ is the implicational system $\overline{\Sigma^R}$ on $V \setminus \Sigma(R)$ defined as

$$\big\{X \to y \mid X \cup S \to y \in \Sigma, \text{ and } S \subseteq \Sigma(R) \text{ and } X \cup \{y\} \subseteq V \setminus \Sigma(R)\big\} \bigcup \big\{X \to y \in \Sigma \mid X \cup \{y\} \subseteq V \setminus \Sigma(R)\big\}.$$

**Proposition 18.** *[22] Let $\Sigma$ be an implicational system on a finite set $V$. Let $S$ be a source component of $\mathscr{G}(\Sigma)$. Then each minimum key of $\Sigma$ is a union of a minimum key of $\Sigma^{V_S}$ and of a minimum key of $\overline{\Sigma^{V_S}}$. Moreover, all such unions are minimum keys of $\Sigma$.*

If $\Sigma$ is an implicational system on $V$, then $x \in V$ is called an *extreme point* if $x$ is not the conclusion of any functional dependency in $\Sigma$. It is straightforward to verify that any extreme point is a source component in $\mathscr{G}(\Sigma)$ and then is in any minimum key of $\Sigma$. (Notice that a similar notion of extreme points has been already used in the literature, see for instance [4].)

*Example 1.* Let $V := \{1, 2, 3, 4, 5, 6\}$ and let $\Sigma := \{\{1, 2\} \to 3, \{2, 4\} \to 1, \{1, 3, 5\} \to 6, \{1, 4\} \to 3, 1 \to \{2, 4\}\}$. 5 is an extreme point, and indeed it must be in any key of $\Sigma$, but 6 cannot be in any key. Examples of keys are $\{1, 5\}, \{2, 4, 5\}$, and examples of $\Sigma$-closed sets are $\{2\}, \{1, 2, 3, 4\}, \{3, 6\}$.

We can now explain the algorithm for connected chordal graphs. We will associate with every graph $G$ the implicational system on $V_G$

$$\Sigma_G := \bigcup_{x,y \in V} \{xy \to z \mid z \in \mathcal{SP}^o_G(\{x,y\}) \setminus \{x,y\}\}.$$

One notices that a vertex $x$ of a graph $G$ is simplicial if and only if $x$ is an extreme point in $\Sigma_G$.

**Fact 19.** *Let $G$ be an undirected graph. Then, $\mathcal{SP}^+_G(X) = \Sigma_G(X)$ for every $X \subseteq V_G$. Then, $K$ is a minimum $\mathcal{SP}_G$-hull set of $G$ if and only if $K$ is a minimum key of $\Sigma_G$.*

According to Proposition 18, our strategy is recursively.

1. Take a source component $S$ of $\Sigma_G$ and decompose $\Sigma_G$ into $\Sigma_G^{V_S}$ and $\overline{\Sigma_G^{V_S}}$.
2. Compute a minimum key $K_1$ of $\Sigma_G^{V_S}$.
3. Compute a minimum key $K_2$ of $\overline{\Sigma_G^{V_S}}$.
4. Return $K_1 \cup K2$.

The first difficulty is here when the source component is not a single element. The second difficulty is to iterate this process in $\Sigma := \overline{\Sigma_G^{V_S}}$. We overcome these difficulties by first constructing an elimination ordering $\sigma := (x_1, \ldots, x_n)$ of $G$ and the algorithm treats the vertices of $G$, in this order, and decides at each step whether the current vertex can be in a minimum key (that is updated at each step). At the beginning $\Sigma$ is set up to $\Sigma_G$. At each step $i$, if $x_i$ is in the closure of the already computed key, then we go to the next step (the current vertex cannot be in a minimum key). Otherwise, if $x_i$ is an extreme point in $\Sigma$, then it is a source component and then we can include it in a minimum key of $\Sigma$ and use Proposition 18 to update $\Sigma$ for the next iteration. If $x_i$ is not an extreme point, then since it is a simplicial vertex in $G[\{x_i, \ldots, x_n\}]$, it cannot be a conclusion in any functional dependencies with premises of size 2 in $\Sigma$. So, that means it appears as a conclusion in functional dependencies with premises of size 1 in $\Sigma$ (those latter are created during the process by Proposition 18). Since we cannot decide whether to include $x_i$ in the computed minimum key, we update $\Sigma$, according to Lemma 20 below, in order to remove $x_i$ in the premises of functional dependencies in $\Sigma$. At the end of the $n$ iterations, $\Sigma$ is reduced to a unit-premise implicational system and by Proposition 17 we can compute minimum keys of such implicational systems.

**Lemma 20.** *Let $\Sigma := \Sigma_1 \cup \Sigma_2$ be an implicational system on a finite set $V$ with $\Sigma_1 := \{z \to t \in \Sigma\}$ and $\Sigma_2 := \{zt \to y \in \Sigma\}$. Let $x$ in $V$ be an extreme point in $\Sigma_2$ and not in $\Sigma_1$ and let $\Sigma' := \big(\Sigma \setminus \{xz \to y \in \Sigma\}\big) \cup \big(\{tz \to y \mid xz \to y, t \to x \in \Sigma\}\big)$. Then, any minimum key $K'$ of $\Sigma'$ is a minimum key of $\Sigma$. Conversely, to any minimum key $K$ of $\Sigma$, one can associate a minimum key $K'$ of $\Sigma'$.*

We can therefore state the following.

**Theorem 21.** *For any connected chordal graph $G$, one can compute a minimum $\mathcal{SP}_G$-hull set of $G$ in time $O\left(\|G\| + \sum_{x,y \in V_G} \left|\mathcal{SP}^o_G(\{x,y\})\right|\right)$, which is less than $O(|V_G|^3)$. Therefore, one can compute a minimum $\mathcal{SP}_G$-hull set of any chordal graph $G$ in time $O(|V_G|^3)$.*

## 5 Concluding Remarks

In this paper we have given a linear time algorithm and a cubic-time algorithm for computing a minimum $\mathcal{SP}_G$-hull set in distance-hereditary and chordal graphs respectively. The techniques used to get these two algorithms are different from the ones known in the literature, specially those techniques based on functional dependencies borrowed from database community. We hope these techniques will be fruitful to obtain new algorithms for graph classes where the complexity of computing a minimum $\mathcal{SP}_G$-hull set is open. We can cite among them graph classes of bounded tree-width and more generally those of bounded clique-width, weakly chordal graphs, degenerate graphs, ...

We conclude by pointing out that our techniques for distance-hereditary graphs can be applied to compute a minimum $\mathcal{SP}_G$-hull set or $\mathcal{SP}_G$-geodetic set in other graph classes. We have for instance the following.

**Proposition 22.** *Let $k$ be a fixed integer. Let $G$ be an undirected graph such that $\mathcal{G}(\Sigma_G)$ has clique-width at most $k$. Then, one can compute in time $O(|V_G|^6)$ a minimum $\mathcal{SP}_G$-hull set and a minimum $\mathcal{SP}_G$-geodetic set of $G$.*

A question that arises then is which graphs have dependence graphs of bounded clique-width?

The logical tools can be also applied to other betweenness relations. For instance, for an undirected graph $G$, we let $\mathcal{P}_G$ be the betweenness relation where $\mathcal{P}_G(x,z,y)$ holds if and only if $z$ is in a chordless path between $x$ and $y$. We can prove the following.

**Proposition 23.** *Let $k$ be a fixed positive integer. Let $G$ be an undirected graph of clique-width at most $k$. Then, one can compute in time $O(|V_G|^3)$ a minimum $\mathcal{P}_G$-hull set and a minimum $\mathcal{P}_G$-geodetic set of $G$.*

## References

1. Araújo, J., Campos, V., Giroire, F., Nisse, N., Sampaio L., Soares, R.P.: On the hull number of some graph classes. Technical report (2011)
2. Bouchet, A.: Transforming trees by successive local complementations. J. Graph Theory 12(2), 195–207 (1988)
3. Chartrand, G., Fink, J.F., Zhang, P.: The hull number of an oriented graph. International Journal of Mathematics and Mathematical Sciences (36), 2265–2275 (2003)

4. Chvátal, V.: Antimatroids, betweenness, convexity. In: Cook, W.J., Lovász, L. (eds.) Research Trends in Combinatorial Optimization, pp. 57–64. Springer (2009)
5. Codd, E.F.: Further normalization of the data base relational model. In: Rustin, R. (ed.) Data Base Systems, pp. 33–64. Prentice Hall, Englewood Cliffs (1972)
6. Courcelle, B.: The monadic second-order logic of graphs XVI: Canonical graph decompositions. Logical Methods in Computer Science 2(2) (2006)
7. Courcelle, B., Engelfriet, J.: Graph Structure and Monadic Second-Order Logic: a Language Theoretic Approach. Encyclopedia of Mathematics and its Applications, vol. 138. Cambridge University Press (2012)
8. Cunningham, W.H., Edmonds, J.: A combinatorial decomposition theory. Canadian Journal of Mathematics 32, 734–765 (1980)
9. Dahlhaus, E.: Parallel algorithms for hierarchical clustering and applications to split decomposition and parity graph recognition. J. Algorithms 36(2), 205–240 (2000)
10. Diestel, R.: Graph Theory, 3rd edn. Springer (2005)
11. Dirac, G.A.: On rigid circuit graphs. Abhandlungen Aus Dem Mathematischen Seminare der Universität Hamburg 25(1-2), 71–76 (1961)
12. Dourado, M.C., Gimbel, J.G., Kratochvíl, J., Protti, F., Szwarcfiter, J.L.: On the computation of the hull number of a graph. Discrete Mathematics 309(18), 5668–5674 (2009)
13. Dourado, M.C., Protti, F., Rautenbach, D., Szwarcfiter, J.L.: On the hull number of triangle-free graphs. SIAM J. Discrete Math. 23(4), 2163–2172 (2010)
14. Dourado, M.C., Protti, F., Rautenbach, D., Szwarcfiter, J.L.: Some remarks on the geodetic number of a graph. Discrete Mathematics 310(4), 832–837 (2010)
15. Ekim, T., Erey, A., Heggernes, P., van 't Hof, P., Meister, D.: Computing Minimum Geodetic Sets of Proper Interval Graphs. In: Fernández-Baca, D. (ed.) LATIN 2012. LNCS, vol. 7256, pp. 279–290. Springer, Heidelberg (2012)
16. Everett, M.G., Seidman, S.B.: The hull number of a graph. Discrete Mathematics 57(3), 217–223 (1985)
17. Fulkerson, D.R., Gross, O.A.: Incidence Matrices and Interval Graphs. Pacific J. Math. 15(3), 835–855 (1965)
18. Gavoille, C., Paul, C.: Distance labeling scheme and split decomposition. Discrete Mathematics 273(1-3), 115–130 (2003)
19. Golumbic, M.C., Rotics, U.: On the clique-width of some perfect graph classes. Int. J. Found. Comput. Sci. 11(3), 423–443 (2000)
20. Pelayo, I.M.: On convexity in graphs. Technical report (2004)
21. Rose, D.J., Tarjan, R.E., Lueker, G.S.: Algorithmic aspects of vertex elimination on graphs. SIAM J. Comput. 5(2), 266–283 (1976)
22. Saiedian, H., Spencer, T.: An efficient algorithm to compute the candidate keys of a relational database schema. Comput. J. 39(2), 124–132 (1996)

# Permuted Pattern Matching on Multi-track Strings

Takashi Katsura[1], Kazuyuki Narisawa[1], Ayumi Shinohara[1], Hideo Bannai[2], and Shunsuke Inenaga[2]

[1] Graduate School of Information Science, Tohoku University, Japan
{katsura@shino,narisawa@,ayumi@}ecei.tohoku.ac.jp
[2] Department of Informatics, Kyushu University, Japan
{bannai,inenaga}@inf.kyushu-u.ac.jp

**Abstract.** We propose a new variant of pattern matching on a multi-set of strings, or multi-tracks, called permuted-matching, that looks for occurrences of a multi-track pattern of length $m$ with $M$ tracks, in a multi-track text of length $n$ with $N$ tracks over $\Sigma$. We show that the problem can be solved in $O(nN \log |\Sigma|)$ time and $O(mM + N)$ space, and further in $O(nN)$ time and space when assuming an integer alphabet. For the case where the number of strings in the text and pattern are equal (full-permuted-matching), we propose a new index structure called the multi-track suffix tree, as well as an $O(nN \log |\Sigma|)$ time and $O(nN)$ space construction algorithm. Using this structure, we can solve the full-permuted-matching problem in $O(mN \log |\Sigma| + occ)$ time for any multi-track pattern of length $m$ with $N$ tracks which occurs $occ$ times.

## 1 Introduction

Data that can be represented as multi-track strings, or a multi-set of strings, have recently been rapidly increasing in many areas, e.g., polyphonic music data, multiple sensor data, multiple genomes, etc. [9, 13–15]. Pattern matching on such multi-tracks is naturally a fundamental and important problem for retrieving useful information contained in these data. Since multi-tracks simply consist of multiple strings, pattern matching algorithms for a single string can be of some use to such data. However, in order to capture more meaningful characteristics which lie in the data, it is desirable to consider more sophisticated types of patterns. For example, when considering polyphonic music data, rather than finding a pattern in one of the parts of a multi-part music piece, it may be useful to look for patterns that can capture relationships between different parts.

Lemström et al. addressed the multi-track string pattern matching problem for music information retrieval [14, 15]. They considered the following type of matching: given a set of strings $\mathbb{T} = \{t_1, \ldots, t_N\}$ and a single pattern $p$ of length $m$, a pattern occurs at position $j$ if there exists $i_1, \ldots, i_m \in \{1, \ldots, N\}$ such that $t_{i_1}[j] = p[1]$, $t_{i_2}[j+1] = p[2]$, …, $t_{i_m}[j+m-1] = p[m]$. For example, given multi-track text $\mathbb{T} = \{t_1, t_2, t_3\} = \{\texttt{acccca}, \texttt{bccabc}, \texttt{babccc}\}$ and a single string pattern $p = \texttt{ababa}$, $p = t_3[2]t_3[3]t_2[4]t_2[5]t_1[6]$. Then, pattern $p$ occurs at

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 280–291, 2013.

position 2. Their algorithm solves this problem in $O(nN\lceil\frac{m}{w}\rceil)$ time, where $w$ is the size of machine word in bits.

In this paper, we consider a similar but distinct setting. While Lemström et al. considered matching a string pattern to a multi-track text, we try to match a multi-track pattern to a multi-track text, allowing the order of the pattern tracks to be permuted. Formally, given a multi-set of strings $\mathbb{T} = \{t_1, \ldots, t_N\}$ and a multi-set pattern $\mathbb{P} = \{p_1, \ldots, p_M\}$ where $M \leq N$ and $|p_i| = m$ for $1 \leq i \leq M$, $\mathbb{P}$ occurs at position $j$ of $\mathbb{T}$ if there exists a permutation $(i_1, \ldots, i_M)$ of a subsequence of $(1, \ldots, N)$ such that $p_1 = t_{i_1}[j : j + m - 1]$, ..., $p_M = t_{i_M}[j : j + m - 1]$, where $t_i[j : k]$ is the substring of $t_i$ from $j$ to $k$. $\mathbb{P}$ is then said to *permuted-match* $\mathbb{T}$.

The problem we consider somewhat resembles the two dimensional pattern matching problem proposed in the context of image retrieval [2, 4, 5]. In our case, however, we do not fix the order on the strings in the pattern and text multi-tracks for the matching, and therefore the problem becomes very different. Consider for example, multi-track text $\mathbb{T} = \begin{pmatrix} t_1, \\ t_2, \\ t_3 \end{pmatrix} = \begin{pmatrix} \texttt{ababa}, \\ \texttt{aabbb}, \\ \texttt{bbaab} \end{pmatrix}$ and multi-track pattern $\mathbb{P} = \begin{pmatrix} p_1, \\ p_2 \end{pmatrix} = \begin{pmatrix} \texttt{ab}, \\ \texttt{bb} \end{pmatrix}$. In two dimensional pattern matching, only the position at the first row and the third column is regarded as a match, since $(t_1[3:4], t_2[3:4]) = (\texttt{ab}, \texttt{bb}) = \mathbb{P}$. In our proposed permuted-matching, the multi-track pattern occurs at positions 1, 3, and 4, since $(t_1[1:2], t_3[1:2])$, $(t_1[3:4], t_2[3:4])$, and $(t_2[4:5], t_1[4:5])$ equal to $\mathbb{P} = (\texttt{ab}, \texttt{bb})$. The problem of computing all these positions can be solved in $O(nmN \log N)$ time by the following simple method: in each position $i$ of $\mathbb{T}$, we sort all substrings of tracks of $\mathbb{T}$ and all tracks of $\mathbb{P}$. Then, we consider whether each substring of tracks of the text matches each track of the pattern. Can we solve the problem more efficiently than in $O(nmN \log N)$ time?

The contribution of this paper is as follows: We show that the problem can be solved in $O(nN \log |\Sigma|)$ time and $O(mM+N)$ space for general alphabets, and in $O(nN)$ time and space for integer alphabets, where $|\Sigma|$ is the size of the alphabet. For the case where $M = N$ which we call full-permuted-matching, we propose a new index structure called the multi-track suffix tree and give an $O(nN \log |\Sigma|)$ time and $O(nN)$ space algorithm for its construction. Using this data structure, we can solve the full-permuted-matching problem in $O(mN \log |\Sigma| + occ)$ time, where $m$ is the length of the query pattern and $occ$ is the number of occurrences.

## 2 Preliminaries

### 2.1 Notation

Let $\Sigma$ be a finite set of symbols, called an *alphabet*. An element of $\Sigma^*$ is called a *string*. Let $\Sigma^n$ be the set of strings of length $n$. For a string $w = xyz$, strings $x$, $y$, and $z$ are called *prefix*, *substring*, and *suffix* of $w$, respectively. $|w|$ denotes the length of $w$, and $w[i]$ denotes the $i$-th symbol of $w$ for $1 \leq i \leq |w|$. Let $x \cdot y$, briefly

denote $xy$, be the *concatenation* of strings $x$ and $y$. Then, $w = w[1]w[2]\dots w[|w|]$. The substring of $w$ that begins at position $i$ and ends at position $j$ is denoted by $w[i:j]$ for $1 \le i \le j \le |w|$, i.e., $w[i:j] = w[i]\ w[i+1] \cdots w[j]$. Moreover, let $w[:i] = w[1:i]$ and $w[i:] = w[i:|w|]$ for $1 \le i \le |w|$. The empty string is denoted by $\varepsilon$, that is, $|\varepsilon| = 0$. For convenience, let $w[i:j] = \varepsilon$ if $i > j$. For two strings $x$ and $y$, we denote by $x \prec y$ if $x$ is lexicographically smaller than $y$, and by $x \preceq y$ if $x \prec y$ or $x = y$. For a set $S$, we denote the cardinality of $S$ by $|S|$. For a multi-set $S$, let $\#_S(x)$ denote the multiplicity of element $x \in S$.

### 2.2 Multi-track Strings

As was mentioned in the introductory section, our pattern matching problem is defined over a multi-set of strings. For ease of presentation, however, in what follows we assume an arbitrary order of the strings in the multi-set and define a multi-track string as a tuple.

Let $\mathbf{\Sigma}_N$ be the set of all $N$-tuples $(a_1, a_2, \dots, a_N)$ with $a_i \in \Sigma$ for $1 \le i \le N$, called a *multi-track alphabet*. An element of $\mathbf{\Sigma}_N$ is called a *multi-track character* (mt-character), and an element of $\mathbf{\Sigma}_N^*$ is called a *multi-track string* (or simply *multi-track*), where the concatenation between two multi-track strings is defined as $(a_1, a_2, \dots, a_N) \cdot (b_1, b_2, \dots, b_N) = (a_1b_1, a_2b_2, \dots, a_Nb_N)$.

For a multi-track $\mathbb{T} = (t_1, t_2, \dots, t_N) \in \mathbf{\Sigma}_N^n$, the $i$-th element $t_i$ of $\mathbb{T}$ is called the *$i$-th track*, the length of multi-track $\mathbb{T}$ is denoted by $\ell(\mathbb{T}) = |t_1| = |t_2| = \cdots = |t_N| = n$, and the number of tracks in multi-track $\mathbb{T}$ or the *track count* of $\mathbb{T}$, is denoted by $h(\mathbb{T}) = N$. Let $\mathbf{\Sigma}_N^+$ be the set of all multi-tracks of length at least 1, and let

$\mathbb{E}_N = (\varepsilon, \varepsilon, \dots, \varepsilon)$

denotes the *empty multi-track of track count $N$*. Then, $\mathbf{\Sigma}_N^* = \mathbf{\Sigma}_N^+ \cup \{\mathbb{E}_N\}$. For a multi-track $\mathbb{T} = \mathbb{X}\mathbb{Y}\mathbb{Z}$, multi-track $\mathbb{X}$, $\mathbb{Y}$, and $\mathbb{Z}$ are called *prefix*, *substring*, and *suffix* of $\mathbb{T}$, respectively. We call a prefix and suffix of a multi-track by an *mt-prefix* and *mt-suffix* of the multi-track, respectively. $\mathbb{T}[i]$ denotes the $i$-th mt-character of $\mathbb{T}$ for $1 \le i \le \ell(\mathbb{T})$, i.e., $\mathbb{T} = \mathbb{T}[1]\mathbb{T}[2]\dots\mathbb{T}[\ell(\mathbb{T})]$. The substring of $\mathbb{T}$ that begins at position $i$ and ends at position $j$ is denoted by $\mathbb{T}[i:j] = (t_1[i:j], t_2[i:j], \dots, t_N[i:j])$ for $1 \le i \le j \le \ell(\mathbb{T})$. Moreover, let $\mathbb{T}[:i] = \mathbb{T}[1:i]$ and $\mathbb{T}[i:] = \mathbb{T}[i:\ell(\mathbb{T})]$, respectively.

**Definition 1 (Permuted multi-track).** *Let $\mathbb{X} = (x_1, x_2, \dots, x_N)$ be a multi-track of track count $N$. Let $\mathbf{r} = (r_1, r_2, \dots, r_K)$ be a sub-permutation of $(1, \dots, N)$, where $1 \le K \le N$. A* permuted multi-track of $\mathbb{X}$ specified by $\mathbf{r}$ *is a multi-track $(x_{r_1}, x_{r_2}, \dots, x_{r_K})$, denoted by either $\mathbb{X}\langle r_1, r_2, \dots, r_K\rangle$ or $\mathbb{X}\langle\mathbf{r}\rangle$. If $K = N$, $\mathbf{r}$ is called a* full-permutation *and $\mathbb{X}\langle\mathbf{r}\rangle$ is called a* full-permuted multi-track *of $\mathbb{X}$.*

For multi-track $\mathbb{X} = (x_1, x_2, \dots, x_N)$, let $SI(\mathbb{X}) = (r_1, r_2, \dots, r_N)$ be a full-permutation such that $x_{r_i} \preceq x_{r_j}$ for any $1 \le i \le j \le h(\mathbb{X})$, and let $Sort(\mathbb{X}) = \mathbb{X}\langle SI(\mathbb{X})\rangle$.

**Definition 2 (Permuted-match).** *For any multi-tracks $\mathbb{X} = (x_1, x_2, \dots, x_{h(\mathbb{X})})$ and $\mathbb{Y} = (y_1, y_2, \dots, y_{h(\mathbb{Y})})$ with $\ell(\mathbb{X}) = \ell(\mathbb{Y})$ and $h(\mathbb{X}) \le h(\mathbb{Y})$,*

*we say that* $\mathbb{X}$ permuted-matches $\mathbb{Y}$*, denoted by* $\mathbb{X} \overset{\bowtie}{\sqsubseteq} \mathbb{Y}$*, if* $\mathbb{X} = \mathbb{Y}'$ *for some permuted multi-track* $\mathbb{Y}'$ *of* $\mathbb{Y}$*. Especially, if* $h(\mathbb{X}) = h(\mathbb{Y})$*, then we say that* $\mathbb{X}$ full-permuted-matches $\mathbb{Y}$*, and denote it by* $\mathbb{X} \overset{\bowtie}{=} \mathbb{Y}$*. Otherwise, i.e., if* $h(\mathbb{X}) < h(\mathbb{Y})$*, then we say that* $\mathbb{X}$ sub-permuted-matches $\mathbb{Y}$*.*

It holds that $\mathbb{X} \overset{\bowtie}{=} \mathbb{Y}$ if and only if $Sort(\mathbb{X}) = Sort(\mathbb{Y})$.

*Example 1.* For multi-tracks $\mathbb{T} = (t_1, t_2, t_3) = (\texttt{abab}, \texttt{abbb}, \texttt{abba})$ $\mathbb{X} = (x_1, x_2, x_3) = (\texttt{abba}, \texttt{abab}, \texttt{abbb})$, and $\mathbb{Y} = (y_1, y_2) = (\texttt{ba}, \texttt{ab})$, we see that $Sort(\mathbb{T}) = Sort(\mathbb{X}) = (\texttt{abab}, \texttt{abba}, \texttt{abbb})$, $SI(\mathbb{T}) = (1, 3, 2)$, $SI(\mathbb{X}) = (2, 1, 3)$, $\mathbb{T} \overset{\bowtie}{=} \mathbb{X}$, and $\mathbb{Y} \overset{\bowtie}{\sqsubseteq} \mathbb{T}[3:4]$.

The problem we consider is formally defined as:

*Problem 1 (Permuted-Matching).* Given multi-tracks $\mathbb{T}$ and $\mathbb{P}$, output all positions $i$ that satisfy $\mathbb{P} \overset{\bowtie}{\sqsubseteq} \mathbb{T}[i : i + m - 1]$.

When $h(\mathbb{T}) = h(\mathbb{P})$, the problem is called the *full*-permuted-matching problem, and when $h(\mathbb{T}) < h(\mathbb{P})$, it is called the *sub*-permuted-matching problem.

## 3 Linear Time Solutions

### 3.1 $O(nN \log |\Sigma|)$-Time $O(mM + N)$-Space Algorithm

We first describe an algorithm based on the Aho-Corasick (AC) automaton [1]. The AC automaton is a well known pattern matching automaton that can find occurrences of multiple pattern strings in a given text string. The automaton is traversed with each character $T[k]$ $(1 \leq k \leq |T|)$ of the text $T$, and if an accepting state is reached, it means that position $k$ is the end position of patterns that can be identified by the state. (We shall omit technical details of the traversal involving failure links in this paper, as they are identical to the original work.) It is known that the AC automaton can be constructed in time linear in the total length of the pattern strings [1, 7].

We solve the permuted-matching problem as follows. First, construct the AC automaton for the strings in the pattern multi-track $\mathbb{P} = (p_1, \ldots, p_M)$. Let $\hat{Q} = \{q_{i_1}, \ldots, q_{i_M}\}$ denote the multi-set of accepting states corresponding to each track of the pattern. Next, we traverse the automaton with each track of the text multi-track in a parallel manner. Let $Q = \{q_{j_1}, \ldots, q_{j_N}\}$ be the multi-set of states reached after traversing with $\mathbb{T}[k]$. Then, $\mathbb{P} \overset{\bowtie}{\sqsubseteq} \mathbb{T}[k - \ell(\mathbb{P}) + 1 : k]$ if and only if all the $M$ accepting states are included in $Q$, i.e., for any $q \in \hat{Q}$, $\#_Q(q) \geq \#_{\hat{Q}}(q)$. This can be easily checked in $O(N)$ time for each position. Since the traversal of the AC automaton can be conducted in $O(n \log |\Sigma|)$ time for each track, the total time required for the traversal is $O(nN \log |\Sigma|)$.

**Theorem 1.** *A multi-track pattern* $\mathbb{P} = (p_1, \ldots, p_M)$*, where* $\ell(\mathbb{P}) = m$*, can be preprocessed in* $O(mM \log |\Sigma|)$*-time and* $O(mM)$ *space so that the permuted-matching problem for any multi-track text* $\mathbb{T} = (t_1, \ldots, t_N)$ *where* $N \geq M$ *and* $\ell(\mathbb{T}) = n \geq m$*, can be solved in* $O(nN \log |\Sigma|)$ *time and* $O(N)$ *working space.*

### 3.2 $O(nN)$-Time and Space Algorithm for Integer Alphabets

Second, we describe an algorithm using the generalized suffix array for a text and a pattern and the longest common extension (LCE). The LCE between two positions $i, j$ respectively in strings $t$ and $p$, is $\max\{k \mid t[i : i+k-1] = p[j : j+k-1]\}$. Consider all strings in the text and pattern multi-tracks, and preprocess them in linear time so that LCE queries between arbitrary positions of arbitrary tracks can be answered in constant time (See e.g. [11]). The preprocessing can be done in $O(nN)$ time for integer alphabets. Notice that we can determine the lexicographic order between the two strings $t[i :]$ and $p[j :]$ in constant time as well, by comparing the characters $t[i+k]$ and $p[j+k]$ which come after the LCE. Next, we calculate $SI(\mathbb{T}[1 :]), \ldots, SI(\mathbb{T}[n :]), SI(\mathbb{P})$. This can be performed in $O(nN)$ time with the following lemma.

**Lemma 1.** *Given a multi-track $\mathbb{T} = (t_1, t_2, \ldots, t_N)$ of length $n$, the permutations $SI(\mathbb{T}[i :])$ for all $(1 \le i \le n)$ can be computed in $O(nN)$ time for an integer alphabet, or in $O(nN \log |\Sigma|)$ time for a general alphabet.*

*Proof.* We can determine the lexicographic order of all the $nN$ suffixes $t_1[1 :]$,..., $t_N[1 :]$, $t_1[2 :]$, ..., $t_N[2 :]$, $t_1[n :]$, ..., $t_N[n :]$ as follows: For integer alphabets, in $O(nN)$ time and space using suffix trees (e.g., [8]) or arrays (e.g., [12]), and for general alphabets, in $O(nN \log |\Sigma|)$ time and $O(nN)$ space using suffix trees (e.g., [16]).

The permutation $SI(\mathbb{T}[i :])$ for all $i$'s can be obtained by a simple linear scan on this order. For each $i$, the permutation $SI(\mathbb{T}[i :])$ corresponds to the order in which each track is encountered during the scan. Thus the lemma holds. □

Finally, for each position $1 \le i \le n - m + 1$ in the text, check whether or not $\mathbb{P} \overset{\text{pm}}{\sqsubseteq} \mathbb{T}[i :]$. This check can be conducted in $O(N + M)$ time for each position, by comparing each text track $\mathbb{T}[i :]$ and pattern track $\mathbb{P}$ in the order of $SI(\mathbb{T}[i :]) = (r_{i,1}, \ldots, r_{i,N})$ and $SI(\mathbb{P}) = (r_1, \ldots, r_M)$ respectively, to see if there exists $1 \le j_1 < \cdots < j_M \le N$ such that $t_{r_{i,j_k}}[i : i+m-1] = p_{r_k}$ for all $k$ $(1 \le k \le M)$. Each pair of text and pattern tracks can be checked in constant time using an LCE query. Since the tracks are checked in lexicographic order, the number of checks required is at most $O(N + M) = O(N)$. The total time for the algorithm is thus $O(nN)$.

**Theorem 2.** *Given multi-track text $\mathbb{T} = (t_1, \ldots, t_N)$ where $\ell(\mathbb{T}) = n$, and multi-track pattern $\mathbb{P} = (p_1, \ldots, p_M)$, where $M \le N$ and $\ell(\mathbb{P}) = m \le n$, the permuted-matching problem can be solved in $O(nN)$ time and space, assuming an integer alphabet.*

## 4 Suffix Trees for Multi-track Texts

In this section, we present a new data structure called a *multi-track suffix tree*, as well as an efficient algorithm for constructing it, with which we can solve the full-permuted multi-track matching problem efficiently. Namely, we will show that we

can construct the multi-track suffix tree for any text multi-track $\mathbb{T} = (t_1, \ldots, t_N)$ in $O(nN \log |\Sigma|)$-time and $O(nN)$ space where $\ell(\mathbb{T}) = n$. Using the multi-track suffix tree, we can conduct full permuted-matching for any multi-track pattern $\mathbb{P} = (p_1, \ldots, p_N)$, where $\ell(\mathbb{P}) = m$, in $O(mN \log |\Sigma| + occ)$ time.

The construction algorithm is based on Ukkonen's suffix tree construction algorithm [16] for a single string. Prior to introducing the multi-track suffix tree and its efficient construction algorithm, let us first recall the suffix tree for a single string and Ukkonen's construction algorithm.

### 4.1 Ukkonen's Suffix Tree Construction Algorithm

The suffix tree $STree(w)$ for a string $w$ is a compacted trie of all suffixes of $w$, where $w[:\, n-1] \in \Sigma^{n-1}$ and $w[n] = \$$ is lexicographically smaller than any other symbol in $\Sigma$. Using the suffix tree of a text, we can find occurrences of patterns in the text efficiently (see, e.g., [6, 10]). For a node $u$ and a descendant node $v$ of $u$, we denote by $label(u, v)$ the string that is the concatenation of the labels from $u$ to $v$. If $u$ is the root, then we abbreviate $label(u, v)$ by $label(v)$. If, for a substring $x$ of $w$, there is no explicit node $v$ with $label(v) = x$, then traversal from the root spelling out $x$ terminates on an edge, and in this case we say that $x$ is represented by an implicit node. Assume that for some node $v$, $label(v)$ is a prefix of $x$. We represent $x$ by a *reference pair* $(v, (b, e))$ such that $x = label(v)w[b : e]$. We can represent any implicit node as well as any explicit node using a reference pair. If $v$ is the deepest node such that $label(v)$ is a prefix of $w$, then the pair $(v, (b, e))$ is said to be *canonical*.

The *suffix link* of a node representing $ax$, with $a \in \Sigma$ and $x \in \Sigma^*$, is the directed edge from the node $v$ representing $ax$ to the node $u$ representing $x$, which is denoted by $sl(v) = u$. For the *root* node, we let $sl(root) = \perp$ for convenience, where $\perp$ is an auxiliary node.

The Ukkonen algorithm consists of $n+1$ phases. In the initial phase it creates the root, the auxiliary node, the edge and the suffix link between them. In phase $i \geq 1$, the algorithm updates $STree(w[:\, i-1])$ to $STree(w[:\, i])$. The update operation starts at a location in $STree(w[:\, i-1])$ called the *active point*, which we shall refer to as AP. The AP in the $i$-th phase initially is represented by a canonical reference pair $(v, (b, e))$ such that $label(v)w[b : e]$ is the longest suffix of $w[:\, i-1]$ that appears at least twice in $w[:\, i-1]$. Let $w[j : i-1]$ be this suffix. In the $i$th phase we execute the following procedure until finding the longest suffix of $w[:\, i]$ that appears at least twice in $w[:\, i]$.

If the AP is explicit (i.e., $b > e$), then we find the out-going edge of $v$ that begins with $w[e+1]$. If there is such an edge, then $w[j : i]$ is the longest suffix of $w[:\, i]$ that appears at least twice in $w[:\, i]$. Thus the AP is moved one character ahead along the edge, and the reference pair $(v, (b, e+1))$ is canonized. We go to the $(i+1)$th phase. If there is no such edge, then a new edge from $v$ to a new leaf $\ell$ is created, with the edge label $(e+1, \infty)$, and $label(\ell) = w[j : \infty]$, where $\infty$ represents an arbitrary integer such that $\infty \geq n$. Then the AP is moved to the location which represents $w[j+1 : i-1]$, which can be done by using $sl(v)$,

as $(sl(v), (b, e))$ represents $w[j+1 : i-1]$. This reference pair is then canonized, and we repeat the procedure.

If the AP is implicit (i.e., $b \leq e$), then we check if the next character in the edge label is equal to $w[i]$, i.e., $w[i] = w[e+1]$. If so, then the AP is moved one character ahead along the edge, and the reference pair $(v, (b, e+1))$ is canonized. We go to the $(i+1)$th phase. If not, then a new node $u$ for which $label(u) = label(v)w[b : e]$ is created, and a new edge from $u$ to a new leaf $\ell$ with edge label $(e+1, \infty)$ is created, where $label(\ell) = w[j+1 : \infty]$. If the node $q$ such that $label(q) = w[b - |label(v)| - 1 : e]$ (i.e., the node which may have been created in the previous step) has no suffix link yet, then we set $sl(q) = u$. Then the AP is moved to the location which represents $w[j+1 : i-1]$, which can be done by using $sl(v)$, as $(sl(v), (b, e))$ represents $w[j+1 : i-1]$. The reference pair is then canonized, and we repeat the procedure.

The cost for inserting a leaf $w[j : \infty]$ for each $1 \leq j \leq n$ is proportional to the cost for canonizing the reference pair. Ukkonen showed that the amortized cost for each canonization is $O(\log |\Sigma|)$, and hence $STree(w)$ is constructed in $O(n \log |\Sigma|)$ time.

### 4.2 Suffix Trees for Multi-track Texts

We propose a new data structure called the *multi-track suffix tree* (the mt-suffix tree, in short), which enables us to efficiently solve the problems concerning full-permuted-matchings on multi-tracks. For multi-track $\mathbb{T} = (t_1, t_2, \ldots, t_N)$ with $\ell(\mathbb{T}) = n$ and $h(\mathbb{T}) = N$, we assume that any track $t_i$ terminates with a special symbol $\$_i$, where each $\$_i \notin \Sigma$ is lexicographically smaller than any character in $\Sigma$, and $\$_1 \prec \$_2 \prec \cdots \prec \$_N$. The mt-suffix tree of a multi-track text $\mathbb{T}$, denoted $MTSTree(\mathbb{T})$, is a compacted trie of all sorted mt-suffixes of $\mathbb{T}$, i.e. $Sort(\mathbb{T}[1 :])$, $Sort(\mathbb{T}[2 :])$, $\ldots$, $Sort(\mathbb{T}[n :])$. Fig 1 illustrates $MTSTree(\mathbb{T})$ with $\mathbb{T} = (\texttt{abababb\$}_1, \texttt{aaababa\$}_2, \texttt{babaaab\$}_3)$, where dot-lines denote the suffix links that we will describe later.

The following lemma shows a simple but important observation regarding multi-track text $\mathbb{T}$.

**Lemma 2.** *For any multi-track strings* $\mathbb{X} \in \Sigma_N^+$ *and* $\mathbb{Y}, \mathbb{Z} \in \Sigma_N^*$, $Sort(\mathbb{X}\mathbb{Y})[1 : \ell(\mathbb{X})] = Sort(\mathbb{X}\mathbb{Z})[1 : \ell(\mathbb{X})]$.

*Proof.* Let $\mathbb{X} = (x_1, x_2, \ldots, x_N)$ and $\mathbb{Y} = (y_1, y_2, \ldots, y_N)$. For any $1 \leq i \neq j \leq N$, we consider the following two cases: (1) if $x_i \neq x_j$: We assume w.l.o.g. that $x_i \prec x_j$. Then $x_i y_i \prec x_j y_j$. (2) if $x_i = x_j$: We assume w.l.o.g. that $y_i \preceq y_j$. Then although $x_i y_i \preceq x_j y_j$, we have $x_i y_i[1 : \ell(\mathbb{X})] = x_j y_j[1 : \ell(\mathbb{X})]$. In either case, the lexicographical order between $x_i y_i$ and $x_j y_j$ depends only on the prefixes $x_i$ and $x_j$. It is same for $x_i z_i$ and $x_j z_j$. Thus, $Sort(\mathbb{X}\mathbb{Y})[1 : \ell(\mathbb{X})] = Sort(\mathbb{X}) = Sort(\mathbb{X}\mathbb{Z})[1 : \ell(\mathbb{X})]$. □

Due to the above lemma, for any substring $\mathbb{X}$ of $\mathbb{T}$, there is a path that spells out $Sort(\mathbb{X})$ in $MTSTree(\mathbb{T})$. This also implies that a multi-track pattern $\mathbb{P}$ with $h(\mathbb{P}) = h(\mathbb{T})$ has an occurrence in $\mathbb{T}$ iff there is a path that spells out $Sort(\mathbb{P})$ in $MTSTree(\mathbb{T})$. More in detail, we can solve the full-permuted-matching as follows:

**Theorem 3.** *Let $\mathbb{T} = (t_1, \ldots, t_N)$ with $\ell(\mathbb{T}) = n$. We can augment $MTSTree(\mathbb{T})$ in $O(nN \log |\Sigma|)$ time and $O(nN)$ space so that the full-permuted-matching problem for any given multi-track pattern $\mathbb{P} = (p_1, \ldots, p_N)$ with $\ell(\mathbb{P}) = m$ can be solved in $O(mN \log |\Sigma| + occ)$ time.*

*Proof.* First we compute $Sort(\mathbb{P})$ in $O(mN \log |\Sigma|)$ time using a trie that represents all tracks in $\mathbb{P}$. We then traverse down $MTSTree(\mathbb{T})$ from the root to search for a path that corresponds to $Sort(\mathbb{P})$. At each edge of $MTSTree(\mathbb{T})$, comparisons of mt-characters between $\mathbb{T}$ and $\mathbb{P}$ can be done in $O(N)$ time provided that $SI(\mathbb{T}[1:]), \ldots, SI(\mathbb{T}[n:])$ have been already computed by Lemma 1 in $O(nN \log |\Sigma|)$ time and $O(nN)$ space. Since the number of children of a node in $MTSTree(\mathbb{T})$ is $O(n)$, a straightforward search for occurrences of $\mathbb{P}$ takes a total of $O(mN \log n + occ)$ time. However, we can reduce the cost to $O(mN \log |\Sigma| + occ)$ by the following preprocessing on $\mathbb{T}$ : In each node of $MTSTree(\mathbb{T})$ we maintain a trie that represents the first mt-characters of the labels of the node. Since each mt-character in $\mathbb{T}$ is of length $N$, searching at each node can be done in $O(N \log |\Sigma|)$ time. Since there are $O(n)$ edges in $MTSTree(\mathbb{T})$, these tries for all nodes occupy a total of $O(nN)$ space and can be constructed in a total of $O(nN \log |\Sigma|)$ time. □

Clearly $MTSTree(\mathbb{T})$ has $n$ leaves and each internal node has more than one child node, and hence the number of all nodes of $MTSTree(\mathbb{T})$ is at most $2n - 1$ and the number of all edges is at most $2n - 2$. Each edge of $MTSTree(\mathbb{T})$ is labeled by $\mathbb{S}$ such that $\mathbb{T}[b:e]\langle SI(\mathbb{T}[j:])\rangle = \mathbb{S}$, and the label is represented by a triple $(b, e, j)$. Thus we can represent $MTSTree(\mathbb{T})$ in a total of $O(nN)$ space. Note that representing edge labels using the triples does not increase the asymptotic time complexity of the pattern matching algorithm of Theorem 3.

We shall use the next lemma to show the correctness of our algorithm which constructs $MTSTree(\mathbb{T})$.

**Lemma 3.** *Let $\mathbb{AX}$ be a substring of $\mathbb{T}$ with $\mathbb{A} \in \Sigma_N$ and $\mathbb{X} \in \Sigma_N^*$. There are paths that spell out $Sort(\mathbb{AX})$ and $Sort(\mathbb{X})$ from the root of $MTSTree(\mathbb{T})$.*

*Proof.* Let $\mathbb{Y} \in \Sigma_N^*$ be a multi-track s.t. $\mathbb{AXY}$ is a suffix of $\mathbb{T}$. Then there is a path spelling out $Sort(\mathbb{AXY})$ from the root of $MTSTree(T)$. By Lemma 2, $Sort(\mathbb{AX}) = Sort(\mathbb{AXY})[1 : \ell(\mathbb{AX})]$, and therefore there is a path spelling out $Sort(\mathbb{AX})$ from the root. Since $\mathbb{XY}$ is also a suffix of $\mathbb{T}$, paths spelling out $Sort(\mathbb{XY})$ and $Sort(\mathbb{X})$ from the root exist for the same reasoning as above. □

### 4.3 Construction of Multi-track Suffix Tree

In this subsection we propose an efficient $O(nN \log |\Sigma|)$-time algorithm that constructs the mt-suffix tree. The algorithm consists of two parts (a pseudo-code is shown in Algorithm 1). In the first part, we compute the permutation for the sorted mt-suffixes of $\mathbb{T}$ of each length, $SI(\mathbb{T}[1:]), \ldots, SI(\mathbb{T}[n:])$. In the second part, we construct the mt-suffix tree by using the permutations above.

Assume that for a node $v$ of $MTSTree(T)$ and a substring $\mathbb{X}$ of $\mathbb{T}$, $label(v)$ is a prefix of $Sort(\mathbb{X})$. We represent $Sort(\mathbb{X})$ by a *reference pair* $(v, (b, e, j))$ such

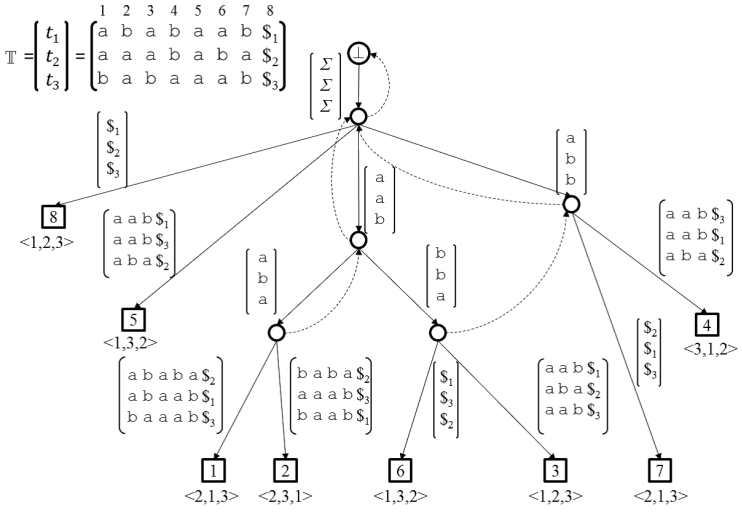



**Fig. 1.** The multi-track suffix tree $MTSTree(\mathbb{T})$ of a multi-track $\mathbb{T} = (\texttt{ababaab\$}_1, \texttt{aaababa\$}_2, \texttt{babaaab\$}_3)$

that $Sort(\mathbb{X}) = label(v)\mathbb{T}[b : e]\langle SI(\mathbb{T}[j :])\rangle$. If $v$ is the deepest node such that $label(v)$ is a prefix of $Sort(\mathbb{X})$, then the pair $(v, (b, e, j))$ is said to be *canonical*.

We define the suffix link on the mt-suffix tree as follows:

**Definition 3 (Multi-track Suffix Link).** *Let $slink(root) = \bot$. For any non-root explicit node $v$ of $MTSTree(\mathbb{T})$, if $label(v) = Sort(\mathbb{A}\mathbb{X})$ where $\mathbb{A} \in \mathbf{\Sigma}_N$, $\mathbb{X} \in \mathbf{\Sigma}_N^*$ and $\mathbb{A}\mathbb{X}$ is a substring of $\mathbb{T}$, then $slink(v) = u$, where $label(u) = Sort(\mathbb{X})$.*

We remark the suffix link of a node of $MTSTree(\mathbb{T})$ may point to an implicit node:

Let $\mathbb{A}\mathbb{X}$ be any substring of $\mathbb{T}$ with $\mathbb{A} \in \Sigma_N$ and $\mathbb{X} \in \Sigma_N^*$. If $Sort(\mathbb{A}\mathbb{X})$ is an explicit node $v$ of $MTSTree(\mathbb{T})$, then there are at least two distinct non-empty multi-track strings $\mathbb{Y}, \mathbb{Z} \in \Sigma_N^+$ for which there exist paths spelling out $Sort(\mathbb{A}\mathbb{X}\mathbb{Y})$ and $Sort(\mathbb{A}\mathbb{X}\mathbb{Z})$ from the root. It follows from Lemma 3 that there also exist paths from the root which spell out $Sort(\mathbb{X}\mathbb{Y})$ and $Sort(\mathbb{X}\mathbb{Z})$. However, $Sort(\mathbb{X}\mathbb{Y})[: \ell(\mathbb{X}) + 1]$ may or may not be equal to $Sort(\mathbb{X}\mathbb{Z})[: \ell(\mathbb{X}) + 1]$. Thus, $Sort(\mathbb{X})$, which is pointed to by $slink(v)$, can be an implicit node.

Still, we can design an algorithm to construct $MTSTree(\mathbb{T})$ for a given multi-track text $\mathbb{T}$ based on the Ukkonen algorithm [16], using similar techniques to construction of parameterized suffix trees [3].

In the initial phase, we create $MTSTree(\mathbb{E}_N)$ that consists only of the root node, the auxiliary node, and the edge and the suffix link between them (in Lines 1 and 1). In phase $i \geq 1$, our algorithm updates $MTSTree(\mathbb{T}[1 : i-1])$ to $MTSTree(\mathbb{T}[1 : i])$. The second part basically follows Ukkonen's suffix tree construction algorithm, with a distinction that when we insert the $j$th suffix $Sort(\mathbb{T}[j :])$ into the tree, then the $i$th mt-character $\mathbb{T}[i]$ is read *with the*

**Algorithm 1.** Algorithm to construct multi-track suffix trees

**Input**: multi-track text $\mathbb{T}$ of length $n$ and of track count $N$

compute $SI(\mathbb{T}[i:])$ for all $1 \le i \le n$;
create nodes *root* and $\perp$, and an edge with label $\Sigma_N$ from $\perp$ to *root*;
$slink(root) := \perp$;
$(v, (b, e, j)) := (root, (1, 0, 1))$; $oldu := root$;
**foreach** $i = 1, \cdots, n$ **do**
  **while** $j \le n$ **do**
    **if** $b \le e$ *and* $\mathbb{T}[i]\langle SI(\mathbb{T}[j:])\rangle = \mathbb{T}[e+1]\langle SI(\mathbb{T}[b:])\rangle$ **then**
      $(v, (b, e, j)) := canonize(v, (b, e+1, j))$;
      **break**;
    **if** $b > e$ *and there is a* $\mathbb{T}[i]\langle SI(\mathbb{T}[j:])\rangle$*-edge from* $v$ **then**
      $(v, (b, e, j)) := canonize(v, (i, i, j))$;
      **break**;
    $u := v$;
    **if** $b \le e$ **then**
      let $(v, (p, q, h), z)$ be the $\mathbb{T}[b]\langle SI(\mathbb{T}[j:])\rangle$-edge from $v$;
      split the edge to two edges $(v, (p, p+e-b, h), u)$ and $(u, (p+e-b+1, q, h), z)$;
      **if** $oldu \ne root$ **then** $slink(oldu) := u$;
    create a new edge $(u, (i, \infty, j), \ell)$ with a new leaf $\ell$;
    $oldu := u$;
    **if** $slink(u)$ *is defined* **then**
      $v := slink(u)$; $j := j + 1$ ;
    **else** $(v, (b, e, j)) := canonize(slink(v), (b, e, j+1))$;

**Algorithm 2.** Function *canonize*

**Input**: reference pair $(v, (b, e, j))$ for mt-substring $\mathbb{X} = label(v) \cdot \mathbb{T}[b:e]\langle \mathbb{T}[j:]\rangle$.
**Output**: canonical reference pair for mt-substring $\mathbb{X}$.

**if** $b > e$ **then return** $(v, (b, e, j))$;
find the $\mathbb{T}[b]\langle \mathbb{T}[j:]\rangle$-edge $(v, (p, q, h), u)$ from $v$;
**while** $q - p \le e - b$ **do**
  $b := b + q - p + 1$; $v := u$;
  **if** $b \le e$ **then** find the $\mathbb{T}[b]\langle \mathbb{T}[j:]\rangle$-edge $(v, (p, q, h), u)$ from $v$;
**return** $(v, (b, e, j))$;

*permutation of the jth mt-suffix*, i.e., as $\mathbb{T}[i]\langle SI(\mathbb{T}[j:])\rangle$. The AP for the $i$th phase initially corresponds to the longest mt-suffix $Sort(\mathbb{T}[j : i-1])$ of $\mathbb{T}[: i-1]$ that matches at least two positions $1 \le k \ne h \le i-1$ of $\mathbb{T}[1 : i-1]$, that is, $Sort(\mathbb{T}[k : k+i-j-1]) = Sort(\mathbb{T}[j : i-1])$ and $Sort(\mathbb{T}[h : h+i-j-1]) = Sort(\mathbb{T}[j : i-1])$. Then, we examine whether it is possible to traverse down from the AP with $\mathbb{T}[i]\langle SI(\mathbb{T}[j:])\rangle$. The examination is conducted in Line 1 when the AP is on an implicit node, and in Line 1 when the AP is on an explicit node. If it turns out to be possible, then after the reference pair is canonized, we go to

the $(i+1)$th phase. If it turns out to be impossible, then we create a new node if the reference pair represents an implicit node (Line 1). Then a new leaf node representing $Sort(\mathbb{T}[j:])$ is inserted in Line 1, whose parent node is $u$. The AP is then moved using the suffix link of $u$ if it exists (in Line 1), and using the suffix link of the parent $v$ of $u$ if the suffix link of $u$ does not exist yet (in Line 1), and we go to the $(j+1)$th step to insert $Sort(\mathbb{T}[j+1:])$. If follows from the definition of the suffix link and from Lemma 3 that there always exists a path that spells out $\mathbb{T}[b:e]\langle SI(T[j+1:])\rangle$ from node $slink(v)$ in Line 1.

**Theorem 4.** *Given a multi-track* $\mathbb{T}$ *which* $\ell(\mathbb{T}) = n$ *and* $h(\mathbb{T}) = N$*, Algorithm 1 constructs* $MTSTree(\mathbb{T})$ *in* $O(nN \log |\Sigma|)$ *time and* $O(nN)$ *space.*

*Proof.* The correctness follows from the above arguments.

Let us analyze the time complexity. In the first part of Algorithm 1 (in Line 1), the permutations for all sorted mt-suffixes of $\mathbb{T}$ can be computed in $O(nN)$ time due to Lemma 1. Next, we consider the time complexity of the second part. The condition of Line 1 can be checked in constant time using the permutations for sorted suffixes, and that of Line 1 can be checked in $O(N \log |\Sigma|)$ time using the same way to Theorem 3. At each step the function *canonize* is called at most once. A pseudo-code for *canonize* is given in Algorithm 2. Using a similar analysis to the Ukkonen algorithm [16], the amortized number of nodes that are visited at each call of *canonize* is constant. Since it takes $O(N \log |\Sigma|)$ time to search a branching node (in Lines 2 and 2), the amortized cost for canonizing a reference pair is $O(N \log |\Sigma|)$. We consider the number of steps in the double loop. Both $i$ and $j$ are monotonically non-decreasing, and they satisfy $1 \leq i$ and $j \leq n$. Therefore the total number of steps of the algorithm is $O(n)$. Thus, the second part of the Algorithm 1 takes $O(nN \log |\Sigma|)$ time, and hence the entire algorithm works in a total of $O(nN \log |\Sigma|)$ time.

The permutations for each sorted mt-suffix require $O(nN)$ space, and the tries for searching branches of each node require a total of $O(nN)$ space. Hence the overall space complexity is $O(nN)$. □

## 5 Conclusion

We introduced a new form of string pattern matching called the permuted-matching on multi-track strings. We showed that the permuted-matching problem on multi-track strings can be solved in $O(nN \log |\Sigma|)$ time and $O(mM+N)$ space using the AC-automaton. Furthermore, by using constant time longest common extension queries after linear time pre-processing, we can solve the problem in $O(nN)$ time and space for integer alphabets. However, these solutions do not allow a linear pre-processing of the text multi-tracks so that pattern matching cannot be performed in worst case linear time with respect to the pattern length plus output size, as do various string indices (e.g., suffix trees, suffix arrays) for normal string pattern matching.

For this problem, we proposed a new indexing structure called multi-track suffix trees (mt-suffix tree). Given the mt-suffix tree for a text multi-track, we can

solve the full-permuted-matching (i.e., $M = N$) problem for any pattern multi-track in $O(mN \log |\Sigma| + occ)$ time, where $occ$ is the number of positions of the text where the pattern permuted-matches. We also developed an algorithm for constructing the mt-suffix tree, based on the Ukkonen algorithm [16], running in $O(nN \log |\Sigma|)$ time and $O(nN)$ space. For constant size alphabets, the proposed algorithm performs in optimal linear time in the total size of the input texts.

The problem of developing a text index that can be used for solving sub-permuted-matching (i.e., $M < N$) in $O(mM \log |\Sigma| + occ)$ time is an open problem. Boyer-Moore type algorithms for permuted-matching may also be of interest for further research.

**Acknowledgements.** This work was partially supported by KAKENHI Grant Numbers 23300051, 23220001, 23650002, 22680014, and 23700022.

## References

1. Aho, A., Corasick, M.: Efficient string matching: an aid to bibliographic search. Communications of the ACM 18(6), 333–340 (1975)
2. Amir, A., Farach, M.: Efficient 2-dimensional approximate matching of non-rectangular figures. In: Proc. SODA 1991, pp. 212–223 (1991)
3. Baker, B.S.: Parameterized pattern matching: Algorithms and applications. J. Comput. Syst. Sci. 52(1), 28–42 (1996)
4. Baker, T.P.: A technique for extending rapid exact-match string matching to arrays of more than one dimension. SIAM Journal on Computing 7(4), 533–541 (1978)
5. Bird, R.S.: Two dimensional pattern matching. Information Processing Letters 6(5), 168–170 (1977)
6. Crochemore, M., Rytter, W.: Jewels of Stringology. World Scientific (2002)
7. Dori, S., Landau, G.M.: Construction of Aho Corasick automaton in linear time for integer alphabets. Information Processing Letters 98(2), 66–72 (2006)
8. Farach, M.: Optimal suffix tree construction with large alphabets. In: Proc. FOCS 1997, pp. 137–143 (1997)
9. Gandhi, S., Nath, S., Suri, S., Liu, J.: Gamps: Compressing multi sensor data by grouping and amplitude scaling. In: ACM SIGMOD (2009)
10. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press (1997)
11. Ilie, L., Navarro, G., Tinta, L.: The longest common extension problem revisited and applications to approximate string searching. Journal of Discrete Algorithms 8(4), 418–428 (2010)
12. Kärkkäinen, J., Sanders, P., Burkhardt, S.: Linear work suffix array construction. J. ACM 53(6), 918–936 (2006)
13. Kuruppu, S., Puglisi, S.J., Zobel, J.: Relative Lempel-Ziv Compression of Genomes for Large-Scale Storage and Retrieval. In: Chavez, E., Lonardi, S. (eds.) SPIRE 2010. LNCS, vol. 6393, pp. 201–206. Springer, Heidelberg (2010)
14. Lemström, K., Mäkinen, V.: On minimizing pattern splitting in multi-track string matching. In: Proc. of CPM 2003, pp. 237–253 (2003)
15. Lemström, K., Tarhio, J.: Transposition invariant pattern matching for multi-track strings. Nordic Journal of Computing 10, 185–205 (2003)
16. Ukkonen, E.: On-line construction of suffix trees. Algorithmica 14(3), 249–260 (1995)

# Online and Quasi-online Colorings of Wedges and Intervals

Balázs Keszegh[1,⋆], Nathan Lemons[2], and Dömötör Pálvölgyi[3,⋆⋆]

[1] Rényi Institute of Mathematics, Hungarian Academy of Sciences
keszegh.balazs@renyi.mta.hu
[2] Los Alamos National Laboratory, T-5 and CNLS
nlemons@lanl.gov
[3] Department of Computer Science, Eötvös University
dom@cs.elte.hu

**Abstract.** We consider proper online colorings of hypergraphs defined by geometric regions. We prove that there is an online coloring method that colors $N$ intervals of the real line using $\Theta(\log N/k)$ colors such that for every point $p$, contained in at least $k$ intervals, not all the intervals containing $p$ have the same color. We also prove the corresponding result about online coloring quadrants in the plane that are parallel to a given fixed quadrant. These results contrast to recent results of the first and third author showing that in the quasi-online setting 12 colors are enough to color quadrants (independent of $N$ and $k$). We also consider coloring intervals in the quasi-online setting. In all cases we present efficient coloring algorithms as well.

## 1 Introduction

The study of proper colorings of geometric hypergraphs has attracted much attention, not only because this is a very basic and natural theoretical problem but also because such problems often have important applications. One such application area is resource allocation: to determine the number of CPUs necessary to run several jobs, each with fixed starting and stopping times is exactly the problem of finding the chromatic number of the associated interval graph. Similarly, the coloring of geometric shapes in the plane is related to the problems of cover decomposability and conflict free colorings; these problems have applications in sensor networks and frequency assignment as well as other areas. For surveys on these and related problems see [23,25,22].

Despite the well-known applications of the study of colorings of geometric graphs and hypergraphs, relatively little attention has been paid to the online

⋆ Research supported by OTKA, grant NK 78439 and by OTKA under EUROGIGA project GraDR 10-EuroGIGA-OP-003.
⋆⋆ Research supported by OTKA, grant PD 104386, by OTKA under EUROGIGA project GraDR 10-EuroGIGA-OP-003 and the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 292–306, 2013.

and quasi online versions of these problems. In online coloring problems, the set of objects to be colored is not known beforehand; objects come to be colored one-by-one and a proper coloring must be maintained at all times. We give asymptotically tight bounds on the number of colors necessary to properly online color wedges in the plane and then relate these to bounds on the number of colors necessary to properly online color intervals.

In quasi-online coloring, objects come online and must be colored one by one, such that a valid coloring is maintained at each step, yet the objects and their order are known in advance. Such problems can be used to solve corresponding offline higher dimensional problems. In [11] it was shown that coloring wedges in the plane in a quasi-online manner is equivalent to coloring wedges in the space offline. This motivated us to revisit a problem about quasi-online coloring intervals.

### 1.1 Definitions and Description of Main Results

A *wedge* in the plane is the set of points $\{(x, y) \in \mathbb{R} \times \mathbb{R} | x \leq x_0\ y \leq y_0\}$ for fixed $x_0, y_0$. We are interested in coloring with $c$ colors a set of wedges (or intervals) such that for any point contained in at least $k$ wedges (respectively intervals) not all the wedges (intervals) containing that point have the same color. We will often refer to this as simply a coloring of the set of points; the parameters $k$ and $c$ will be obvious from the context. For a collection of $c$-colored points in the plane, we define the associated *color-vector* to be a vector of length $c$ where the $i^{\text{th}}$ coordinate is the size of the largest wedge consisting only of points with color $i$. The size of the color-vector refers to the sum of its coordinates.

Note that given a set of wedges (or intervals) we can define a hypergraph whose vertices are the wedges (intervals) and whose edges consist exactly of those subsets of the vertices such that there exists a point contained exactly in the corresponding wedges (intervals.) Then the coloring problem above is exactly the problem of finding a proper coloring of this hypergraph if we disregard those edges which contain less than $k$ points.

For convenience we will work with the equivalent dual-form of the wedge-coloring: a finite set of points $S$ in the plane is called *$k$-properly $c$-colored* if $S$ is colored with $c$ colors such that every wedge intersecting $S$ in at least $k$ points contains at least two points of different colors. A wedge containing points of only one color is said to be *monochromatic*.

In [11] the following theorem was proved.

**Theorem 1.** *[11] Any finite family of wedges in the plane can be colored quasi-online with* 2 *colors such that any point contained by at least* $k = 12$ *of these wedges is not monochromatic.*

Gábor Tardos [26] asked whether such a coloring can be achieved in a completely online setting, possibly with a larger $k$ and more colors (again such that all large wedges are non-monochromatic). It is easy to see that 2 colors are not enough to guarantee colorful wedges (i.e. there may be arbitrarily large monochromatic wedges) even when the points are restricted to a diagonal line. However, 3 colors

(and $k = 2$) are enough if the points are restricted to a diagonal, see [11]. We prove that in general, for any $c$ and $k$, there exists a method of placing points in the plane such that any online-coloring of these points with $c$ colors will result in the creation of monochromatic wedges of size at least $k$.

Next we consider the cases when either $c$ or $k$ is fixed. In the former we derive upper bounds on $k$ (in terms of $n$) for which there is always a $k$-proper online $c$-coloring of a set of $n$ points. For fixed $k$, we derive bounds on $c$ for which there is always a $k$-proper online $c$-coloring of $n$ points.

In Section 2.2 we show how our results on online colorings of wedges directly relates to the online coloring of intervals and in the Section 3 we describe our results on quasi-online colorings of intervals.

## 2 Online Coloring Wedges and Intervals

### 2.1 Online Coloring Wedges

Our first result is a negative answer to the question of Tardos [26]: for every $c$ and $k$, there exists a method of placing points in the plane such that any online-coloring of these points with $c$ colors will result in the creation of monochromatic wedges of size at least $k$. Actually we prove a stronger statement.

**Theorem 2.** *There exists a method to give $N = 2^n - 1$ points in a sequence such that for any online-coloring method using $c$ colors there will be $c$ monochromatic wedges, $W_1, W_2, \ldots, W_c$, and nonnegative integers $x_1, \ldots, x_c$ such that for each $i$, the wedge $W_i$ contains exactly $x_i$ points colored with color $i$ and $\sum x_i \geq n+1$ if $n \geq 2$.*

**Corollary 1.** *No online-coloring method using $c$ colors can avoid to make a monochromatic wedge of size $k+1$ for some sequence of $N = 2^{ck} - 1$ points.*

*Proof (of Theorem 2).* By induction on the size of the color-vector. Clearly, one point gives a color-vector of size 1. Two points guarantee a color-vector of size 2 if they are placed diagonally from each other. Now we can place the third point diagonally between the first two if they had a different color or place it diagonally below them to get a color-vector of size 3 for three points.

By the inductive hypothesis, using at most $2^{n-1} - 1$ points, we can force a color-vector with size $n$. Moving southeast (i.e. so that all the new points are southeast from the previous ones) we repeat the procedure, again using at most $2^{n-1} - 1$ points we can force a second color-vector with sum $n$. If the two color-vectors are different then the whole point set has a color-vector of size at least $n + 1$. If they are the same then we put an additional point southwest from all the points of the first set of points but not interfering with the second set of points. Then as this point is colored with some color, $i$, the $i^{\text{th}}$ coordinate of the first color-vector increases. (The rest of its coordinates will becomes 0.) Together with the wedges found in the second set of points, we can see that the size of the color-vector of the whole point set increased by one. Altogether we used at most $2(2^{n-1} - 1) + 1 = 2^n - 1$ points, as desired.

What happens if $c$ or $k$ is fixed? The case when $c = 2$ was considered, e.g., in [11]. It is not hard to see that using $2k - 1$ points, the size of the largest monochromatic wedge can be forced to be at least $k$ and this is the best possible. For $c = 3$ a quadratic (in $k$) number of points is needed to force a monochromatic wedge of size $k$:

**Proposition 1.** *The following statements hold.*

1. *There exists a method of placing $k^2$ points such that any online 3-coloring of these points produces a monochromatic wedge of size at least $k$.*
2. *There exists a method of online 3-coloring $k^2 - 1$ points such that all monochromatic wedges have size less than $k$.*

We use the following terminology. Given a collection of 3-colored points in the plane, we say a new, uncolored point $x$ is a *potential member* of a monochromatic wedge $W$, if by giving $x$ the color of $W$, the size of $W$ increases. Furthermore if $x$ is a potential member of $W$, and giving $x$ a color different from the color of $W$ destroys the wedge $W$, then $x$ *threatens* the wedge $W$.

*Proof.* To prove the first statement, consider the largest monochromatic wedge of each color after some points have already been placed and colored. Moving in the northwest / southeast directions label the wedges $W_1, W_2$ and $W_3$. It is clear that $W_2$ lies between the other two wedges. Note that it is possible to place a new point directly southwest of the points in $W_2$ such that the point is a potential member of all three wedges but only threatens $W_2$. Thus if the point is assigned the color of one of the other wedges (say $W_1$), the size of $W_1$ increments while $W_3$ remains the same and $W_2$ is destroyed (it is no longer monochromatic). Now suppose $W_2$ is not larger than either of the other two wedges. In this case, a point is placed, as described above, such that it is a potential member of all three wedges. If the point is assigned the color of $W_1$ or $W_3$ then $W_2$ is destroyed and while $W_1$ (or $W_3$) moves from size $i$ to $i + 1$, the $j \leq i$ points of $W_2$ are rendered ineffective for forming monochromatic wedges. On the other hand suppose $W_2$ has size larger than (at least) one of the other wedges (say $W_3$). Then we forget the wedge $W_3$ and proceed as above where there is a new wedge $W_0$ (of size 0) between $W_1$ and $W_2$. (We can think that in the previous step $W_2$ increased from $i$ to $i + 1$ while the $j \leq i$ points of $W_3$ are destroyed.) As we proceed in this way, the sizes of the two "side" wedges only increase at each step while the "middle" wedge may be reduced to size 0 at some steps. However, the $j$ points of the middle wedge are only destroyed when a side wedge increases from $i$ to $i + 1$ when $i \geq j$. Thus by destroying at most $2\binom{k}{2}$ points we can guarantee that the two side wedges have size $k - 1$. Adding at most $k$ more points to the middle, a monochromatic wedge of size $k$ is guaranteed.

To prove the second statement we must assign colors to the points to avoid a monochromatic $k$-wedge. When a new point, $x$ is given, consider those wedges of which $x$ is a potential member. Note that at most two of these are not threatened by $x$. Let $s$ be the size of the smallest wedge, $W$ which is not threatened by $x$ but of which $x$ is a potential member. If by giving $x$ the color of $W$ at least $s$

points are destroyed among the wedge(s) threatened by $x$, then give $x$ this color. Otherwise give $x$ the color different from the two non-threatened wedges. In this way we guarantee that a wedge only increases in size from $i$ to $i+1$ if at the same time $i$ other points are destroyed (i.e. rendered ineffective) or if two other wedges of size $i+1$ already exist. Therefore if only $2\sum_{i=1}^{k-2} i + 3(k-1) = k^2 - 1$ vertices are online-colored, we can avoid a monochromatic $k$-wedge.

For $c \geq 4$ we can give an exponential (in $ck$) lower bound for the worst case:

**Theorem 3.** *For $c \geq 4$ we can online color with $c$ colors any set of $N = O(1.22074^{ck})$ points such that throughout the process there is no monochromatic wedge of size $k$. Moreover, if $c$ is large enough, then we can even online color $N = O(1.46557^{ck})$ points.*

*Proof.* Denote the colors by the numbers $\{1, \ldots, c\}$. A wedge refers to both an area in the plane as well as the collection of placed points which fall within that area. For brevity, we will often refer to maximal monochromatic wedges as simply wedges. If a wedge is not monochromatic, we will specifically note it. At each step, we define a partition of all the points which have come online in such a way that each set in the partition contains exactly one maximal monochromatic wedge. Two maximal monochromatic wedges are called *neighbors* if they are contained within a larger (non-monochromatic) wedge which contains no other monochromatic wedges. If the placement and coloring of a point cause a wedge to no longer be monochromatic, that wedge has been *killed*.

We now describe how to color a new point given that we have already colored some (or possibly no) points. If the new point is Northeast of an earlier point, it is given a different color from the earlier point. In this case no new wedges are created and no wedges increase in size. Otherwise the new point will eventually be part of a wedge. We want to make sure that the color of the point is distinct from its neighbors colors. In particular, consider the (at most) two wedges which are neighbors of wedges containing the point but which do not actually contain the point. From the $c$ colors we disregard these two colors. From the remaining, we choose the color which first minimizes the size of the wedge containing the point and secondly minimizes the color (as a number from 1 to $c$.) This means that our order of preference is first to have size 1 wedge of color 1, then a size 1 wedge of color 2, ..., size 1 wedge of color $c$, size 2 wedge of color 1, ... etc. These rules determine our algorithm. For an illustration see Figure 2.1 for a 5-coloring, where the new vertex $v$ cannot get the neighboring colors 2 and 4 and by our order of preference it gets color 3, thus introducing a (monochromatic) wedge (of color 3) of size 2. Now we have to see how effective this coloring algorithm is.

To prove the theorem we show that the partition set associated with the newly created (or incremented) wedge is relatively large. Suppose this wedge is of color $i$ and size $j$. Let $A_{i,j}$ denote the smallest possible size of the associated partition set. One can regard $A_{i,j}$ as the least number of points that are required to "build" a wedge $W$ of size $i$ of color $j$. For simplicity, we also use the notation $B_{c(i-1)+j} = A_{i,j}$. Note that this notation is well defined as $j$ is always less than $c$. Thus we have $B_1 = B_2 = B_3 = 1$ and $B_4 = 2$.

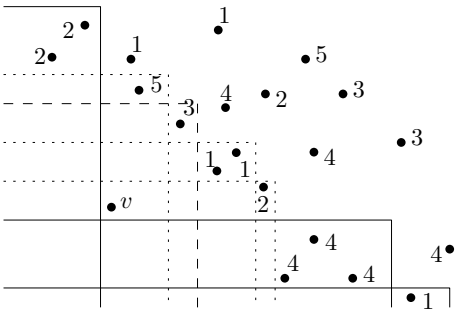



**Fig. 1.** A general step of the coloring in the proof of Theorem 3

It follows from our preferences that $B_i \leq B_j$ if $i \leq j$. Our goal is to give a good lower bound on $A_{k,1} = B_{c(k-1)+1}$.

Notice that when we create a new wedge, it will kill many points that were contained in previous wedges. More precisely, from our preferences we have $B_i \geq 1 + B_{i-3} + B_{i-4} + \ldots + B_{i-c}$ (where $B_i = 0$ for $i \leq 0$). Note that $B_{i-1}$ and $B_{i-2}$ is missing from this sum because the coloring method must choose a color different from the new points two to-be-neighbors'.

From the solution of this recursion we know that the magnitude of $B_i$ is at least $q^i$ where $q$ is the (unique, real, $> 1$) solution of $q^c = (q^{c-2}-1)/(q-1)$, which is equivalent to $q^{c+1} = q^c + q^{c-2} - 1$. Moreover, since trivially $B_i \geq 1 \geq q^{i-c}$ if $i \leq c$, from the recursion we also have $B_i \geq q^{i-c}$ for all $i$. If we suppose $c \geq 4$, then $q \geq 1.22074$ and from this $B_{c(k-1)+1} \geq 2.22074^{k-2}$. As $c$ tends to infinity, $q$ tends (from below) to the real root of $q^3 = q^2 + 1$, which is $\geq 1.46557$. From this we obtain that $B_{c(k-1)+1} \geq 1.46557^{c(k-2)}$ if $c$ is large. Also, in the special case $k = 2$, we get the well known sequence A000930 (see [20]), which is at least $1.46557^c$, if $c$ is big enough.

Summarizing, if we have $c \geq 4$ colors, the smallest $N_0$ number of points that forces a monochromatic wedge of size $k$ is exponential in $ck$. Thus, if the number of colors, $c$, is given, these bounds give an estimate of $\Theta(\log N/c)$ on the size of the biggest monochromatic wedge in the worst case.

If we consider $k$ fixed (and we want to use as few colors as possible), by the above bound the number of colors needed to avoid a monochromatic wedge of size $k$ is $\Theta(\log N/k)$ for $k \geq 1$.

**Corollary 2.** *There is a method to color online $N$ points in the plane using $\Theta(\log N/k)$ colors such that all monochromatic wedges have size strictly less than $k$.*

Recall that Theorem 2 stated that $N = 2^n - 1$ points can always force a size $n + 1$ color-vector (for the definition see the proof of Theorem 2). We remark that Theorem 3 implies a lower bound close to this bound too. Indeed, fix, e.g.,

$c = 4$ and $k = \lceil n/4 \rceil$. If the number of points is at most $N = O(1.22074^n) = O(1.22074^{ck})$ then by Theorem 3 there is an online coloring such that at any time there is no monochromatic wedge of size $k$, thus the color-vector is always at most $4(k-1) < n$.

Suppose now that $k$ is fixed and we want to use as few colors as possible without knowing in advance how many points will come, i.e. for $k$ fix we want to minimize $c$ without knowing $N$. To solve this, we alter our previous algorithm. (Note that we could also easily adjust the algorithm if for an unknown $N$ we want to minimize $\min(c, k)$, or $ck$, the answer would be still logarithmic in $N$.) All this comes with the price of loosing a bit on the base of the exponent. The following theorem implies that for $k = 2$ (and thus also for any $k \geq 2$) we can color any set of $N = O(1.0905^{ck})$ points and if $k$ is big enough then we can color any set of $N = O(1.1892^{ck})$ points.

**Theorem 4.** *For fixed $k \geq 1$ we can color a countable set of points such that for any $c$, and any $n < 2^{\lfloor (c+1)/4 \rfloor (k-1)}$, the first $n$ points of the set are $k$-properly $c$-colored.*

*Proof.* We need to define a coloring algorithm and prove that it uses many colors only if there were many points. The coloring and the proof is similar to the proof of Theorem 3, we only need to change our preferences when coloring and because of this the analysis of the performance of the algorithm differs slightly too. We fix a $c$ and an $N < 2^{\lfloor (c+1)/4 \rfloor (k-1)}$ for which we will prove the claim of the theorem (the coloring we define can obviously not depend on $c$ or $N$, but it depends on $k$). Denote the colors by the numbers $\{1, 2, \ldots, c-1, c, \ldots\}$.

We can suppose again that every new point will be on the actual diagonal. When we add a point its color must still be different from its to-be-neighbors' and together with this point we still cannot have a monochromatic wedge of size $k$. Our primary preference now is that we want to keep $\lfloor c/4 \rfloor$ small where $c$ is the color of the new point (as a number). Our secondary preference is that the size of the biggest wedge containing the new point should be small.

This means that our order of preference is first to have size 1 wedge of color 1, then a size 1 wedge of color 2, size 1 wedge of color 3, a size 1 wedge of color 4, a size 2 wedge of color 1, ..., a size $k-1$ wedge of color 1, size $k-1$ wedge of color 2, a size $k-1$ wedge of color 3, a size $k-1$ wedge of color 4, size 1 wedge of color 5, size 2 wedge of color 5, ... etc. These rules determine our algorithm, now we have to see how effective it is.

$A_{i,j}$ is defined as in the proof of Theorem 3. We only need to prove that $A_{i,j} \geq 2^{(k-1)(\lfloor j/4 \rfloor + i - 1)}$ as this means that if the algorithm uses the color $c+1$, then we had at least $A_{1,c+1} \geq 2^{(k-1)\lfloor (c+1)/4 \rfloor} > N$ points, a contradiction. Recall that $A_{i,j}$ denotes the least number of points that are required to "build" a wedge $W$ of size $i$ of color $j$.

We prove by induction. First, $A_{1,1} = A_{1,2} = A_{1,3} = A_{1,4} = 1$ indeed. By our preferences, whenever we introduce a size one wedge with color $j$, we had to kill at least two (four minus the two forbidden colors of the neighbors of the new point) points that have colors from the previous 4-tuple

of colors and are contained in monochromatic wedges of size $k-1$. Thus $A_{1,j} \geq A_{k-1,4(\lfloor j/(4-1)\rfloor+1)} + A_{k-1,4(\lfloor j/(4-1)\rfloor+2} \geq 2 \cdot 2^{(k-1)(\lfloor j/(4-1)\rfloor+k-2)} = 2 \cdot 2^{(k-1)(\lfloor j/4\rfloor-1} = 2^{(k-1)(\lfloor j/4\rfloor+1-1}$. If we introduce a wedge of size $i > 1$ with color $j$, we had to kill at least two points that have colors from the same 4-tuple of colors as $j$ and are contained in monochromatic wedges of size $i-1$. Thus in this case $A_{i,j} \geq A_{i-1,4(\lfloor j/4\rfloor+1} + A_{i-1,4(\lfloor j/4\rfloor+2} \geq 2 \cdot 2^{(k-1)(\lfloor j/4\rfloor+i-2)} = 2^{(k-1)(\lfloor j/4\rfloor+i-1}$.

**Proposition 2.** *The online coloring methods guaranteed by the second part of Proposition 1, Theorem 3 and Theorem 4 run in $O(n \log n)$ time to color the first $n$ points (even if we have a countable number of points and $n$ is not known in advance).*

The proof of this proposition is omitted as it follows easily from the analysis of the algorithms.

### 2.2 Online Coloring Intervals

This section deals with the following *interval coloring problem*. Given a finite family of intervals on the real line, we want to online color them with $c$ colors such that throughout the process if a point is covered by at least $k$ intervals, then not all of these intervals have the same color.

**Proposition 3.** *The interval coloring problem is equivalent to a restricted case of the point with respect to wedges coloring problem, where we care only about the wedges with apex on the line $L$ defined by $y = -x$.*

*Proof.* Consider the natural bijection of the real line and $L$. Associate to every point $p$ of $L$ the wedge with apex $p$ and associate with every interval $I = (x_1, -x_1), (x_2, -x_2)$ of $L$ the point $(x_1, -x_2)$. It is easy to see that $p \in I$ if and only if the point associated to $I$ is contained in the wedge associated to $p$.

**Corollary 3.** *Any upper bound on the number of colors necessary to (online) color wedges in the plane is also an upper bound for the number of colors necessary to (online) color intervals in $\mathbb{R}$.*

Also the lower bounds of Theorem 2 and of Proposition 1 follow for intervals easily by either repeating the proofs for intervals or by Observation 1:

**Observation 1.** *The proofs of Theorem 2 and of the first part of Proposition 1 can be easily modified such that all the relevant wedges have their apex on the line $y = -x$.*

In particular, we have the following.

**Corollary 4.** *There is a method to online color $N$ intervals in $\mathbb{R}$ using $\Theta(\log N/k)$ colors such that for every point $x$, contained in at least $k$ intervals, there exist two intervals containing $x$ of different colors.*

As we have seen the results about intervals follow in a straightforward way from the results about wedges. Thus all the statements we proved hold for online coloring wedges, also hold for intervals, however, it seems unlikely that the exact bounds are the same. Thus, we would be happy to see (small) examples where there is a distinction. As the next section shows, there is a difference between the exact bounds for quasi-online coloring wedges and intervals.

## 3 Quasi-online Coloring Intervals

A quasi-online coloring of an ordered collection of intervals $\{I_t\}_{t=1}^{n}$, is a coloring $\phi$ such that for every $k$, the sub-collection $\{I_t\}_{t=1}^{k}$ is properly colored under $\phi$. Proofs of the following Theorems are omitted from this conference version.

**Theorem 5.** *Any finite family of intervals on the line can be colored quasi-online with* 2 *colors such that at any time any point contained by at least* 3 *of these intervals is not monochromatic.*

**Theorem 6.** *Any finite family of intervals on the line can be colored quasi-online with* 3 *colors such that at any time any point contained by at least* 2 *of these intervals is not monochromatic.*

**Theorem 7.** *Colorings guaranteed by Theorem 5 and Theorem 6 can be found in* $O(n \log n)$ *time.*

These problems are equivalent to (offline) colorings of bottomless rectangles in the plane. Using this notation, Theorem 6 and Theorem 5 were proved already in [14] and [13], yet those proofs are quite involved and they only give quadratic time algorithms, so these results are improvements regarding simplicity of proofs and efficiency of the algorithms.

## References

1. Ajwani, D., Elbassioni, K., Govindarajan, S., Ray, S.: Conflict-Free Coloring for Rectangle Ranges Using $\tilde{O}(n^{.382+\epsilon})$ Colors. In: Proceedings of the 19th Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 181–187 (2007)
2. Bar-Noy, A., Cheilaris, P., Smorodinsky, S.: Deterministic conflict-free coloring for intervals: From offline to online. ACM Transactions on Algorithms 4(4) (2008)
3. Cardinal, J., Cohen, N., Collette, S., Hoffmann, M., Rote, G: Coloring Dynamic Point Sets on a Line. In: EuroCG (2012)
4. Chen, X., Pach, J., Szegedy, M., Tardos, G.: Delaunay graphs of point sets in the plane with respect to axis-parallel rectangles. In: Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, pp. 94–101 (2008)
5. Chen, K., Fiat, A., Levy, M., Matoušek, J., Mossel, E., Pach, J., Sharir, M., Smorodinsky, S., Wagner, U., Welzl, E.: Online conflict-free coloring for intervals, Siam. J. Comput. 36, 545–554 (2006)
6. Chrobak, M., Slusarek, M.: On some packing problems related to dynamic storage allocation. RAIRO Inform. Thor. Appl. 22(4), 487–499 (1988)

7. Even, G., Lotker, Z., Ron, D., Smorodinsky, S.: Conflict-free colorings of simple geometric regions with applications to frequency assignment in cellular networks. SIAM J. Comput. 33(1), 94–136 (2003)
8. Gyárfás, A., Lehel, J.: On-line and first fit colorings of graphs. Journal of Graph Theory 12(2), 217–227 (1988)
9. Halldórsson, M.M.: Online Coloring of Hypergraphs. Information Processing Letters 110, 370–372 (2010)
10. Nagy-György, J.: Cs. Imreh, Online hypergraph coloring. Information Processing Letters 109(1), 23–26 (2008)
11. Keszegh, B., Pálvölgyi, D.: Octants are Cover Decomposable. Discrete and Computational Geometry (2011)
12. Keszegh, B., Pálvölgyi, D.: Octants are Cover Decomposable Into Many Coverings (manuscript)
13. Keszegh, B.: Coloring half-planes and bottomless rectangles. Computational Geometry: Theory and Applications (2012), http://dx.doi.org/10.1016/j.comgeo.2011.09.004
14. Keszegh, B.: Weak conflict free colorings of point sets and simple regions. In: Proceedings of the 19th Canadian Conference on Computational Geometry, pp. 97–100 (2007)
15. Kierstead, H.A.: On-line coloring k-colorable graphs. Israel Journal of Mathematics 105(1), 93–104 (1996)
16. Kierstead, H.A.: The linearity of first-fit coloring of interval graphs. SIAM J. Discrete Math. 1(4), 526–530 (1988)
17. Kierstead, H.A., Smith, D., Trotter, W.T.: First-fit coloring of interval graphs. In: SIAM Conference on Discrete Mathematics, Combinatorics and Partially Ordered Sets (2010)
18. Lovász, L., Saks, M., Trotter, W.T.: An on-line graph coloring algorithm with sublinear performance ratio. Discrete Mathematics 75(13), 319–325 (1989)
19. Narayanaswamy, N.S., Babu, R.S.: A Note on First-Fit Coloring of Interval Graphs. Order 25(1), 49–53 (2008)
20. https://oeis.org/A000930
21. Pach, J., Tóth, G.: Conflict free colorings. In: Basu, S., et al. (eds.) Discrete and Computational Geometry - The Goodman-Pollack Festschrift, pp. 665–671. Springer, Berlin (2003)
22. Pálvölgyi, D.: Decomposition of Geometric Set Systems and Graphs, PhD thesis, arXiv:1009.4641[math.co]
23. Smorodinsky, S.: Conflict-Free Coloring and its Applications, arXiv:1005.3616v3 [math.co]
24. Smorodinsky, S.: On The Chromatic Number of Some Geometric Hypergraphs. SIAM Journal on Discrete Mathematics 21(3), 676–687 (2007)
25. Trotter, W.T.: New perspectives on interval orders and interval graphs. In: Bailey, R.A. (ed.) Surveys in Combinatorics. London Math. Soc. Lecture Note Ser, vol. 241, pp. 237–286 (1997)
26. Tardos, G.: Personal Communication
27. Tsaia, Y.-T., Lina, Y.-L., Hsu, F.R.: The on-line first-fit algorithm for radio frequency assignment problems. Information Processing Letters 84(4), 195–199 (2002)
28. Woodall, D.R.: Problem No. 4, Combinatorics. In: McDonough, T.P., Marvon, V.C. (eds.) Proc. British Combin. Conf. 1973, Cambridge. London Math. Soc. Lecture Note Series, vol. 13, p. 202 (1974)

## A Proofs of Theorems 5-7

We exploit an idea used in [11]; instead of online coloring the intervals we online build a labelled acyclic graph (i.e., a forest) with the following properties. Each interval will correspond to a vertex in this graph (there might be other vertices in the graph as well). The final coloring of the intervals will then be generated from this graph. In particular, to define a two-coloring, we will assign each edge in the forest one of two labels, "different" or "same". For an arbitrary coloring of exactly one vertex in each component (tree) of the graph, there is a unique extension to a coloring of the whole graph compatible with the labelling, i.e., such that each edge labelled "same" is adjacent to vertices of the same color and each edge labelled "different" is adjacent to vertices of different colors. In [11] all the edges were labelled "different" so it was actually a simpler variant of our current scheme. As we will see, this idea can also be generalized to more than two colors.

We denote the color of an interval $I$ by $\phi(I)$, the left (resp. right) endvertex of $I$ by $l(I)$ (resp. by $r(I)$). These vertices are real numbers, and so they can be compared.

*Proof (of Theorem 5).* Let $\{I_t\}_{t=1}^n$ be the given enumeration of the intervals to be quasi-online colored. We first build the forest and then show that the coloring defined by this forest works. As we build the forest we will maintain also a set of intervals, called the active intervals (not necessarily a subset of the given set of intervals). At any time $t$ the vertices of the actual forest correspond to the intervals of $\{I_t\}_{t=1}^n$ and the set of current or past active intervals. The set of active intervals will change during the process, but we maintain that the following properties hold any time.

1. Every point of the line is covered by at most two active intervals.
2. No active interval contains another active interval.
3. A point is either forced by the labelling to be contained in original intervals of different colors or it is contained in the same number of active intervals as original intervals, and additionally the labelling forces these original intervals to have the same colors as these active intervals.
4. Each tree in the forest contains exactly one vertex that corresponds to an active interval.

The last property ensures that a coloring of the active intervals determines a unique coloring of all the intervals which is compatible with the labelling of the forest.

Note that in the third property one or two of the original intervals can actually coincide with one or two of the active intervals.

For the first step we simply make the first interval active; our forest will consist of a single vertex corresponding to this interval. In general, at the beginning of step $t$, we have a list of active intervals, $\mathcal{J}_{t-1}$. Consider the $t^{\text{th}}$ interval, $I_t$. If $I_t$ is covered by an active interval, $J \in \mathcal{J}_{t-1}$, then we add $I_t$ to the forest and connect it to $J$ with an edge labelled "different". Note that there is at most one

such active interval. If there is no active interval containing $I_t$, we add $I_t$ to the set of active intervals and also add a corresponding vertex to the forest. Now if there are active intervals contained in $I_t$, these are all deactivated (removed from the set of active intervals) and each is connected to $I_t$ in the graph with an edge labelled "different". Note that this way any point covered by these inactivated intervals will be covered by intervals of both colors.

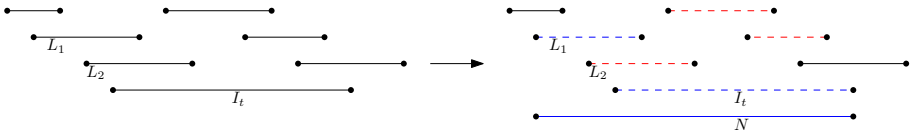



**Fig. 2.** A general step in the proof of Theorem 5

It remains to ensure that no point is contained within three active intervals. If there still do exist such points, by induction they must be contained within $I_t$. Let $L_1$ and $L_2$ be the (at most) two active intervals covering $l(I_t)$ such that $l(L_1) < l(L_2)$ (if both of them exist.) Similarly, let $R_1$ and $R_2$ be the (at most) two active intervals covering $r(I_t)$ such that $l(R_1) < l(R_2)$ (if both of them exist.) We note that the $L_i$ and $R_j$ cannot be the same, as such an interval would cover $I_t$. Also, no other active intervals can intersect $I_t$, as they would necessarily be contained in $I_t$. Without loss of generality we can assume that both $L_1$ and $L_2$ exist. If $R_1$ and $R_2$ also both exist, deactivate $L_1, L_2, I_t, R_1$ and $R_2$ and activate a new interval $N = L_1 \cup I_t \cup R_2$ (and add a corresponding vertex to the graph). In the graph, connect $L_1, I_t$ and $R_2$ to $N$ with edges labelled "same". Connect $L_2$ and $R_1$ to $N$ with edges labelled "different". Otherwise, if at most one active edge contains $r(I_t)$ we deactive $L_1$ and $L_2$ and connect these to the new interval $N = L_1 \cup I_t$ (again with edges labelled "same" and "different", respectively), also we deactive $I_t$ and connect it to $N$ with an edge labelled "same". Figure is an illustration of this case when the active interval $N$ is assigned color blue and deactivated intervals are shown with dashed lines.

This way within a given step, any point which is contained in (at least) two intervals deactivated during the step, is forced by the labelling to be contained in intervals of different colors. For any other point $v$ the number of original intervals containing $v$ remains the same as the number of active intervals covering $v$ (both remains the same or both increases by 1). The first three properties were maintained and also it is easy to check that the graph remains a forest such that in each component there is a unique active interval.

At the end of the process any coloring of the final set of active intervals extends to a coloring of all the intervals (compatible with the labelling of the graph). We have to prove that for this coloring at any time any point contained by at least 3 of these intervals is not monochromatic. By induction at any time $t < n$ the coloring is compatible with the graph at that time, thus by induction any point contained by at least 3 of these intervals is not monochromatic. Now at

time $n$, if the active intervals are colored, the extension (by induction) is such that every point not in $I_n$ is either covered by at most two original intervals or it is covered by intervals of both colors. On the other hand, from the way we defined the graph, we can see that points covered by $I_n$ and contained in at least 3 intervals are covered by intervals of both colors as well. Indeed, by the properties maintained, if a point $v$ is not covered by intervals of both colors than it is covered by as many active intervals as original intervals. Yet, no point is covered by more than 2 active intervals at any time, thus $v$ is covered by no more than 2 active and thus no more than 2 original intervals.

*Proof (of Theorem 6)*
We proceed similarly to Theorem 5. In particular, we require the same four properties from active intervals, although the second two need some modifications. Now instead of a labelled graph we define rules of the following form: some interval $I$ (original or auxiliary) gets a different color from at most two other intervals $J_1, J_2$. We say that $I$ depends from $J_1, J_2$, otherwise $I$ is independent. If there is an order on the intervals such that an interval depends only on intervals later in this order then starting with any coloring of the independent intervals and then coloring the dependent ones from the last going backwards we can naturally extend this coloring to all the intervals such that the coloring is compatible with the rules (i.e. $I$ gets a color different from the color of $J_1, J_2$ for all dependent triples). For a representation with directed acyclic graphs - showing more clearly the similarities with the previous proof - see the proof of Theorem 7.

1. Every point of the line is covered by at most two active intervals.
2. No active interval contains another active interval.
3. A point of the line is either forced by the rules to be contained in original intervals of different colors or it is contained in the same number of active intervals as original intervals, and additionally the rules force these original intervals to have the same colors as these active intervals.
4. An interval is independent if and only if it is an active interval.

The first two properties ensure the following structure on the set of active intervals. Define a chain as a sequence of active intervals such that everyone intersects the one before and after it in the chain. The set of active intervals can be partitioned into disjoint chains. The last property guarantees that any coloring of the active intervals extends naturally and uniquely to a coloring of all the intervals which is compatible with the rules.

We will define the rules such that if we start by a proper coloring of the active intervals then the extension is a quasi-online coloring (as required by the theorem) of the original set of intervals. Note that in the previous proof we started with an arbitrary coloring of the active intervals, which was not necessarily proper, thus now we additionally have to take care that a proper coloring of the active intervals extends to a coloring which is a proper coloring of the active intervals at any previous time as well.

In the first step we add $I_1$ and activate it. In the induction step we add $I_t$ to the set of active intervals. If $I_t$ is covered by an active interval or by two

intervals of a chain, then we deactivate $I_t$ and the rule is that we give a color to it differing from the color(s) of the interval(s). If $I_t$ does not create a triple intersection, it remains activated. Otherwise, denote by $L$ (resp. $R$) the interval with the leftmost left end (resp. rightmost right end) that covers a triple covered point. We distinguish two cases.

Case i) If $I_t$ is not covered by one chain, then either $L$ or $R$ is $I_t$, or $L$ and $R$ are not in the same chain. In either case we deactivate all intervals covered by $N = L \cup I_t \cup R$, except for $L$, $I_t$ and $R$. The rule to color the now deactivated intervals is that they get a color different from $I_t$, in an alternating way along their chains starting from $L$ and $R$.

It is easy to check that the four properties are maintained.

Given a proper coloring of the active intervals at step $n$ by our rules it extends to a proper coloring of the active intervals in the previous step. Thus by induction at any time $t < n$ for any point $v$ it is either covered by differently colored intervals or it is covered by at most one interval. For time $n$ it is either covered by differently colored intervals or it is covered by as many original intervals as active intervals, and they have the same set of colors (by the third property). As the coloring was proper on the active intervals, $v$ is either covered by two original intervals and then two active intervals which have different colors, thus the original intervals have different colors as well, or $v$ is covered by at most one active and thus by at most one original interval.

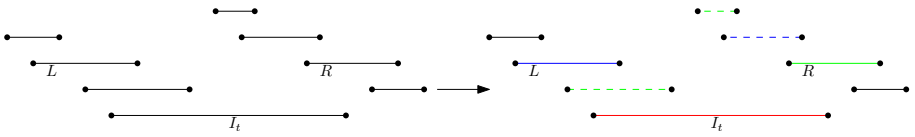



**Fig. 3.** Case i) of Theorem 6

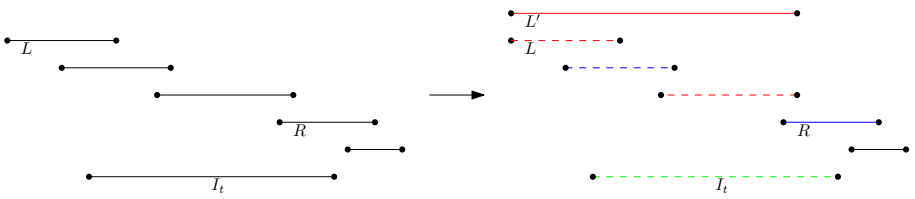



**Fig. 4.** Case ii), odd subcase of Theorem 6

Case ii) If $I_t$ is covered by one chain, then $L$ and $R$ both differ from $I_t$. We deactivate all intervals covered by $L \cup I_t \cup R$ (including $I_t$), except for $L$ and $R$. Notice that apart from $I_t$ these intervals are all between $L$ and $R$ in this chain.

If we deactivated an odd number of intervals this way (so an even number from the chain), then we insert the new active interval $L'$ that we get from $L$ by prolonging the right end of $L$ such that $L'$ and $R$ intersect in an epsilon short

interval. We deactivate $L$ and the rule is to color it the same as we color $L'$. The rule to color the deactivated $I_t$ is to color it differently from the color of $L'$ (or, equivalently, $L$) and $R$. The rule to color the deactivated intervals of the chain is to color them in an alternating way using the colors of $L$ and $R$ (in a final proper coloring of the active intervals they get different colors as $L'$ and $R$ intersect). If we deactivated an even number of intervals this way (so an odd number from the chain), then we deactivate $L$ and $R$ as well and add a new active interval $N = L \cup I_t \cup R$. The rule to color the deactivated $I_t$ is to color it differently from the color of $N$. The rule to color the deactivated intervals of the chain is to color them in an alternating way using the color of $N$ ($L$ and $R$ get this color) and the color that is different from the color of $N$ and $I_t$.

It is easy to check that the four properties are maintained. Also, similarly to the previous case, it can be easily checked that if we extend a proper coloring of the active intervals then for its extension it is true at any time (for time $t < n$ by induction, otherwise by the way we defined the rules) that every point is either covered by at most one original interval or it is covered by intervals of different colors.

*Proof (of Theorem 7).* Instead of a rigorous proof we provide only a sketch, the easy details are left to the reader. In both algorithms we have $n$ intervals, thus $n$ steps. In each step we define a bounded number of new active intervals, thus altogether we have $cn$ regular and active intervals. We always maintain the (well-defined) left-to-right order of the active intervals. Also we maintain an order of the (active and regular) intervals such that an interval's color depends only on the color of one or two intervals' that are later in this order. This order can be easily maintained as in each step the new interval and the new active intervals come at the end of the order. We also save for each interval the one or two intervals which it depends on. This can be imagined as the intervals represented by vertices on the horizontal line arranged according to this order and an acyclic directed graph on them representing the dependency relations, thus each edge goes backwards and each vertex has indegree at most two (at most one in the first algorithm, i.e. the graph is a directed forest in that case). In each step we have to update the order of active intervals and the acyclic graph of all the intervals, this can be done in $c \log n$ time plus the time needed for the deletion of intervals from the order. Although the latter can be linear in a step, yet altogether during the whole process it remains $cn$, which is still ok. At the end we just color the vertices one by one from right to left following the rules, which again takes only $cn$ time. Altogether this is $cn \log n$ time.

We note that the algorithms in [14] and [13] proceed with the intervals in backwards order and the intervals are colored immediately (in each step many of them are also recolored), this might be a reason why a lot of recolorings are needed there (which we don't need in the above proofs), adding up to quadratic time algorithms (contrasting the near-linear time algorithms above).

# Logic Characterization of Invisibly Structured Languages: The Case of Floyd Languages

Violetta Lonati[1], Dino Mandrioli[2], and Matteo Pradella[2]

[1] DI - Università degli Studi di Milano, via Comelico 39/41, Milano, Italy
lonati@di.unimi.it
[2] DEI - Politecnico di Milano, via Ponzio 34/5, Milano, Italy
{dino.mandrioli,matteo.pradella}@polimi.it

**Abstract.** Operator precedence grammars define a classical Boolean and deterministic context-free language family (called Floyd languages or FLs). FLs have been shown to strictly include the well-known Visibly Pushdown Languages, and enjoy the same nice closure properties. In this paper we provide a complete characterization of FLs in terms of a suitable Monadic Second-Order Logic. Traditional approaches to logic characterization of formal languages refer explicitly to the structures over which they are interpreted - e.g, trees or graphs - or to strings that are isomorphic to the structure, as in parenthesis languages. In the case of FLs, instead, the syntactic structure of input strings is "invisible" and must be reconstructed through parsing. This requires that logic formulae encode some typical context-free parsing actions, such as shift-reduce ones.

**Keywords:** operator precedence languages, deterministic context-free languages, monadic second-order logic, pushdown automata.

## 1 Introduction

Floyd languages (FL), as we renamed Operator Precedence Languages and grammars (FG) after their inventor, were originally introduced to support deterministic parsing of programming and other artificial languages [7]; then, interest in them decayed for several decades, probably due to the advent of more expressive grammars, such as LR ones [8] which also allow for efficient deterministic parsing.

In another context Visual Pushdown Languages (VPL) -and other connected families e.g. [3]- have been introduced and investigated [1] with the main motivation to extend to them the same or similar automatic analysis techniques -noticeably, model checking- that have been so successful for regular languages. Recently we discovered that VPL are a proper subclass of FL, which in turn enjoy the same properties that make regular and VP languages amenable to extend to them typical model checking techniques; in fact, to the best of our knowledge, FL are the largest family closed w.r.t. Boolean operation, concatenation, Kleene * and other classical operations [5]. Another relevant feature of FL is their "locality property", i.e., the fact that partial strings can be parsed independently of the context in which they occur within a whole string. This enables more effective parallel and incremental parsing techniques than for other deterministic languages [2].

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 307–318, 2013.

We also introduced an appropriate automata family that matches FGs in terms of generative power: Floyd Automata (FA) are reported in [10] and, with more details and precision, in [9]. In this paper we provide the "last tile of the puzzle", i.e., a complete characterization of FL in terms of a suitable Monadic Second-Order (MSO) logic, so that, as well as with regular languages, one can, for instance, state a language property by means of an MSO formula; then automatically verify whether a given FA accepts a language that enjoys that property.

In the literature various other classes of languages (including VPL) and structures (trees and graphs) have been characterized by means of MSO logic [4] by extending the original approach of Büchi (presented e.g. in [12]). To the best of our knowledge, however, all these approaches refer to a tree or graph structure which is explicitly available. In the case of FLs, instead, the syntax tree is not immediately visible in the string, hence a parsing phase is needed. In fact, Floyd automata are the only non-real-time automata we are aware of, characterized in terms of MSO logic.

The paper is structured as follows: Section 2 provides the necessary background about FL and their automata. Section 3 defines an MSO over strings and provides two symmetric constructions to derive an equivalent FA from an MSO formula and conversely. Section 4 offers some conclusion and hints for future work.

## 2 Preliminaries

FL are normally defined through their generating grammars [7]; in this paper, however, we characterize them through their accepting automata [9,10] which are the natural way to state equivalence properties with logic characterization. Nevertheless we assume some familiarity with classical language theory concepts such as context-free grammar, parsing, shift-reduce algorithm, syntax tree [8].

Let $\Sigma = \{a_1, \ldots, a_n\}$ be an alphabet. The empty string is denoted $\epsilon$. We use a special symbol # not in $\Sigma$ to mark the beginning and the end of any string. This is consistent with the typical operator parsing technique that requires the look-back and look-ahead of one character to determine the next parsing action [8].

**Definition 1.** *An operator precedence matrix (OPM) $M$ over an alphabet $\Sigma$ is a partial function $(\Sigma \cup \{\#\})^2 \to \{\lessdot, \doteq, \gtrdot\}$, that with each ordered pair $(a, b)$ associates the OP relation $M_{a,b}$ holding between $a$ and $b$. We call the pair $(\Sigma, M)$ an* operator precedence alphabet *(OP). Relations $\lessdot, \doteq, \gtrdot$, are named yields precedence, equal in precedence, takes precedence, respectively. By convention, the initial # can only yield precedence, and other symbols can only take precedence on the ending #.*

If $M_{a,b} = \circ$, where $\circ \in \{\lessdot, \doteq, \gtrdot\}$, we write $a \circ b$. For $u, v \in \Sigma^*$ we write $u \circ v$ if $u = xa$ and $v = by$ with $a \circ b$. $M$ is *complete* if $M_{a,b}$ is defined for every $a$ and $b$ in $\Sigma$. Moreover in the following we assume that $M$ is *acyclic*, which means that $c_1 \doteq c_2 \doteq \ldots \doteq c_k \doteq c_1$ does not hold for any $c_1, c_2, \ldots c_k \in \Sigma, k \geq 1$. See [6,5,9] for a discussion on this hypothesis.

**Definition 2.** *A nondeterministic* Floyd automaton *(FA) is a tuple $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ where: $(\Sigma, M)$ is a precedence alphabet; $Q$ is a set of states (disjoint from $\Sigma$); $I, F \subseteq Q$ are sets of initial and final states, respectively; $\delta : Q \times (\Sigma \cup Q) \to 2^Q$ is the transition function.*

The transition function is the union of two disjoint functions: $\delta_{\text{push}} : Q \times \Sigma \to 2^Q$, and $\delta_{\text{flush}} : Q \times Q \to 2^Q$. A nondeterministic FA can be represented by a graph with $Q$ as the set of vertices and $\Sigma \cup Q$ as the set of edge labelings: there is an edge from state $q$ to state $p$ labelled by $a \in \Sigma$ if and only if $p \in \delta_{push}(q, a)$ and there is an edge from state $q$ to state $p$ labelled by $r \in Q$ if and only if $p \in \delta_{flush}(q, r)$. To distinguish flush transitions from push transitions we denote the former ones by a double arrow.

To define the semantics of the automaton, we introduce some notations. We use letters $p, q, p_i, q_i, \ldots$ for states in $Q$ and we set $\Sigma' = \{a' \mid a \in \Sigma\}$; symbols in $\Sigma'$ are called *marked* symbols. Let $\Gamma = (\Sigma \cup \Sigma' \cup \{\#\}) \times Q$; we denote symbols in $\Gamma$ as $[a\ q]$, $[a'q]$, or $[\#\ q]$, respectively. We set $smb([a\ q]) = smb([a'q]) = a$, $smb([\#\ q]) = \#$, and $st([a\ q]) = st([a'q]) = st([\#\ q]) = q$.

A *configuration* of a FA is any pair $C = \langle B_1 B_2 \ldots B_n,\ a_1 a_2 \ldots a_m \rangle$, where $B_i \in \Gamma$ and $a_i \in \Sigma \cup \{\#\}$. The first component represents the contents of the stack, while the second component is the part of input still to be read.

A computation is a finite sequence of moves $C \vdash C_1$; there are three kinds of moves, depending on the precedence relation between $smb(B_n)$ and $a_1$:

**(push)** if $smb(B_n) \doteq a_1$ then $C_1 = \langle B_1 \ldots B_n[a_1\ q],\ a_2 \ldots a_m \rangle$, with $q \in \delta_{push}(st(B_n), a_1)$;
**(mark)** if $smb(B_n) \lessdot a_1$ then $C_1 = \langle B_1 \ldots B_n[a_1{}'q],\ a_2 \ldots a_m \rangle$, with $q \in \delta_{push}(st(B_n), a_1)$;
**(flush)** if $smb(B_n) \gtrdot a_1$ then let $i$ the greatest index such that $smb(B_i) \in \Sigma'$.
$C_1 = \langle B_1 \ldots B_{i-2}[smb(B_{i-1})\ q],\ a_1 a_2 \ldots a_m \rangle$, with $q \in \delta_{flush}(st(B_n), st(B_{i-1}))$.

Finally, we say that a configuration $[\#\ q_I]$ is *starting* if $q_I \in I$ and a configuration $[\#\ q_F]$ is *accepting* if $q_F \in F$. The language accepted by the automaton is defined as:

$$L(\mathcal{A}) = \left\{ x \mid \langle [\#\ q_I],\ x\# \rangle \overset{*}{\vdash} \langle [\#\ q_F],\ \# \rangle, q_I \in I, q_F \in F \right\}.$$

Notice that transition function $\delta_{\text{push}}$ is used to perform both push and mark moves. To distinguish them, we need only to remember the last symbol read (i.e., the *look-back*), encoding such an information into the states. Hence, in the graphical representation of a FA we will use a bold arrow to denote mark moves in the state diagram.

The deterministic version of FA is defined along the usual lines.

**Definition 3.** *A FA is deterministic if I is a singleton, and the ranges of $\delta_{push}$ and $\delta_{flush}$ are both Q rather than $2^Q$.*

In [9] we proved in a constructive way that nondeterministic FA have the same expressive power as the deterministic ones and both are equivalent to the original Floyd grammars.

*Example 1.* We define here the stack management of a simple programming language that is able to handle nested exceptions. For simplicity, there are only two procedures, called $a$ and $b$. Calls and returns are denoted by $call_a$, $call_b$, $ret_a$, $ret_b$, respectively. During execution, it is possible to install an exception handler *hnd*. The last signal that we use is *rst*, that is issued when an exception occurs, or after a correct execution to uninstall the handler. With a *rst* the stack is "flushed", restoring the state right before the last *hnd*. Every *hnd* not installed during the execution of a procedure is managed by the OS. We require also that procedures are called in an environment controlled by the

OS, hence calls must always be performed between a *hnd*/*rst* pair (in other words, we do not accept *top-level* calls). The automaton modeling the above behavior is presented in Figure 1. Note that every arrow labeled with *hnd* is bold as it represents a mark transition. An example run and the corresponding tree are presented in Figure 2.

| | $call_a$ | $ret_a$ | $call_b$ | $ret_b$ | *hnd* | *rst* | # |
|---|---|---|---|---|---|---|---|
| $call_a$ | $\lessdot$ | $\doteq$ | $\lessdot$ | | $\lessdot$ | $\gtrdot$ | |
| $ret_a$ | $\lessdot$ | $\gtrdot$ | $\lessdot$ | $\gtrdot$ | | $\gtrdot$ | $\gtrdot$ |
| $call_b$ | $\lessdot$ | | $\lessdot$ | $\doteq$ | $\lessdot$ | $\gtrdot$ | |
| $ret_b$ | $\lessdot$ | $\gtrdot$ | $\lessdot$ | $\gtrdot$ | | $\gtrdot$ | $\gtrdot$ |
| *hnd* | $\lessdot$ | $\lessdot$ | $\lessdot$ | $\lessdot$ | | $\doteq$ | |
| *rst* | $\gtrdot$ | $\gtrdot$ | $\gtrdot$ | $\gtrdot$ | $\gtrdot$ | $\gtrdot$ | $\gtrdot$ |
| # | $\lessdot$ | | $\lessdot$ | | $\lessdot$ | | $\doteq$ |

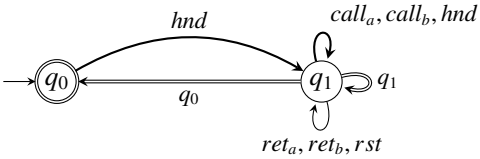



**Fig. 1.** Precedence matrix and automaton of Example 1

Such a language is not a VPL but somewhat extends their rationale: in fact, whereas VPL allow for unmatched parentheses only at the beginning of a sentence (for returns) or at the end (for calls), in this language we can have unmatched $call_a$, $call_b$, $ret_a$, $ret_b$ within a pair *hnd*, *rst*.

**Definition 4.** *A* simple chain *is a string* $c_0c_1c_2\dots c_\ell c_{\ell+1}$*, written as* ${}^{c_0}[c_1c_2\dots c_\ell]^{c_{\ell+1}}$*, such that:* $c_0, c_{\ell+1} \in \Sigma \cup \{\#\}$*,* $c_i \in \Sigma$ *for every* $i = 1, 2, \dots \ell$*, and* $c_0 \lessdot c_1 \doteq c_2 \dots c_{\ell-1} \doteq c_\ell \gtrdot c_{\ell+1}$*.*

*A* composed chain *is a string* $c_0s_0c_1s_1c_2\dots c_\ell s_\ell c_{\ell+1}$*, where* ${}^{c_0}[c_1c_2\dots c_\ell]^{c_{\ell+1}}$ *is a simple chain, and* $s_i \in \Sigma^*$ *is the empty string or is such that* ${}^{c_i}[s_i]^{c_{i+1}}$ *is a chain (simple or composed), for every* $i = 0, 1, \dots, \ell$*. Such a composed chain will be written as* ${}^{c_0}[s_0c_1s_1c_2\dots c_\ell s_\ell]^{c_{\ell+1}}$*.*

*A string* $s \in \Sigma^*$ *is* compatible *with the OPM* $M$ *if* ${}^{\#}[s]^{\#}$ *is a chain.*

**Definition 5.** *Let* $\mathcal{A}$ *be a Floyd automaton. We call a* support *for the simple chain* ${}^{c_0}[c_1c_2\dots c_\ell]^{c_{\ell+1}}$ *any path in* $\mathcal{A}$ *of the form*

$$q_0 \xrightarrow{c_1} q_1 \longrightarrow \dots \longrightarrow q_{\ell-1} \xrightarrow{c_\ell} q_\ell \overset{q_0}{\Longrightarrow} q_{\ell+1} \tag{1}$$

*Notice that the label of the last (and only) flush is exactly* $q_0$*, i.e. the first state of the path; this flush is executed because of relation* $c_\ell \gtrdot c_{\ell+1}$*.*

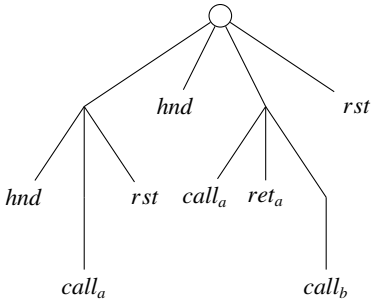
*We call a* support for the composed chain ${}^{c_0}[s_0c_1s_1c_2\dots c_\ell s_\ell]^{c_{\ell+1}}$ *any path in* $\mathcal{A}$ *of the form*

$$q_0 \overset{s_0}{\rightsquigarrow} q'_0 \xrightarrow{c_1} q_1 \overset{s_1}{\rightsquigarrow} q'_1 \xrightarrow{c_2} \dots \xrightarrow{c_\ell} q_\ell \overset{s_\ell}{\rightsquigarrow} q'_\ell \overset{q'_0}{\Longrightarrow} q_{\ell+1} \tag{2}$$

*where, for every* $i = 0, 1, \dots, \ell$*:*

- *if* $s_i \neq \epsilon$*, then* $q_i \overset{s_i}{\rightsquigarrow} q'_i$ *is a support for the chain* ${}^{c_i}[s_i]^{c_{i+1}}$*, i.e., it can be decomposed as* $q_i \overset{s_i}{\rightsquigarrow} q''_i \overset{q_i}{\Longrightarrow} q'_i$*.*
- *if* $s_i = \epsilon$*, then* $q'_i = q_i$*.*

*Notice that the label of the last flush is exactly* $q'_0$*.*




$$
\begin{array}{lll}
 & \langle [\#\ q_0] & , \ hnd\ call_a\ rst\ hnd\ call_a\ ret_a\ call_b\ rst\#\rangle \\
\text{mark} & \langle [\#\ q_0][hnd'\ q_1] & , \ call_a\ rst\ hnd\ call_a\ ret_a\ call_b\ rst\#\rangle \\
\text{mark} & \langle [\#\ q_0][hnd'\ q_1][call'_a\ q_1] & , \ rst\ hnd\ call_a\ ret_a\ call_b\ rst\#\rangle \\
\text{flush} & \langle [\#\ q_0][hnd'\ q_1] & , \ rst\ hnd\ call_a\ ret_a\ call_b\ rst\#\rangle \\
\text{push} & \langle [\#\ q_0][hnd'\ q_1][rst\ q_1] & , \ hnd\ call_a\ ret_a\ call_b\ rst\#\rangle \\
\text{flush} & \langle [\#\ q_0] & , \ hnd\ call_a\ ret_a\ call_b\ rst\#\rangle \\
\text{mark} & \langle [\#\ q_0][hnd'\ q_1] & , \ call_a\ ret_a\ call_b\ rst\#\rangle \\
\text{mark} & \langle [\#\ q_0][hnd'\ q_1][call'_a\ q_1] & , \ ret_a\ call_b\ rst\#\rangle \\
\text{push} & \langle [\#\ q_0][hnd'\ q_1][call'_a\ q_1][ret_a\ q_1] & , \ call_b\ rst\#\rangle \\
\text{mark} & \langle [\#\ q_0][hnd'\ q_1][call'_a\ q_1][ret_a\ q_1][call'_b\ q_1] & , \ rst\#\rangle \\
\text{flush} & \langle [\#\ q_0][hnd'\ q_1][call'_a\ q_1][ret_a\ q_1] & , \ rst\#\rangle \\
\text{flush} & \langle [\#\ q_0][hnd'\ q_1] & , \ rst\#\rangle \\
\text{push} & \langle [\#\ q_0][hnd'\ q_1][rst\ q_1] & , \ \#\rangle \\
\text{flush} & \langle [\#\ q_0] & , \ \#\rangle
\end{array}
$$

**Fig. 2.** Example run and corresponding tree of the automaton of Example 1

The chains fully determine the structure of the parsing of any automaton over $(\Sigma, M)$. Indeed, if the automaton performs the computation $\langle [a\ q_0],\ sb\rangle \overset{*}{\vdash} \langle [a\ q],\ b\rangle$, then ${}^a[s]^b$ is necessarily a chain over $(\Sigma, M)$ and there exists a support like (2) with $s = s_0 c_1 \dots c_\ell s_\ell$ and $q_{\ell+1} = q$.

Furthermore, the above computation corresponds to the parsing by the automaton of the string $s_0 c_1 \dots c_\ell s_\ell$ within the context $a$,$b$. Notice that such context contains all information needed to build the subtree whose frontier is that string. This is a distinguishing feature of FL, not shared by other deterministic languages: we call it the *locality principle* of Floyd languages.

*Example 2.* With reference to the tree in Figure 1, the parsing of substring $hnd\ call_a\ rst\ hnd$ is given by computation $\langle [\#\ q_0]\ ,\ hnd\ call_a\ rst\ hnd\rangle \overset{*}{\vdash} \langle [\#\ q_0]\ ,\ hnd\rangle$ which corresponds to support $q_0 \overset{hnd}{\longrightarrow} q_1 \overset{call_a}{\longrightarrow} q_1 \overset{q_1}{\Longrightarrow} q_1 \overset{rst}{\longrightarrow} q_1 \overset{q_0}{\Longrightarrow} q_0$ of the composed chain ${}^{\#}[hnd\ call_a\ rst]^{hnd}$.

**Definition 6.** *Given the OP alphabet* $(\Sigma, M)$*, let us consider the FA* $\mathcal{A}(\Sigma, M) = \langle \Sigma, M, \{q\}, \{q\}, \{q\}, \delta_{max} \rangle$ *where* $\delta_{max}(q, q) = q$*, and* $\delta_{max}(q, c) = q$*,* $\forall c \in \Sigma$*. We call* $\mathcal{A}(\Sigma, M)$ *the* Floyd Max-Automaton *over* $\Sigma, M$*.*

For a max-automaton $\mathcal{A}(\Sigma, M)$ each chain has a support; since there is a chain $^{\#}[s]^{\#}$ for any string $s$ compatible with $M$, a string is accepted by $\mathcal{A}(\Sigma, M)$ iff it is compatible with $M$. Also, whenever $M$ is complete, each string is compatible with $M$, hence accepted by the max-automaton. It is not difficult to verify that a max-automaton is equivalent to a max-grammar as defined in [6]; thus, when M is complete both the max-automaton and the max-grammar define the universal language $\Sigma^*$ by assigning to any string the (unique) structure compatible with the OPM.

In conclusion, given an OP alphabet, the OPM $M$ assigns a structure to any string in $\Sigma^*$; unlike parentheses languages such a structure is not visible in the string, and must be built by means of a non-trivial parsing algorithm. A FA defined on the OP alphabet selects an appropriate subset within the "universe" of strings compatible with $M$. In some sense this property is yet another variation of the fundamental Chomsky-Shützenberger theorem.

## 3 Logic Characterization of FL

We are now ready to provide a characterization of FL in terms of a suitable Monadic Second Order (MSO) logic in the same vein as originally proposed by Büchi for regular languages and subsequently extended by Alur and Madhusudan for VPL. The essence of the approach consists in defining language properties in terms of relations between the positions of characters in the strings: first order variables are used to denote positions whereas second order ones denote subsets of positions; then, suitable constructions build an automaton from a given formula and conversely, in such a way that formula and corresponding automaton define the same language. The extension designed by [1] introduced a new basic binary predicate $\rightsquigarrow$ in the syntax of the MSO logic, $x \rightsquigarrow y$ representing the fact that in positions $x$ and $y$ two matching parentheses –named call and return, respectively in their terminology– are located. In the case of FL, however, we have to face new problems.

- Both finite state automata and VPA are real-time machines, i.e., they read one input character at every move; this is not the case with more general machines such as FA, which do not advance the input head when performing flush transitions, and may also apply many flush transitions before the next push or mark which are the transitions that consume input. As a consequence, whereas in the logic characterization of regular and VP languages any first order variable can belong to only one second order variable representing an automaton state, in this case –when the automaton performs a flush– the same position may correspond to different states and therefore belong to different second-order variables.
- In VPL the $\rightsquigarrow$ relation is one-to-one, since any call matches with only one return, if any, and conversely. In FL, instead the same position $y$ can be "paired" with different positions $x$ in correspondence of many flush transitions with no push/mark in between, as it happens for instance when parsing a derivation such as $A \overset{*}{\Rightarrow} \alpha^k A$,

consisting of $k$ immediate derivations $A \Rightarrow \alpha A$; symmetrically the same position $x$ can be paired with many positions $y$.

In essence our goal is to formalize in terms of MSO formulas a complete parsing algorithm for FL, a much more complex algorithm than it is needed for regular and VP languages. The first step to achieve our goal is to define a new relation between (first order variables denoting) the positions in a string.

In some sense the new relation formalizes structural properties of FL strings in the same way as the VPL $\rightsquigarrow$ relation does for VPL; the new relation, however, is more complex than its VPL counterpart in a parallel way, as FL are richer than VPL.

**Definition 7.** *Consider a string $s \in \Sigma^*$ and a OPM $M$. For $0 \leq x < y \leq |s| + 1$, we write $x \curvearrowright y$ iff there exists a sub-string of $\#s\#$ which is a chain ${}^a[r]^b$, such that $a$ is in position $x$ and $b$ is in position $y$.*

*Example 3.* With reference to the string of Figure 1, we have $1 \curvearrowright 3$, $0 \curvearrowright 4$, $6 \curvearrowright 8$, $4 \curvearrowright 8$, and $0 \curvearrowright 9$. In the parsing of the string, these pairs correspond to contexts where a reduce operation is executed (they are listed according to their execution order). For instance, the pair $6 \curvearrowright 8$ is the context for the reduction of the last $call_b$, whereas $4 \curvearrowright 8$ encloses $call_a\ ret_a\ call_b$.

In general $x \curvearrowright y$ implies $y > x+1$, and a position $x$ may be in such a relation with more than one position and vice versa. Moreover, if $s$ is compatible with $M$, then $0 \curvearrowright |s| + 1$.

### 3.1 A Monadic Second-Order Logic over Operator Precedence Alphabets

Let $(\Sigma,M)$ be an OP alphabet. According to Definition 7 it induces the relation $\curvearrowright$ over positions of characters in any words in $\Sigma^*$. Let us define a countable infinite set of first-order variables $x, y, \ldots$ and a countable infinite set of monadic second-order (set) variables $X, Y, \ldots$.

**Definition 8.** *The $MSO_{\Sigma,M}$ (*monadic second-order logic *over $(\Sigma, M)$) is defined by the following syntax:*

$$\varphi := a(x) \mid x \in X \mid x \leq y \mid x \curvearrowright y \mid x = y + 1 \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\varphi \mid \exists X.\varphi$$

*where $a \in \Sigma$, $x, y$ are first-order variables and $X$ is a set variable.*

$\mathrm{MSO}_{\Sigma,M}$ formulae are interpreted over $(\Sigma, M)$ strings and the positions of their characters in the following natural way:

- first-order variables are interpreted over positions of the string;
- second-order variables are interpreted over sets of positions;
- $a(x)$ is true iff the character in position $x$ is $a$;
- $x \curvearrowright y$ is true iff $x$ and $y$ satisfy Definition 7;
- the other logical symbols have the usual meaning.

A sentence is a formula without free variables. The language defined by $\varphi$ is $L(\varphi) = \{s \in \Sigma^* \mid \#s\# \models \varphi\}$ where $\models$ is the standard satisfaction relation.

*Example 4.* Consider the language of Example 1, with the structure implied by its OPM. The following sentence defines it:

$$\forall z \left( \begin{pmatrix} call_a(z) \vee ret_a(z) \\ \vee \\ call_b(z) \vee ret_b(z) \end{pmatrix} \Rightarrow \exists x, y \begin{pmatrix} x \curvearrowright y \wedge x < z < y \\ \wedge \\ hnd(x+1) \wedge rst(y-1) \end{pmatrix} \right).$$

*Example 5.* Consider again Example 1. If we want to add the additional constraint that procedure $b$ cannot directly install handlers (e.g. for security reasons), we may state it through the following formula:

$$\forall z \, (hnd(z) \Rightarrow \neg \exists u \, (call_b(u) \ \wedge \ (u + 1 = z \vee u \curvearrowright z)))$$

We are now ready for the main result.

**Theorem 1.** *A language $L$ over $(\Sigma, M)$ is a FL if and only if there exists a $MSO_{\Sigma,M}$ sentence $\varphi$ such that $L = L(\varphi)$.*

The proof is constructive and structured in the following two subsections.

### 3.2 From $MSO_{\Sigma,M}$ to Floyd Automata

This part of the construction follows the lines of Thomas [12], with some extra technical difficulties due to the need of preserving precedence relations.

**Proposition 1.** *Let $(\Sigma, M)$ be an operator precedence alphabet and $\varphi$ be an $MSO_{\Sigma,M}$ sentence. Then $L(\varphi)$ can be recognized by a Floyd automaton over $(\Sigma, M)$.*

*Proof.* The proof is composed of two steps: first the formula is rewritten so that no predicate symbols nor first order variables are used; then an equivalent FA is built inductively.

Let $\Sigma$ be $\{a_1, a_2, \ldots, a_n\}$. For each predicate symbol $a_i$ we introduce a fresh set variable $X_i$, therefore formula $a_i(x)$ will be translated into $x \in X_i$. Following the standard construction of [12], we also translate every first order variable into a fresh second order variable with the additional constraint that the set it represents contains exactly one position.

Let $\varphi'$ be the formula obtained from $\varphi$ by such a translation and consider any subformula $\psi$ of $\varphi'$: let $X_1, X_2, \ldots, X_n, X_{n+1}, \ldots X_{n+m(\psi)}$ be the (second order) variables appearing in $\psi$. Recall that $X_1, \ldots, X_n$ represent symbols in $\Sigma$, hence they are never quantified.

As usual we interpret formulae over strings; in this case we use the alphabet

$$\Lambda(\psi) = \left\{ \alpha \in \{0,1\}^{n+m(\psi)} \mid \exists! i \text{ s.t. } 1 \leq i \leq n, \ \alpha_i = 1 \right\}$$

A string $w \in \Lambda(\psi)^*$, with $|w| = \ell$, is used to interpret $\psi$ in the following way: the projection over the $j$-th component of $\Lambda(\psi)$ gives an evaluation $\{1, 2, \ldots, \ell\} \to \{0, 1\}$ of $X_j$, for every $1 \leq j \leq n + m(\psi)$.

For any $\alpha \in \Lambda(\psi)$, the projection of $\alpha$ over the first $n$ components encodes a symbol in $\Sigma$, denoted as $symb(\alpha)$. The matrix $M$ over $\Sigma$ can be naturally extended to the OPM $M(\psi)$ over $\Lambda(\psi)$ by defining $M(\psi)_{\alpha,\beta} = M_{symb(\alpha),symb(\beta)}$ for any $\alpha, \beta \in \Lambda(\psi)$.

We now build a FA $\mathcal{A}$ equivalent to $\varphi'$. The construction is inductive on the structure of the formula: first we define the FA for all atomic formulae. We give here only the construction for $\curvearrowright$, since for the other ones the construction is standard and is the same as in [12].

Figure 3 represents the FA for atomic formula $\psi = X_i \curvearrowright X_j$ (notice that $i, j > n$). For the sake of brevity, we use notation $[X_i]$ to represent the set of all tuples $\Lambda(\psi)$ having the $i$-th component equal to 1; notation $[\bar{X}]$ represents the set of all tuples in $\Lambda(\psi)$ having both $i$-th and $j$-th components equal to 0.

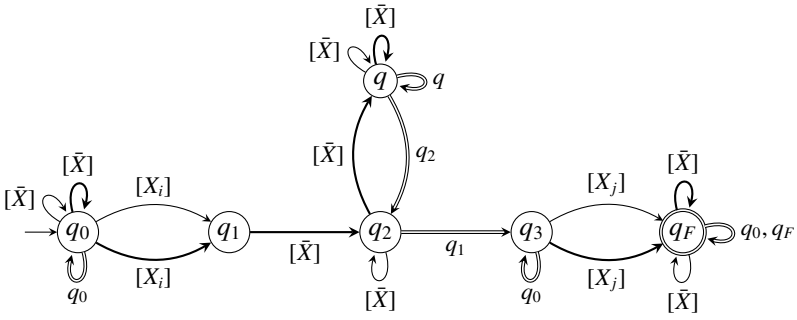



**Fig. 3.** Floyd automaton for atomic formula $\psi = X_i \curvearrowright X_j$

The automaton, after a generic sequence of moves corresponding to visiting an irrelevant portion of the syntax tree, when reading $X_i$ performs either a mark or a push move, depending on whether $X_i$ is a leftmost leaf of the tree or not; then it visits the subsequent subtree ending with a flush labeled $q_1$; at this point, if it reads $X_j$, it accepts anything else will follow the examined fragment.

Then, a natural inductive path leads to the construction of the automaton associated with a generic MSO formula: the disjunction of two subformulae can be obtained by building the union automaton of the two corresponding FA; similarly for negation. The existential quantification of $X_i$ is obtained by projection erasing the $i$-th component. Notice that all matrices $M(\psi)$ are well defined for any $\psi$ because the first $n$ components of the alphabet are never erased by quantification. The alphabet of the automaton equivalent to $\varphi'$ is $\Lambda(\varphi') = \{0, 1\}^n$, which is in bijection with $\Sigma$.

### 3.3 From Floyd Automata to $\mathrm{MSO}_{\Sigma,M}$

Unlike the previous construction, this part of the theorem sharply departs from traditional techniques.

Let $\mathcal{A}$ be a deterministic FA over $(\Sigma, M)$. We build an $\mathrm{MSO}_{\Sigma,M}$ sentence $\varphi$ such that $L(\mathcal{A}) = L(\varphi)$. The main idea for encoding the behavior of the FA is based on assigning the states visited during its run to positions along the same lines stated by Büchi [12] and extended for VPL [1]. Unlike finite state automata and VPA, however, FA do not work on-line. Hence, it is not possible to assign a single state to every position. Let $Q = \{q_0, q_1, \ldots, q_N\}$ be the states of $\mathcal{A}$ with $q_0$ initial; as usual, we will use second order

variables to encode them. We shall need three different sets of second order variables, namely $P_0, P_1, \ldots, P_N$, $M_0, M_1, \ldots, M_N$ and $F_0, F_1, \ldots, F_N$: set $P_i$ contains those positions of $s$ where state $i$ may be assumed after a push transition. $M_i$ and $F_i$ represent the state reached after a flush: $F_i$ contains the positions where the flush occurs, whereas $M_i$ contains the positions preceding the corresponding mark. Notice that any position belongs to only one $P_i$, whereas it may belong to several $F_i$ or $M_i$ (see Figure 4).

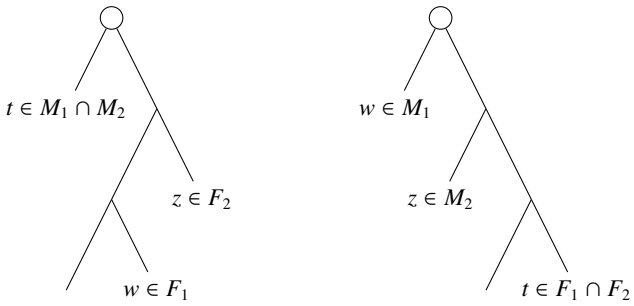



**Fig. 4.** Example trees with a position $t$ belonging to more than one $M_i$ (left) and $F_i$ (right).

We show that $\mathcal{A}$ accepts a string $s$ iff $\#s\# \models \varphi$, where

$$\varphi := \exists P_0, P_1, \ldots, P_N, M_0, M_1, \ldots, M_N, F_0, F_1, \ldots, F_N, e\ (0 \in P_0 \wedge \\ \wedge \bigvee_{i \in F} e \in F_i \wedge \neg\exists x(e+1 < x) \wedge \#(e+1) \wedge \varphi_\delta \wedge \varphi_{exist} \wedge \varphi_{unique}). \tag{3}$$

The first clause encodes the initial state, whereas the second, third and fifth ones encode the final states. We use variable $e$ to refer to the *end* of $s$, i.e., $e$ equals the last position $|s|$. The other remaining clauses are defined in the following: the fourth one encodes the transition function; the last ones together encode the fact that there exists exactly one state that may be assumed by a push transition in any position, and exactly one state as a consequence of a flush transition.

For convenience we introduce the following notational conventions.

$$x \circ y := \bigvee_{M_{a,b} = \circ} a(x) \wedge b(y), \text{ for } \circ \in \{\lessdot, \doteq, \gtrdot\}$$

$$\mathrm{Tree}(x,z,w,y) := \begin{pmatrix} x \curvearrowright y \wedge \\ (x+1 = z \ \vee \ x \curvearrowright z) \wedge \neg\exists t(x < t < z \wedge x \curvearrowright t) \wedge \\ (w+1 = y \ \vee \ w \curvearrowright y) \wedge \neg\exists t(w < t < y \wedge t \curvearrowright y) \end{pmatrix}$$

$$\mathrm{Succ}_k(x,y) := x + 1 = y \wedge x \in P_k$$

$$\mathrm{Next}_k(x,y) := x \curvearrowright y \wedge x \in M_k \wedge y - 1 \in F_k$$

$$\mathrm{Flush}_k(x,y) := x \curvearrowright y \ \wedge \ x \in M_k \ \wedge y - 1 \in F_k \ \wedge \\ \exists z, w \left( \mathrm{Tree}(x,z,w,y) \wedge \bigvee_{i=0}^{N} \bigvee_{j=0}^{N} \begin{pmatrix} \delta(q_i, q_j) = q_k \wedge \\ (\mathrm{Succ}_i(w,y) \vee \mathrm{Next}_i(w,y)) \wedge \\ (\mathrm{Succ}_j(x,z) \vee \mathrm{Next}_j(x,z)) \end{pmatrix} \right)$$

$$\mathrm{Tree}_{i,j}(x,z,w,y) := \mathrm{Tree}(x,z,w,y) \wedge \begin{pmatrix} \mathrm{Succ}_i(w,y) \vee \mathrm{Flush}_i(w,y)) \wedge \\ (\mathrm{Succ}_j(x,z) \vee \mathrm{Flush}_j(x,z)) \end{pmatrix}$$

*Remark.* If ${}^a[c_1 c_2 \dots c_\ell]^b$ is a simple chain with support

$$q_i = q_{t_0} \xrightarrow{c_1} q_{t_1} \xrightarrow{c_2} \dots \xrightarrow{c_\ell} q_{t_\ell} \overset{q_0}{\Longrightarrow} q_k \tag{4}$$

then $\mathrm{Tree}_{t_0,t_\ell}(0, 1, \ell, \ell+1)$ and $\mathrm{Flush}_k(0, \ell+1)$ hold; if ${}^a[s_0 c_1 s_1 c_2 \dots c_\ell s_\ell]^b$ is a composed chain with support

$$q_i = q_{t_0} \overset{s_0}{\rightsquigarrow} q_{f_0} \xrightarrow{c_1} q_{t_1} \overset{s_1}{\rightsquigarrow} q_{f_1} \xrightarrow{c_2} \dots \xrightarrow{c_g} q_{t_g} \overset{s_g}{\rightsquigarrow} q_{f_g} \dots \xrightarrow{c_\ell} q_{t_\ell} \overset{s_\ell}{\rightsquigarrow} q_{f_\ell} \overset{q_{f_0}}{\Longrightarrow} q_k \tag{5}$$

then by induction we can see that $\mathrm{Tree}_{f_\ell,f_0}(0, x_1, x_\ell, |s|+1)$ and $\mathrm{Flush}_k(0, |s|+1)$ hold, where $x_g$ is the position of $c_g$ for $g = 1, 2, \dots, \ell$.

On the basis of the above definitions and properties, it is possible to bind $\mathcal{A}$ to suitable axioms $\varphi_\delta$, $\varphi_{exist}$, and $\varphi_{unique}$ that are satisfied by any chain with a support in $\mathcal{A}$, and in turn guarantee the existence of an $\mathcal{A}$'s support for every chain.

For the sake of brevity we report here only a (small) part of $\varphi_\delta$ which should provide enough evidence of the construction and of its correctness. The complete axiomatization and equivalence proof (based on a natural induction) are given in [11].

The following *Forward formulae* formalize how $\mathcal{A}$ enters new states through push and flush transitions.

$$\varphi_{push\_fw} := \forall x, y \bigwedge_{i=0}^{N} \begin{pmatrix} (x \lessdot y \vee x \doteq y) \wedge a(y) \\ \wedge \\ \mathrm{Succ}_i(x, y) \vee \mathrm{Flush}_i(x, y) \end{pmatrix} \Rightarrow y \in P_{\delta(q_i,a)}$$

$$\varphi_{flush\_fw} := \forall x, z, w, y \bigwedge_{i=0}^{N} \bigwedge_{j=0}^{N} \left( \mathrm{Tree}_{i,j}(x, z, w, y) \Rightarrow \begin{matrix} x \in M_{\delta(q_i,q_j)} \\ \wedge \\ y - 1 \in F_{\delta(q_i,q_j)} \end{matrix} \right)$$

Somewhat symmetric *Backward formulae* allow to reconstruct (in a unique way) $\mathcal{A}$'s states that lead to a given state.

Finally, for any chain $\#s\#$, by the complete axiom $\varphi$ defined in (3) we obtain the following proposition which, together with Proposition 1, completes Theorem 1.

**Proposition 2.** *For any Floyd automaton $\mathcal{A}$ there exists an $MSO_{\Sigma,M}$ sentence $\varphi$ such that $L(\mathcal{A}) = L(\varphi)$.*

## 4 Conclusions and Future Work

This paper at last completes a research path that began more than four decades ago and was resumed only recently with new -and old- goals. FLs enjoy most of the nice properties that made regular languages highly appreciated and applied to achieve decidability and, therefore, automatic analysis techniques. In this paper we added to the above collection the ability to formalize and analyze FL by means of suitable MSO logic formulae. New research topics, however, stimulate further investigation. Here we briefly mention only two mutually related ones. On the one hand, FA devoted to analyze strings should be extended in the usual way into suitable transducers. They could

be applied, e.g. to translate typical mark-up languages such as XML, HTML, LaTeX, ... into their end-user view. Such languages, which motivated also the definition of VPL, could be classified as "explicit parenthesis languages" (EPL), i.e. languages whose syntactic structure is explicitly apparent in the input string. On the other hand, we plan to start from the remark that VPL are characterized by a well precise shape of the OPM [5] to characterize more general classes of such EPL: for instance the language of Example 1 is such a language that is not a VPL. Another notable feature of FL, in fact, is that they are suitable as well to parse languages with implicit syntax structure such as most programming languages, as well as to analyze and translate EPL.

## References

1. Alur, R., Madhusudan, P.: Adding nesting structure to words. Journ. ACM 56(3) (2009)
2. Barenghi, A., Viviani, E., Reghizzi, S.C., Mandrioli, D., Pradella, M.: PAPAGENO: a parallel parser generator for operator precedence grammars. In: 5th International Conference on Software Language Engineering, SLE 2012 (2012)
3. Berstel, J., Boasson, L.: Balanced Grammars and Their Languages. In: Brauer, W., Ehrig, H., Karhumäki, J., Salomaa, A. (eds.) Formal and Natural Computing. LNCS, vol. 2300, pp. 3–25. Springer, Heidelberg (2002)
4. Courcelle, B., Engelfriet, J.: Graph Structure and Monadic Second-Order Logic: A Language-Theoretic Approach. Cambridge University Press (2012)
5. Crespi Reghizzi, S., Mandrioli, D.: Operator precedence and the visibly pushdown property. Journal of Computer and System Science 78(6), 1837–1867 (2012)
6. Crespi Reghizzi, S., Mandrioli, D., Martin, D.F.: Algebraic properties of operator precedence languages. Information and Control 37(2), 115–133 (1978)
7. Floyd, R.W.: Syntactic analysis and operator precedence. Journ. ACM 10(3), 316–333 (1963)
8. Grune, D., Jacobs, C.J.: Parsing techniques: a practical guide. Springer, New York (2008)
9. Lonati, V., Mandrioli, D., Pradella, M.: Precedence automata and languages (2010), http://arxiv.org/abs/1012.2321
10. Lonati, V., Mandrioli, D., Pradella, M.: Precedence Automata and Languages. In: Kulikov, A., Vereshchagin, N. (eds.) CSR 2011. LNCS, vol. 6651, pp. 291–304. Springer, Heidelberg (2011)
11. Lonati, V., Mandrioli, D., Pradella, M.: Logic characterization of Floyd languages (2012), http://arxiv.org/abs/1204.4639
12. Thomas, W.: Automata on infinite objects. In: Handbook of Theoretical Computer Science, vol. B: Formal Models and Sematics, pp. 133–192 (1990)

# Incomplete Transition Complexity of Some Basic Operations*

Eva Maia**, Nelma Moreira, and Rogério Reis

CMUP & DCC, Faculdade de Ciências da Universidade do Porto,
Rua do Campo Alegre, 4169-007 Porto, Portugal
{emaia,nam,rvr}@dcc.fc.up.pt

**Abstract.** Y. Gao *et al.* studied for the first time the transition complexity of Boolean operations on regular languages based on not necessarily complete DFAs. For the intersection and the complementation, tight bounds were presented, but for the union operation the upper and lower bounds differ by a factor of two. In this paper we continue this study by giving tight upper bounds for the concatenation, the Kleene star and the reversal operations. We also give a new tight upper bound for the transition complexity of the union, which refutes the conjecture presented by Y. Gao, et al.

## 1 Introduction

The descriptional complexity of regular languages has recently been extensively investigated. For deterministic finite automata (DFA), the complexity measure usually studied is the state complexity (number of states of the complete minimal DFA) [1,2,8,15,16,17], while for nondeterministic finite automata (NFA) both state and transition complexity were considered [5,7,8,9,12], being this last one a more interesting measure. Considering complete DFAs (when the transition function is total) it is obvious that the transition complexity is the product of the alphabet size by the state complexity. But in many applications where large alphabets need to be considered or, in general, when very sparse transition functions take place, partial transition functions are very convenient. Examples include lexical analysers, discrete event systems, or any application that uses dictionaries where compact automaton representations are essential [3,4,11]. Thus, it makes sense to study the transition complexity of regular languages based on not necessarily complete DFAs.

Y. Gao *et al.* [6] studied for the first time the transition complexity of Boolean operations on regular languages based on not necessarily complete DFAs. For the intersection and the complementation, tight bounds were presented, but

* This work was partially funded by the European Regional Development Fund through the programme COMPETE and by the Portuguese Government through the FCT under projects PEst-C/MAT/UI0144/2011 and CANTE-PTDC/EIA-CCO/101904/2008.

** Eva Maia is funded by FCT grant SFRH/BD/78392/2011.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 319–331, 2013.

for the union operation the upper and lower bounds differ by a factor of two. Nevertheless, they conjectured a tight upper bound for this operation.

In this paper, we continue this study by extending the analysis to the concatenation, the Kleene star and the reversal operations. For these operations tight upper bounds are given. We also give a tight upper bound for the transition complexity of the union, which refutes the conjecture presented by Y. Gao *et al.*. The same study was made for unary languages. The algorithms and the witness language families used in this work, although new, are based on the ones of Yu *et al.* [18] and several proofs required new techniques. In the Tables 1 and 2 we summarize our results (in bold) as well as some known results for other descriptional complexity measures. All the proofs not present in this paper, can be found in an extended version of this work[1]

## 2 Preliminaries

We recall some basic notions about finite automata and regular languages. For more details, we refer the reader to the standard literature [10,14,13].

A *deterministic finite automaton* (DFA) is a five-tuple $A = (Q, \Sigma, \delta, q_0, F)$ where $Q$ is a finite set of states, $\Sigma$ is a finite input alphabet, $q_0$ in $Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta$ is the transition function mapping $Q \times \Sigma \to Q$. The transition function can be extended to sets — $2^Q \times \Sigma \to 2^Q$. A DFA is *complete* if the transition function ($\delta$) is total. In this paper we consider the DFAs to be not necessarily complete. For $s \in Q$ and $\tau \in \Sigma$, if $\delta(s, \tau)$ is defined we write $\delta(s, \tau) \downarrow$, and $\delta(s, \tau) \uparrow$, otherwise, and, when defining a DFA, an assignment $\delta(s, \tau) = \uparrow$ means that the transition is undefined.

The transition function is extended in the usual way to a function $\hat{\delta} : Q \times \Sigma^\star \to Q$. This function can also be used in sets — $\hat{\delta} : 2^Q \times \Sigma^\star \to 2^Q$. The *language* accepted by $A$ is $\mathcal{L}(A) = \{w \in \Sigma^\star \mid \hat{\delta}(q_0, w) \in F\}$. Two DFAs are *equivalent* if they accept the same language. For each regular language, considering a total transition function or not a total one, there exists a unique minimal complete DFA with a least number of states. The *left-quotient* of $L \subseteq \Sigma^\star$ by $x \in \Sigma^\star$ is $D_x L = \{z \mid xz \in L\}$. The equivalence relation $R_L \subseteq \Sigma^\star \times \Sigma^\star$ is defined by $(x, y) \in R_L$ if and only if $D_x L = D_y L$. The *Myhill-Nerode Theorem* states that a language $L$ is regular if and only if $R_L$ has a finite number of equivalence classes, *i.e.*, $L$ has a finite number of left quotients. This number is equal to the number of states of the minimal complete DFA. The *state complexity*, $sc(L)$, of a regular language $L$ is the number of states of the minimal complete DFA of $L$. If the minimal DFA is not complete its number of states is the number of left quotients minus one (the sink state is removed).

The *incomplete state complexity* of a regular language $L$ ($isc(L)$) is the number of states of the minimal DFA, not necessarily complete, that accepts $L$. Note that $isc(L)$ is either equal to $sc(L) - 1$ or to $sc(L)$. The *incomplete transition complexity*, $itc(L)$, of a regular language $L$ is the minimal number of transitions

[1] http://www.dcc.fc.up.pt/Pubs/TReports/TR12/dcc-2012-02

over all DFAs that accepts $L$. Whenever the model is explicitly given we refer only to *state* or *transition* complexity, by omitting the term incomplete[2] . When we talk about the minimal DFA, we refer the DFA with the minimal number of states and transitions because we have the following result:

**Proposition 1.** *The state-minimal DFA, not necessarily complete, which recognizes $L$ has the minimal number of transitions of any DFA that recognizes $L$.*

A transition labeled by $\tau \in \Sigma$ is named by *$\tau$-transition* (represented by $\delta(s, \tau)$, where $s \in Q$) and the number of $\tau$-transitions of a DFA $A$ is denoted by $t(\tau, A)$. The *$\tau$-transition complexity* of $L$, $itc_\tau(L)$ is the minimal number of $\tau$-transitions of any DFA recognizing $L$. In [6, Lemma 2.1] it was showed that the minimal DFA accepting $L$ has the minimal number of $\tau$-transitions of any DFA accepting $L$. From this and Proposition 1 follows that $itc(L) = \sum_{\tau \in \Sigma} itc_\tau(L)$.

The *state complexity of an operation* on regular languages is the (worst-case) state complexity of a language resulting from the operation, considered as a function of the state complexities of the operands. The (worst-case) transition complexity of an operation is defined in the same way. Usually an *upper bound* is obtained by providing an algorithm, which given DFAs as operands, constructs a DFA that accepts the language resulting from the referred operation. The number of states or transitions of the resulting DFA are upper bounds for the state or the transition complexity of the operation, respectively. To prove that an upper bound is *tight*, for each operand we can give a family of languages (parametrized by the complexity measures), such that the resulting language achieves that upper bound. For determining the transition complexity of a language operation, we also consider the following measures and refined numbers of transitions. Given a DFA $A = (Q, \Sigma, \delta, q_0, F)$ and $\tau \in \Sigma$, let $f(A) = |F|$, $i(\tau, A)$ be the number of $\tau$-transitions leaving the initial state $q_0$, $u(\tau, A)$ be the number of states without $\tau$-transitions, i.e. $u(\tau, A) = |Q| - t(\tau, A)$, and $\bar{u}(\tau, A)$ be the number of non-final states without $\tau$-transitions. Whenever there is no ambiguity we omit $A$ from the above definitions. If $t(\tau, A) = |Q|$ we say that $A$ is *$\tau$-complete*, and *$\tau$-incomplete* otherwise. All the above measures, can be defined for a regular language $L$, considering the measure values for its minimal DFA. Thus, we have, respectively, $f(L)$, $i_\tau(L)$, $u_\tau(L)$, and $\bar{u}_\tau(L)$. We also prove that the upper bounds are maximal when $f(L)$ is minimal.

## 3 Incomplete Transition Complexity of the Union

It was shown by Y. Gao *et al.* [6] that $itc(L_1 \cup L_2) \leq 2(itc(L_1)itc(L_2) + itc(L_1) + itc(L_2))$. The lower bound $itc(L_1)itc(L_2) + itc(L_1) + itc(L_2) - 1$ was given for particular ternary language families which state complexities are relatively prime. The authors conjectured, also, that $itc(L_1 \cup L_2) \leq itc(L_1)itc(L_2) + itc(L_1) + itc(L_2)$, when $itc(L_i) \geq 2$, $i = 1, 2$.

In this section we present an upper bound for the state complexity and we give a new upper bound for the transition complexity of the union of two regular

[2] In [6] the author refer $sc(L)$ and $tc(L)$ instead of $isc(L)$ and $itc(L)$.

languages. We also present families of languages for which these upper bounds are reached, witnessing that these bounds are tight.

### 3.1 An Upper Bound

In the following we describe the algorithm for the union of two DFAs that was presented by Y. Gao *et al.* [6, Lemma 3.1.]. Let $A = (Q, \Sigma, \delta_A, q_0, F_A)$ and $B = (P, \Sigma, \delta_B, p_0, F_B)$ be two DFAs ($-1 \notin Q$ and $-1 \notin P$). Let $C = (R, \Sigma, \delta_C, r_0, F_C)$ be a new DFA with $R = (Q \cup \{-1\}) \times (P \cup \{-1\})$, $r_0 = (q_0, p_0)$, $F_C = (F_A \times (Q \cup \{-1\})) \cup ((P \cup \{-1\}) \times F_B)$ and

$$\delta_C((q'_A, p'_B), \tau) = \begin{cases} (\delta_A(q'_A, \tau), \delta_B(p'_B, \tau)) & \text{if } \delta_A(q'_A, \tau) \downarrow \ \wedge \ \delta_B(p'_B, \tau) \downarrow, \\ (\delta_A(q'_A, \tau), -1) & \text{if } \delta_A(q'_A, \tau) \downarrow \ \wedge \ \delta_B(p'_B, \tau) \uparrow, \\ (-1, \delta_B(p'_B, \tau)) & \text{if } \delta_A(q'_A, \tau) \uparrow \ \wedge \ \delta_B(p'_B, \tau) \downarrow, \\ \uparrow & \text{otherwise,} \end{cases}$$

where $\tau \in \Sigma$, $q'_A \in Q \cup \{-1\}$ and $p'_B \in P \cup \{-1\}$. Note that $\delta_A(-1, \tau)$ and $\delta_B(-1, \tau)$ are always undefined, and the pair $(-1, -1)$ never occurs in the image of $\delta_C$. It is easy to see that DFA $C$ accepts the language $\mathcal{L}(A) \cup \mathcal{L}(B)$. We can determine the number of states and transitions which are sufficient for any DFA $C$ resulting from the previous algorithm:

**Proposition 2 ([6]).** *For any $m$-state DFA $A$ and any $n$-state DFA $B$, $mn + m + n$ states are sufficient for a DFA accepting $\mathcal{L}(A) \cup \mathcal{L}(B)$.*

**Proposition 3.** *For any regular languages $L_1$ and $L_2$ with $isc(L_1) = m$ and $isc(L_2) = n$, one has*

$$itc(L_1 \cup L_2) \leq itc(L_1)(1 + n) + itc(L_2)(1 + m) - \sum_{\tau \in \Sigma} itc_\tau(L_1) itc_\tau(L_2).$$

The proof of Proposition 3 follows from Lemma 3.1 in Y. Gao *et al.*

### 3.2 Worst-Case Witnesses

In this section, we show that the upper bounds given in Proposition 2 and Proposition 3 are tight. We consider two cases, parameterized by the state complexities of the language operands: $m \geq 2$ and $n \geq 2$; and $m = 1$ and $n \geq 2$ (or vice versa). Note that in all that follows we consider automaton families over a binary alphabet, $\Sigma = \{a, b\}$. Using Myhill-Nerode theorem, it is easy to prove that these automata are minimal because all their states correspond to different left quotients.

**Theorem 1.** *For any integers $m \geq 2$ and $n \geq 2$, exist an $m$-state DFA $A$ with $r = m$ transitions and an $n$-state DFA $B$ with $s = 2n - 1$ transitions such that any DFA accepting $\mathcal{L}(A) \cup \mathcal{L}(B)$ needs, at least, $mn + m + n$ states and $(r + 1)(s + 1)$ transitions.*

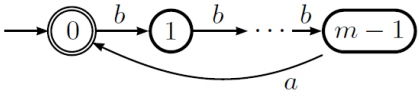



**Fig. 1.** DFA $A$ with $m$ states

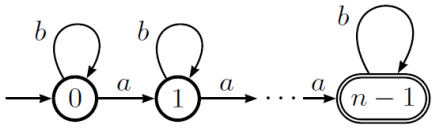



**Fig. 2.** DFA $B$ with $n$ states

*Proof.* Let $A = (Q, \Sigma, \delta_A, 0, F_A)$ with $Q = \{0, \ldots, m-1\}$, $F_A = \{0\}$, $\delta_A(m-1, a) = 0$, and $\delta_A(i, b) = i + 1,\ 0 \leq i < m - 1$; and $B = (P, \Sigma, \delta_B, 0, F_B)$ with $P = \{0, \ldots, n-1\}$, $F_B = \{n-1\}$, $\delta_B(i, a) = i+1,\ 0 \leq i < n-1$, and $\delta_B(i, b) = i,\ 0 \leq i \leq n-1$ (see Figure 1 and Figure 2). Let $C$ be the DFA constructed by the previous algorithm, which can be proved to be minimal. We only consider the part of the theorem corresponding to the transitions. We name $\tau$-transitions of the DFA $A$ by $\alpha_i$ $(1 \leq i \leq t(\tau, A))$ and the undefined $\tau$-transitions named by $\bar{\alpha}_i$ $(1 \leq i \leq u(\tau, A) + 1)$, the $\tau$-transitions of the DFA $B$ by $\beta_j$ $(1 \leq j \leq t(\tau, B))$ and the undefined $\tau$-transitions by $\bar{\beta}_j$ $(1 \leq j \leq u(\tau, B) + 1)$. We need to consider one more undefined transition in each DFA that corresponds to the $\tau$-transition of the state $-1$ added to $Q$ and $P$ in the union algorithm. Then, each of the $\tau$-transitions of DFA $C$ can only have one of the following three forms: $(\alpha_i, \beta_j)$, $(\bar{\alpha}_i, \beta_j)$, or $(\alpha_i, \bar{\beta}_j)$. Thus, the DFA $C$ has: $mn + n - m + 1$ $a$-transitions because there exist $n - 1$ $a$-transitions of the form $(\alpha_i, \beta_j)$; 2 $a$-transitions of the form $(\alpha_i, \bar{\beta}_j)$; and $m(n-1)$ $a$-transitions of the form $(\bar{\alpha}_i, \beta_j)$; and $mn + m + n - 1$ $b$-transitions because there exist $(m-1)n$ transitions of the form $(\alpha_i, \beta_j)$; $m - 1$ $b$-transitions of the form $(\alpha_i, \bar{\beta}_j)$; and $2n$ $b$-transitions of the form $(\bar{\alpha}_i, \beta_j)$.

As $r = m$ and $n = \frac{s+1}{2}$ the DFA $C$ has $(r + 1)(s + 1)$ transitions. □

The referred conjecture $itc(L_1 \cup L_2) \leq itc(L_1)itc(L_2) + itc(L_1) + itc(L_2)$ fails for these families because one has $itc(L_1 \cup L_2) = itc(L_1)itc(L_2) + itc(L_1) + itc(L_2) + 1$. Note that $r = itc(L_1)$ and $s = itc(L_2)$, thus $(r+1)(s+1) = (itc(L_1)+1)(itc(L_2)+1) = itc(L_1)itc(L_2) + itc(L_1) + itc(L_2) + 1$.

**Theorem 2.** *For any integer $n \geq 2$, exists an 1-state DFA $A$ with one transition and an $n$-state DFA $B$ with $s = 2n - 1$ transitions such that any DFA accepting $\mathcal{L}(A) \cup \mathcal{L}(B)$ has, at least, $2n + 1$ states and $2(s + 1)$ transitions.*

*Proof (Sketch).* Let $A = (Q, \Sigma, \delta_A, 0, F_A)$ with $Q = \{0\}$, $F_A = \{0\}$, $\delta_A(0, a) = 0$, and consider the DFA $B$ defined in the previous case. □

## 4 Incomplete Transition Complexity of the Concatenation

In this section we will show how many states and transitions are sufficient and necessary, in the worst case, for a DFA to accept the concatenation of two DFAs.

### 4.1 An Upper Bound

The following algorithm computes a DFA for the concatenation of a DFA $A = (Q, \Sigma, \delta_A, q_0, F_A)$, where $-1 \notin Q$ and $|Q| = n$, with a DFA $B = (P, \Sigma, \delta_B, p_0, F_B)$, where $|P| = m$ . Let $C = (R, \Sigma, \delta_C, r_0, F_C)$ be a new DFA with $R = (Q \cup \{-1\}) \times 2^P - F_A \times 2^{P-\{p_0\}}$, $r_0 = \langle q_0, \emptyset \rangle$ if $q_0 \notin F_A$ or $r_0 = \langle q_0, \{p_0\} \rangle$ if $q_0 \in F_A$, $F_C = \{\langle q, T \rangle \in R \mid T \cap F_B \neq \emptyset\}$, and for $a \in \Sigma$, $\delta_C(\langle q, T \rangle, a) = \langle q', T' \rangle$ with $q' = \delta_A(q, a)$, if $\delta_A(q, a) \downarrow$ or $q' = -1$ otherwise, and $T' = \delta_B(T, a) \cup \{p_0\}$ if $q' \in F_A$ or $T' = \delta_B(T, a)$ otherwise. DFA $C$ recognizes the language $\mathcal{L}(A)\mathcal{L}(B)$.

The following results determine the number of states and transitions which are sufficient for any DFA $C$ resulting from the previous algorithm.

**Proposition 4.** *For any $m$-state DFA $A$ and any $n$-state DFA $B$, $(m+1)2^n - f(A)2^{n-1} - 1$ states are sufficient for any DFA accepting $\mathcal{L}(A)\mathcal{L}(B)$.*

Note that the minus one in the formula is due to the removal of the state $(-1, \emptyset)$.

**Corollary 1.** *The formula in Proposition 4 is maximal when $f(A) = 1$.*

Given an automaton $A$, the alphabet can be partitioned in two sets $\Sigma_c^A$ and $\Sigma_i^A$ such that $\tau \in \Sigma_c^A$ if $A$ is $\tau$-complete, or $\tau \in \Sigma_i^A$ otherwise. In the same way, considering two automata $A$ and $B$, the alphabet can be divided into four disjoint sets $\Sigma_{ci}$, $\Sigma_{cc}$, $\Sigma_{ii}$ and $\Sigma_{ic}$. As before, these notations can be extended to regular languages considering their minimal DFA.

**Proposition 5.** *For any regular languages $L_1$ and $L_2$ with $isc(L_1) = m$, $isc(L_2) = n$, $u_\tau = u_\tau(L_2)$, $f = f(L_1)$ and $\bar{u}_\tau = \bar{u}_\tau(L_1)$, one has*

$$itc(L_1L_2) \leq |\Sigma|(m+1)2^n - |\Sigma_c^{L_2}|(f2^{n-1}+1) - \sum_{\tau \in \Sigma_i^{L_2}} (2^{u_\tau} + f2^{itc_\tau(L_2)}) - \\ - \sum_{\tau \in \Sigma_{ii}} \bar{u}_\tau 2^{u_\tau} - \sum_{\tau \in \Sigma_{ic}} \bar{u}_\tau.$$

*Proof.* Let $A$ and $B$ be the minimal DFAs that recognize $L_1$ and $L_2$, respectively. Consider the DFA $C$ such that $\mathcal{L}(C) = \mathcal{L}(A)\mathcal{L}(B)$ and $C$ is constructed using the algorithm described above. We name the $\tau$-transitions of $A$ and $B$ as in the proof of the Theorem 1, with a slight modification: $1 \leq j \leq u(\tau, B)$. The $\tau$-transitions of $C$ are pairs $(\theta, \gamma)$ where $\theta$ is an $\alpha_i$ or $\bar{\alpha}_i$, and $\gamma$ is a set of $\beta_j$ or $\bar{\beta}_j$. By construction, $C$ cannot have transitions where $\theta$ is an $\bar{\alpha_i}$, and $\gamma$ is a set with only $\bar{\beta}_j$, because these would correspond to pairs of undefined transitions.

Let us count the number of $\tau$-transitions of $C$. If $\tau \in \Sigma_{ci}$, the number of $C$ $\tau$-transitions is $(t(\tau, A)+1)2^{t(\tau,B)+u(\tau,B)} - 2^{u(\tau,B)} - f(A)2^{t(\tau,B)}$. The number of $\theta$s is $t(\tau, A)+1$ and the number of $\gamma$s is $2^{t(\tau,B)+u(\tau,B)}$. From the product we need to remove the $2^{u(\tau,B)}$ sets of transitions of the form $(v, \emptyset)$ where $v$ corresponds to the undefined $\tau$-transition leaving the added state $-1$ of DFA $A$. If $\theta$ corresponds to a transition that leaves a final state of $A$, then $\gamma$ needs to include the initial state of the $B$. Thus we also remove the $f(A)2^{t(\tau,B)}$ pairs. If $\tau \in \Sigma_{cc}$, $C$ has $(t(\tau, A) + 1)2^{t(\tau,B)} - 1 - f(A)2^{t(\tau,B)-1}$ $\tau$-transitions. In this case, $u(\tau, B) = 0$.

The only pair we need to remove is $(v, \emptyset)$ where $v$ corresponds to the undefined $\tau$-transition leaving the added state $-1$ of DFA $A$. Analogously, if $\tau \in \Sigma_{ii}$, $C$ has $(t(\tau, A) + u(\tau, A) + 1)2^{t(\tau,B)+u(\tau,B)} - (\bar{u}(\tau, A) + 1)2^{u(\tau,B)} - f(A)2^{t(\tau,B)}$ $\tau$-transitions. Finally, if $\tau \in \Sigma_{ic}$, $C$ has $(t(\tau, A) + u(\tau, A) + 1)2^{t(\tau,B)} - (\bar{u}(\tau, A) + 1) - f(A)2^{t(\tau,B)-1}$ $\tau$-transitions.

Thus, after some simplifications, the right side of the inequality in the proposition holds. □

### 4.2 Worst-Case Witnesses

The following results show that the complexity upper bounds found in Propositions 4 and 5 are tight. As in the previous section we need to consider three different cases, according to the state and transition complexities of the operands. All following automaton families have $\Sigma = \{a, b, c\}$. For these automata, it is easy to prove that they are minimal. It is also possible to prove that there cannot exist binary language families that reach the upper bounds.

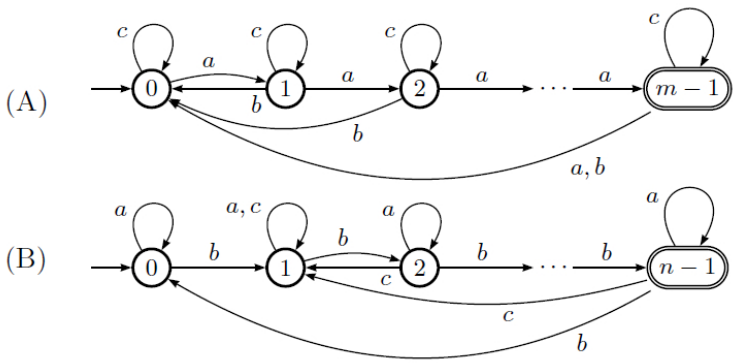



**Fig. 3.** DFA $A$ with $m$ states and DFA $B$ with $n$ states

**Theorem 3.** *For any integers $m \geq 2$ and $n \geq 2$ exist an $m$-state DFA $A$ with $r = 3m - 1$ transitions and an $n$-state DFA $B$ with $s = 3n - 1$ transitions such that any DFA accepting $\mathcal{L}(A)\mathcal{L}(B)$ has, at least, $(m+1)2^n - 2^{n-1} - 1$ states and $(r + 1)2^{\frac{s+1}{3}} + 3.2^{\frac{s-2}{3}} - 5$ transitions.*

*Proof.* Let $A = (Q, \Sigma, \delta_A, 0, F_A)$ with $Q = \{0, \ldots, m - 1\}$, $F_A = \{m - 1\}$, and $\delta_A(i, a) = i + 1 \bmod m$, if $0 \leq i \leq m - 1$, $\delta_A(i, b) = 0$, if $1 \leq i \leq m - 1$, and $\delta_A(i, c) = i$ if $0 \leq i \leq m - 1$; and $B = (P, \Sigma, \delta_B, 0, F_B)$ with $P = \{0, \ldots, n - 1\}$, $F_B = \{n-1\}$, $\delta_B(i, a) = i$ if $0 \leq i \leq n-1$, $\delta_B(i, b) = i+1 \bmod n$, if $0 \leq i \leq n-1$, and $\delta_B(i, c) = 1$, $1 \leq i \leq n - 1$ (see Figure 3). Consider the DFA $C$, constructed with the previous algorithm, such that $\mathcal{L}(C) = \mathcal{L}(A)\mathcal{L}(B)$ and which can be proved to be minimal. We only prove the part of the theorem correspondent to

the number of transitions. As in Proposition 5, the transitions of $C$ are pairs $(\theta, \gamma)$. Then, $C$ has: $(m+1)2^n - 2^{n-1} - 1$, $a$-transitions. There are $m+1$ $\theta$s and $2^n$ $\gamma$s, from which we need to remove the transition pair $(-1, \emptyset)$. If $\theta$ is a transition which leaves a final state of $A$, $\gamma$ needs to include the transition that leaves the initial state of $B$. Thus, $2^{n-1}$ pairs are removed; $(m+1)2^n - 2^{n-1} - 2$, $b$-transitions. Here, the transition $(\bar{\theta}, \emptyset)$ is removed; and $(m+1)2^n - 2^{n-1} - 2$, $c$-transitions. This is analogous to the previous cases. As $m = \frac{r+1}{3}$ and $n = \frac{s+1}{3}$ the DFA $C$ has $(r+1)2^{\frac{s+1}{3}} + 3.2^{\frac{s-2}{3}} - 5$ transitions. □

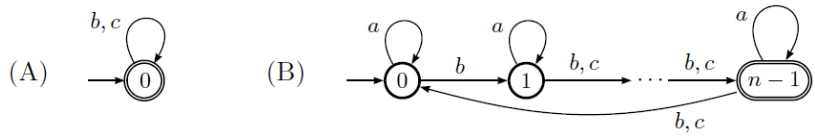



**Fig. 4.** DFA $A$ with 1 state and DFA $B$ with $n$ states

**Theorem 4.** *For any integer $n \geq 2$, exist a 1-state DFA $A$ with 2 transitions and an $n$-state DFA $B$ with $s = 3n-1$ transitions such that any DFA accepting $\mathcal{L}(A)\mathcal{L}(B)$ has, at least, $2^{n+1} - 2^{n-1} - 1$ states and $3(2^{\frac{s+4}{3}} - 2^{\frac{s-2}{3}}) - 4$ transitions.*

*Proof.* Let $A = (Q, \Sigma, \delta_A, 0, F_A)$ with $Q = \{0\}$, $F_A = \{0\}$, $\delta_A(0, b) = \delta_A(0, c) = 0$; and define $B = (P, \Sigma, \delta_B, 0, F_B)$ with $P = \{0, \ldots, n-1\}$, $F_B = \{n-1\}$, $\delta_B(i, a) = i$ if $0 \leq i \leq n-1$, $\delta_B(i, b) = i + 1 \bmod n$ if $0 \leq i \leq n-1$, and $\delta_B(i, c) = i + 1 \bmod n$, if $1 \leq i \leq n-1$ (see Figure 4). Consider the DFA $C = (R, \Sigma, \delta, 0, F)$, constructed by the previous algorithm, such that $\mathcal{L}(C) = \mathcal{L}(A)\mathcal{L}(B)$. One needs to prove that $C$ is minimal, i.e. all states are reachable from the initial state and are pairwise distinguishable. The DFA $C$ has states $(s, c)$ with $s \in \{-1, 0\}$, $c = \{i_1, \ldots, i_k\}$, $1 \leq k \leq n$, and $i_1 < \cdots < i_k$. There are two kinds of states: final states where $i_k = n-1$; and non-final states where $i_k \neq n-1$. Note that whenever $s = 0$, $i_1 = 0$. Taking this form of the states into account is not difficult to prove that the DFA $C$ is minimal. The proof of the second part of the theorem is similar to the proof of Theorem 3. □

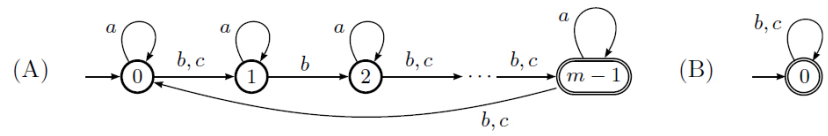



**Fig. 5.** DFA $A$ with $m$ states and DFA $B$ with 1 state

**Theorem 5.** *For any integer $m \geq 2$ exists an $m$-state DFA $A$. with $r = 3m-1$ transitions and an 1-state DFA $B$ with 2 transitions such that any DFA accepting $\mathcal{L}(A)\mathcal{L}(B)$ has at least $2m$ states and $2r$ transitions.*

*Proof (Sketch).* Let $A = (P, \Sigma, \delta_A, 0, F_A)$ with $P = \{0, \ldots, n-1\}$, $F_A = \{m-1\}$, $\delta_A(i, X) = i$, if $0 \leq i \leq m-1$, $\delta_A(i, b) = i + 1 \bmod m$, if $0 \leq i \leq m-1$, $\delta_A(i, c) = i+1 \bmod m$ if $i = 0$ or $2 \leq i \leq m-1$; and $B = (Q, \Sigma, \delta_B, 0, F_B)$ with $Q = \{0\}$, $F_B = \{0\}$, and $\delta_B(0, b) = \delta_B(0, c) = 0$ (see Figure 5). □

## 5 Incomplete Transition Complexity of the Star

In this section we give a tight upper bound for the incomplete transition complexity of the star operation. The incomplete state complexity of star coincides with the one in the complete case.

### 5.1 An Upper Bound

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Let $F_0 = F \setminus \{q_0\}$ and suppose that $l = |F_0| \geq 1$. If $F = \{q_0\}$, then $\mathcal{L}(A)^\star = \mathcal{L}(A)$. The following algorithm obtains the kleene star of a DFA $A$. Let $A' = (Q', \Sigma, \delta', q_0', F')$ be a new DFA where $q_0' \notin Q$ is a new initial state, $Q' = \{q_0'\} \cup \{P \mid P \subseteq (Q \setminus F_0) \wedge P \neq \emptyset\} \cup \{P \mid P \subseteq Q \wedge q_0 \in P \wedge P \cap F_0 \neq \emptyset\}$, $F' = \{q_0'\} \cup \{R \mid R \subseteq Q \wedge R \cap F \neq \emptyset\}$, and for $a \in \Sigma$,

$$\delta'(q_0', a) = \begin{cases} \{\delta(q_0, a)\} & \text{if } \delta(q_0, a)\downarrow \ \wedge\, \delta(q_0, a) \notin F_0, \\ \{\delta(q_0, a), q_0\} & \text{if } \delta(q_0, a)\downarrow \ \wedge\, \delta(q_0, a) \in F_0, \\ \emptyset & \text{if } \delta(q_0, a)\uparrow. \end{cases}$$

and

$$\delta'(R, a) = \begin{cases} \delta(R, a) & \text{if } \delta(R, a) \cap F_0 = \emptyset, \\ \delta(R, a) \cup \{q_0\} & \text{if } \delta(R, a) \cap F_0 \neq \emptyset, \\ \emptyset & \text{if } \delta(R, a) = \emptyset. \end{cases}$$

We can verify that DFA $A'$ recognizes the language $\mathcal{L}(A)^\star$.

The following results present upper bounds for the number of states and transitions for any DFA $A'$ resulting from the algorithm described above.

**Proposition 6.** *For any integer $n \geq 2$ and any $n$-state DFA $A$, any DFA accepting $\mathcal{L}(A)^\star$ needs at least $2^{n-1} + 2^{n-l-1}$ states.*

**Corollary 2.** *The formula in Proposition 6 is maximal when $l = 1$.*

**Proposition 7.** *For any regular language $L$ with $isc(L) = n$, $i_\tau = i_\tau(L)$, and and $\bar{u}_\tau = \bar{u}_\tau(L)$, one has*

$$itc(L^\star) \leq |\Sigma|(2^{n-1} + 2^{n-l-1}) + \sum_{\tau \in \Sigma_i} (i_\tau - 2^{\bar{u}_\tau})$$

*Proof.* Let $A$ be a minimal DFA recognizing a language $L$. Consider the DFA $A'$, constructed with the previous algorithm, such that $\mathcal{L}(A') = \mathcal{L}(A)^\star$. The number of $\tau$-transitions of $A'$ is the summation of:

1. $i_\tau$ $\tau$-transitions leaving the initial state of $A$.
2. The number of sets of $\tau$-transitions of $A$ leaving only non-final states:
   (a) $(2^{t_\tau - l}) - 1$, if $A$ is $\tau$-complete, we have $t_\tau - l$ $\tau$-transitions of this kind, and we remove the empty set.
   (b) $2^{t_\tau - l + u_\tau} - 2^{\bar{u}_\tau}$, if $A$ is $\tau$-incomplete: $t_\tau - l + u_\tau$ of this kind, and we subtract number of sets with only undefined $\tau$-transitions of $A$.
3. The number of sets of $\tau$-transitions of $A$ leaving final and non-final states of $A$. We do not count the transition leaving the initial state of $A$ because, by construction, if a transition of $A'$ contains a transition leaving a final state of $A$ then it also contains the one leaving the initial state of $A$.
   (a) $(2^l - 1)2^{t_\tau - l - 1}$, if $A$ is $\tau$-complete.
   (b) $(2^l - 1)2^{t_\tau - l - 1 + u_\tau}$, if $A$ is $\tau$-incomplete.

Thus, the inequality in the proposition holds. □

### 5.2 Worst-Case Witnesses

Let us present an automaton family for which the upper bounds in Proposition 6 and Proposition 7 are reached. The following automaton family has $\Sigma = \{a, b\}$. Using Myhill-Nerode theorem, it is easy to prove that these automata are minimal.

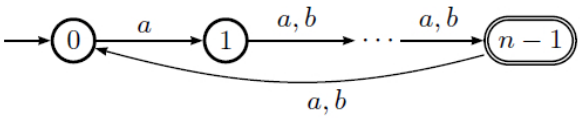



**Fig. 6.** DFA $A$ with $n$ states

**Theorem 6.** *For any integer $n \geq 2$, exists an $n$-state DFA $A$ with $r = 2n - 1$ transitions such that any DFA accepting $\mathcal{L}(A)^\star$ has, at least, $2^{n-1} + 2^{n-2}$ states and $2^{\frac{r+1}{2}} + 2^{\frac{r-1}{2}} - 2$ transitions.*

*Proof.* Let $A = (Q, \Sigma, \delta_A, 0, F_A)$ with $Q = \{0, \ldots, n-1\}$, $F_A = \{n-1\}$, $\delta_A(i, a) = i + 1 \bmod m$ for $0 \leq i \leq n-1$, and $\delta_A(i, b) = i + 1 \bmod m$ for $1 \leq i \leq n-1$ (see Figure 6). Consider the DFA $A'$, constructed with the previous algorithm, such that $\mathcal{L}(A') = (\mathcal{L}(A))^\star$ and which can be proved to be minimal. We only analyse the transition complexity. The DFA $A'$ has:

– $2^{n-1} + 2^{n-2}$ $a$–transitions because $i(a) = 1$, $2^{n-1} - 1$ $a$–transitions which corresponds to case 2 of Proposition 7 and $2^{n-2}$ $a$–transitions which corresponds to case 3 of Proposition 7.

– $2^{n-1}-2+2^{n-2}$ $b$–transitions because it has $2^{n-2+1}-2$ $b$–transitions which corresponds to case 2 of Proposition 7, and $2^{n-3+1}$ $b$–transitions which corresponds to case 3.

As $n=\frac{r+1}{2}$, $A'$ has $2^{\frac{r+1}{2}}+2^{\frac{r-1}{2}}-2$ transitions. □

## 6 Final Remarks

It is known that considering complete DFAs the state complexity of the reversal operation reaches the upper bound $2^n$, where $n$ is the state complexity of the operand language. By the subset construction, a (complete) DFA resulting from the reversal has a state which corresponds to the $\emptyset$, which is a dead state. Therefore, if we remove that state the resulting automaton is not complete and the incomplete state complexity is $2^n-1$. Consequently the transition complexity is $|\Sigma|(2^n-1)$. Note that the worst case of the reversal operation is when the operand is complete.

In this paper we presented tight upper bounds for the incomplete state and incomplete transition complexities for the union, the concatenation, the Kleene star and the reversal of regular languages, with $|\Sigma|\geq 2$. Transition complexity bounds are expressed as functions of several more fine-grained measures of the operands, such as the number of final states, the number of undefined transitions or the number of transitions that leave initial state.

**Table 1.** State Complexity

| Operation | sc | isc | nsc |
|---|---|---|---|
| $L_1\cup L_2$ | $mn$ | $\mathbf{mn+m+n}$ | $m+n+1$ |
| $L_1\cap L_2$ | $mn$ | $mn$ | $mn$ |
| $L^C$ | $n$ | $n+1$ | $2^n$ |
| $L_1L_2$ | $m2^n-f_1 2^{n-1}$ | $\mathbf{(m+1)2^n-f_1 2^{n-1}-1}$ | $m+n$ |
| $L^\star$ | $2^{m-1}+2^{m-l-1}$ | $\mathbf{2^{m-1}+2^{m-l-1}}$ | $m+1$ |
| $L^R$ | $2^m$ | $\mathbf{2^m-1}$ | $m+1$ |

Table 1 and Table 2 summarize some of the results on state complexity and transition complexity of basic operations on regular languages, respectively. In Table 1 we present the state complexity, based on complete DFA (*sc*) [18], DFA (*isc*) (new results here presented and [6]); and NFAs (*nsc*) [7]. Nondeterministic transition complexity (*ntc*) of basic operations on regular languages was studied by Domaratzki and Salomaa [5,12]. They also used refined number of transitions for computing the operational transition complexity. In Table 2, $s(L)$ is the minimal number of transitions leaving the initial state of any transition-minimal NFA $M$ accepting $L$, and $f_{in}(L)$ is the number of transitions entering the final states of any transition-minimal NFA $M$ accepting $L$. The upper bound for the

**Table 2.** Transition Complexity

| Operation | itc | ntc |
|---|---|---|
| $L_1 \cup L_2$ | $\mathbf{itc(L_1)(1+n)} + \mathbf{itc(L_2)(1+m)} - \sum_{\tau\in\mathbf{\Sigma}} \mathbf{itc}_\tau\mathbf{(L_2)itc}_\tau\mathbf{(L_1)}$ | $ntc(L_1) + ntc(L_2) + s(L_1) + s(L_2)$ |
| $L_1 \cap L_2$ | $itc(L_1)itc(L_2)$ | $\sum_{\tau\in\Sigma} ntc_\tau(L_1)ntc_\tau(L_2)$ |
| $L^C$ | $\lvert\Sigma\rvert(itc(L)+2)$ | $\lvert\Sigma\rvert 2^{ntc(L)+1}$<br>$2^{\frac{ntc(L)}{2}-2}-1$ |
| $L_1L_2$ | $\lvert\mathbf{\Sigma}\rvert(\mathbf{m+1})\mathbf{2^n} - \lvert\mathbf{\Sigma_c^{L_2}}\rvert(\mathbf{f}\ \mathbf{2^{n-1}+1}) - \sum_{\tau\in\mathbf{\Sigma_i^{L_2}}} (\mathbf{2^{u_\tau}} + \mathbf{f}\ \mathbf{2^{itc_\tau(L_2)}}) - \sum_{\tau\in\mathbf{\Sigma_{ii}}} \mathbf{\bar{u}_\tau 2^{u_\tau}} - \sum_{\tau\in\mathbf{\Sigma_{ic}}} \mathbf{\bar{u}_\tau}$ | $ntc(L_1) + ntc(L_2) + f_{in}(L_1)$ |
| $L^\star$ | $\lvert\mathbf{\Sigma}\rvert(\mathbf{2^{m-l-1}+2^{m-1}}) + \sum_{\tau\in\mathbf{\Sigma_i}} (\mathbf{i}_\tau - \mathbf{2^{\bar{u}_\tau}})$ | $ntc(L) + f_{in}(L)$ |
| $L^R$ | $\lvert\mathbf{\Sigma}\rvert(\mathbf{2^m-1})$ | $ntc(L) + f(L)$ |

nondeterministic transition complexity of the complementation is not tight, and thus we inscribe the lower and the upper bounds.

In the case of unary languages, if a DFA is not complete it represents a finite language. Thus, the worst-case state complexity of operations occurs when the operand DFAs are complete. For these languages the (incomplete) transition complexity coincide with the (incomplete) state complexity. The study for union and intersection was made by Y. Gao *et al.* [6] and it is similar for the other operations studied in this article.

In future work we plan to extend this study to finite languages and to other regular preserving operations. In order to understand the relevance of these partial transition functions based models, some experimental as well as asymptotic study of the average size of these models must be done.

**Acknowledgements.** This, as many other subjects, was introduced to us by Sheng Yu. He will be forever in our mind.

## References

1. Bordihn, H., Holzer, M., Kutrib, M.: Determination of finite automata accepting subregular languages. Theor. Comput. Sci. 410(35), 3209–3222 (2009)
2. Brzozowski, J.: Complexity in Convex Languages. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 1–15. Springer, Heidelberg (2010)
3. Cassandras, C.G., Lafortune, S.: Introduction to discrete event systems. Springer (2006)
4. Daciuk, J., Weiss, D.: Smaller Representation of Finite State Automata. In: Bouchou-Markhoff, B., Caron, P., Champarnaud, J.-M., Maurel, D. (eds.) CIAA 2011. LNCS, vol. 6807, pp. 118–129. Springer, Heidelberg (2011)

5. Domaratzki, M., Salomaa, K.: Transition complexity of language operations. Theor. Comput. Sci. 387(2), 147–154 (2007)
6. Gao, Y., Salomaa, K., Yu, S.: Transition complexity of incomplete dfas. Fundam. Inform. 110(1-4), 143–158 (2011)
7. Holzer, M., Kutrib, M.: State Complexity of Basic Operations on Nondeterministic Finite Automata. In: Champarnaud, J.-M., Maurel, D. (eds.) CIAA 2002. LNCS, vol. 2608, pp. 148–157. Springer, Heidelberg (2003)
8. Holzer, M., Kutrib, M.: Descriptional and Computational Complexity of Finite Automata. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 23–42. Springer, Heidelberg (2009)
9. Holzer, M., Kutrib, M.: Nondeterministic finite automata - recent results on the descriptional and computational complexity. Int. J. Found. Comput. Sci. 20(4), 563–580 (2009)
10. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley (1979)
11. Owens, S., Reppy, J.H., Turon, A.: Regular-expression derivatives re-examined. J. Funct. Program. 19(2), 173–190 (2009)
12. Salomaa, K.: Descriptional Complexity of Nondeterministic Finite Automata. In: Harju, T., Karhumäki, J., Lepistö, A. (eds.) DLT 2007. LNCS, vol. 4588, pp. 31–35. Springer, Heidelberg (2007)
13. Shallit, J.: A Second Course in Formal Languages and Automata Theory. CUP (2008)
14. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 1, pp. 41–110. Springer (1997)
15. Yu, S.: State complexity: Recent results and open problems. Fundam. Inform. 64(1-4), 471–480 (2005)
16. Yu, S.: On the State Complexity of Combined Operations. In: Ibarra, O.H., Yen, H.-C. (eds.) CIAA 2006. LNCS, vol. 4094, pp. 11–22. Springer, Heidelberg (2006)
17. Yu, S., Gao, Y.: State Complexity Research and Approximation. In: Mauri, G., Leporati, A. (eds.) DLT 2011. LNCS, vol. 6795, pp. 46–57. Springer, Heidelberg (2011)
18. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theor. Comput. Sci. 125(2), 315–328 (1994)

# Algorithms and Almost Tight Results for 3-Colorability of Small Diameter Graphs

George B. Mertzios[1,⋆] and Paul G. Spirakis[2,⋆⋆]

[1] School of Engineering and Computing Sciences, Durham University, UK
george.mertzios@durham.ac.uk
[2] Computer Technology Institute and University of Patras, Greece
spirakis@cti.gr

**Abstract.** The 3-coloring problem is well known to be NP-complete. It is also well known that it remains NP-complete when the input is restricted to graphs with diameter 4. Moreover, assuming the Exponential Time Hypothesis (ETH), 3-coloring can not be solved in time $2^{o(n)}$ on graphs with $n$ vertices and diameter at most 4. In spite of the extensive studies of the 3-coloring problem with respect to several basic parameters, the complexity status of this problem on graphs with small diameter, i.e. with diameter at most 2, or at most 3, has been a longstanding and challenging open question. In this paper we investigate graphs with small diameter. For graphs with diameter at most 2, we provide the first subexponential algorithm for 3-coloring, with complexity $2^{O(\sqrt{n \log n})}$. Furthermore we present a subclass of graphs with diameter 2 that admits a polynomial algorithm for 3-coloring. For graphs with diameter at most 3, we establish the complexity of 3-coloring, even for the case of triangle-free graphs. Namely we prove that for every $\varepsilon \in [0, 1)$, 3-coloring is NP-complete on triangle-free graphs of diameter 3 and radius 2 with $n$ vertices and minimum degree $\delta = \Theta(n^{\varepsilon})$. Moreover, assuming ETH, we use three different amplification techniques of our hardness results, in order to obtain for every $\varepsilon \in [0, 1)$ subexponential asymptotic lower bounds for the complexity of 3-coloring on triangle-free graphs with diameter 3 and minimum degree $\delta = \Theta(n^{\varepsilon})$. Finally, we provide a 3-coloring algorithm with running time $2^{O(\min\{\delta\Delta,\ \frac{n}{\delta}\log\delta\})}$ for arbitrary graphs with diameter 3, where $n$ is the number of vertices and $\delta$ (resp. $\Delta$) is the minimum (resp. maximum) degree of the input graph. To the best of our knowledge, this algorithm is the first subexponential algorithm for graphs with $\delta = \omega(1)$ and for graphs with $\delta = O(1)$ and $\Delta = o(n)$. Due to the above lower bounds of the complexity of 3-coloring, the running time of this algorithm is asymptotically almost tight when the minimum degree of the input graph is $\delta = \Theta(n^{\varepsilon})$, where $\varepsilon \in [\frac{1}{2}, 1)$.

**Keywords:** 3-coloring, graph diameter, graph radius, subexponential algorithm, NP-complete, exponential time hypothesis.

⋆ Partially supported by EPSRC Grant EP/G043434/1.
⋆⋆ Partially supported by the ERC EU Project RIMACO and by the EU IP FET Project MULTIPLEX.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 332–343, 2013.

## 1 Introduction

A *proper k-coloring* (or *k-coloring*) of a graph $G$ is an assignment of $k$ different colors to the vertices of $G$, such that no two adjacent vertices receive the same color. That is, a $k$-coloring is a partition of the vertices of $G$ into $k$ independent sets. The corresponding *k-coloring problem* is the problem of deciding whether a given graph $G$ admits a $k$-coloring of its vertices, and to compute one if it exists. Furthermore, the minimum number $k$ of colors for which there exists a $k$-coloring is denoted by $\chi(G)$ and is termed the *chromatic number* of $G$. The *minimum coloring problem* is to compute the chromatic number of a given graph $G$, and to compute a $\chi(G)$-coloring of $G$ if one exists.

One of the most well known complexity results is that the $k$-coloring problem is NP-complete for every $k \geq 3$, while it can be solved in polynomial time for $k = 2$ [10]. Therefore, since graph coloring has numerous applications besides its theoretical interest, there has been considerable interest in studying how several graph parameters affect the tractability of the $k$-coloring problem, where $k \geq 3$. In view of this, the complexity status of the coloring problem has been established for many graph classes. It has been proved that 3-coloring remains NP-complete even when the input graph is a line graph [13], a triangle-free graph with maximum degree 4 [18], or a planar graph with maximum degree 4 [10].

On the positive side, one of the most famous result in this context has been that the minimum coloring problem can be solved in polynomial time for perfect graphs using the ellipsoid method [11]. Furthermore, polynomial algorithms for 3-coloring have been also presented for classes of non-perfect graphs, such as AT-free graphs [23] and $P_6$-free graphs [22] (i.e. graphs that do not contain any path on 6 vertices as an induced subgraph). Furthermore, although the minimum coloring problem is NP-complete on $P_5$-free graphs, the $k$-coloring problem is polynomial on these graphs for every fixed $k$ [12]. Courcelle's celebrated theorem states that every problem definable in Monadic Second-Order logic (MSO) can be solved in linear time on graphs with bounded treewidth [8], and thus also the coloring problem can be solved in linear time on such graphs.

For the cases where 3-coloring is NP-complete, considerable attention has been given to devise exact algorithms that are faster than the brute-force algorithm (see e.g. the recent book [9]). In this context, asymptotic lower bounds of the time complexity have been provided for the main NP-complete problems, based on the *Exponential Time Hypothesis (ETH)* [14,15]. ETH states that there exists no deterministic algorithm that solves the 3SAT problem in time $2^{o(n)}$, given a boolean formula with $n$ variables. In particular, assuming ETH, 3-coloring can not be solved in time $2^{o(n)}$ on graphs with $n$ vertices, even when the input is restricted to graphs with diameter 4 and radius 2 (see [17,21]). Therefore, since it is assumed that no subexponential $2^{o(n)}$ time algorithms exist for 3-coloring, most attention has been given to decrease the multiplicative factor of $n$ in the exponent of the running time of exact exponential algorithms, see e.g. [4,9,20].

One of the most central notions in a graph is the distance between two vertices, which is the basis of the definition of other important parameters, such as the diameter, the eccentricity, and the radius of a graph. For these graph parameters,

it is known that 3-coloring is NP-complete on graphs with diameter at most 4 (see e.g. the standard proof of [21]). Furthermore, it is straightforward to check that $k$-coloring is NP-complete for graphs with diameter at most 2, for every $k \geq 4$: we can reduce 3-coloring on arbitrary graphs to 4-coloring on graphs with diameter 2, just by introducing to an arbitrary graph a new vertex that is adjacent to all others.

In contrast, in spite of the extensive studies of the 3-coloring problem with respect to several basic parameters, the complexity status of this problem on graphs with small diameter, i.e. with diameter at most 2 or at most 3, has been a longstanding and challenging open question, see e.g. [5,7,16]. The complexity status of 3-coloring is open also for triangle-free graphs of diameter 2 and of diameter 3. It is worth mentioning here that a graph is triangle-free and of diameter 2 if and only if it is a maximal triangle free graph. Moreover, it is known that 3-coloring is NP-complete for triangle-free graphs [18], however it is not known whether this reduction can be extended to maximal triangle free graphs. Another interesting result is that almost all graphs have diameter 2 [6]; however, this result can not be used in order to establish the complexity of 3-coloring for graphs with diameter 2.

**Our Contribution.** In this paper we provide subexponential algorithms and hardness results for the 3-coloring problem on graphs with low diameter, i.e. with diameter 2 and 3. As a preprocessing step, we first present two reduction rules that we apply to an arbitrary graph $G$, such that the resulting graph $G'$ is 3-colorable if and only $G$ is 3-colorable. We call the resulting graph *irreducible* with respect to these two reduction rules. We use these reduction rules to reduce the size of the given graph and to simplify the algorithms that we present.

For graphs with diameter at most 2, we first provide a subexponential algorithm for 3-coloring with running time $2^{O(\min\{\delta, \frac{n}{\delta}\log\delta\})}$, where $n$ is the number of vertices and $\delta$ is the minimum degree of the input graph. This algorithm is simple and has worst-case running time $2^{O(\sqrt{n\log n})}$, which is asymptotically the same as the currently best known time complexity of the graph isomorphism problem [3]. To the best of our knowledge, this algorithm is the first subexponential algorithm for graphs with diameter 2. We demonstrate that this is indeed the worst-case of our algorithm by providing, for every $n \geq 1$, a 3-colorable graph $G_n = (V_n, E_n)$ with $\Theta(n)$ vertices, such that $G_n$ has diameter 2 and both its minimum degree and the size of a minimum dominating set is $\Theta(\sqrt{n})$. In addition, this graph is triangle-free and irreducible with respect to the above two reduction rules. Finally, we present a subclass of graphs with diameter 2, called *locally decomposable* graphs, which admits a polynomial algorithm for 3-coloring. In particular, we prove that whenever an irreducible graph $G$ with diameter 2 has at least one vertex $v$ such that $G - N(v) - \{v\}$ is disconnected, then 3-coloring on $G$ can be decided in polynomial time.

For graphs with diameter at most 3, we establish the complexity of deciding 3-coloring, even for the case of triangle-free graphs. Namely we prove that 3-coloring is NP-complete on irreducible and triangle-free graphs with diameter 3 and radius 2, by providing a reduction from 3SAT. In addition, we provide

a 3-coloring algorithm with running time $2^{O(\min\{\delta\Delta,\ \frac{n}{\delta}\log\delta\})}$ for arbitrary graphs with diameter 3, where $n$ is the number of vertices and $\delta$ (resp. $\Delta$) is the minimum (resp. maximum) degree of the input graph. To the best of our knowledge, this algorithm is the first subexponential algorithm for graphs with $\delta = \omega(1)$ and for graphs with $\delta = O(1)$ and $\Delta = o(n)$. Table 1 summarizes the current state of the art of the complexity of $k$-coloring, as well as our algorithmic and NP-completeness results.

**Table 1.** Current state of the art and our algorithmic and NP-completeness results for $k$-coloring on graphs with diameter $diam(G)$. Our results are indicated by an asterisk.

| $k \setminus diam(G)$ | 2 | 3 | $\geq 4$ |
|---|---|---|---|
| 3 | (∗) $2^{O(\min\{\delta, \frac{n}{\delta}\log\delta\})}$ time algorithm<br>(∗) polynomial alg. for locally decomposable graphs | (∗) NP-complete for min. degree $\delta = \Theta(n^\varepsilon)$, for every $\varepsilon \in [0,1)$, even if $rad(G) = 2$ and $G$ is triangle-free<br>(∗) $2^{O(\min\{\delta\Delta,\ \frac{n}{\delta}\log\delta\})}$ time algorithm | NP-complete [21], no $2^{o(n)}$ algorithm |
| $\geq 4$ | NP-complete | NP-complete | NP-complete |

Furthermore, we provide three different amplification techniques that extend our hardness results for graphs with diameter 3. In particular, we first show that 3-coloring is NP-complete on irreducible and triangle-free graphs $G$ of diameter 3 and radius 2 with $n$ vertices and minimum degree $\delta(G) = \Theta(n^\varepsilon)$, for every $\varepsilon \in [\frac{1}{2}, 1)$ and that, for such graphs, there exists no algorithm for 3-coloring with running time $2^{o(\frac{n}{\delta})} = 2^{o(n^{1-\varepsilon})}$, assuming ETH. This lower bound is asymptotically almost tight, due to our above algorithm with running time $2^{O(\frac{n}{\delta}\log\delta)}$, which is subexponential when $\delta(G) = \Theta(n^\varepsilon)$ for some $\varepsilon \in [\frac{1}{2}, 1)$. With our second amplification technique, we show that 3-coloring remains NP-complete also on irreducible and triangle-free graphs $G$ of diameter 3 and radius 2 with $n$ vertices and minimum degree $\delta(G) = \Theta(n^\varepsilon)$, for every $\varepsilon \in [0, \frac{1}{2})$. Moreover, we prove that for such graphs, when $\varepsilon \in [0, \frac{1}{3})$, there exists no algorithm for 3-coloring with running time $2^{o(\sqrt{\frac{n}{\delta}})} = 2^{o(n^{(\frac{1-\varepsilon}{2})})}$, assuming ETH. Finally, with our third amplification technique, we prove that for such graphs, when $\varepsilon \in [\frac{1}{3}, \frac{1}{2})$, there exists no algorithm for 3-coloring with running time $2^{o(\delta)} = 2^{o(n^\varepsilon)}$, assuming ETH. Table 2 summarizes our lower time complexity bounds for 3-coloring on irreducible and triangle-free graphs with diameter 3 and radius 2, parameterized by their minimum degree $\delta$.

**Organization of the Paper.** We provide in Section 2 the necessary notation and terminology, as well as our two reduction rules and the notion of an irreducible graph. In Sections 3 and 4 we present our results for graphs with diameter 2 and 3, respectively. Detailed proofs have been omitted due to space limitations; a full version can be found in [19].

**Table 2.** Our lower time complexity bounds for deciding 3-coloring on irreducible and triangle-free graphs $G$ with $n$ vertices, diameter 3, radius 2, and minimum degree $\delta(G) = \Theta(n^\varepsilon)$, where $\varepsilon \in [0, 1)$, assuming ETH. The lower bound for $\varepsilon \in [\frac{1}{2}, 1)$ is asymptotically almost tight, as there exists an algorithm for arbitrary graphs with diameter 3 with running time $2^{O(\frac{n}{\delta}\log\delta)} = 2^{O(n^{1-\varepsilon}\log n)}$ by Theorem 4.

| $\delta(G) = \Theta(n^\varepsilon)$: | $0 \leq \varepsilon < \frac{1}{3}$ | $\frac{1}{3} \leq \varepsilon < \frac{1}{2}$ | $\frac{1}{2} \leq \varepsilon < 1$ |
|---|---|---|---|
| Lower time complexity bound: | no $2^{o(n^{(\frac{1-\varepsilon}{2})})}$ time algorithm | no $2^{o(n^\varepsilon)}$ time algorithm | no $2^{o(n^{1-\varepsilon})}$ time algorithm |

## 2 Preliminaries and Notation

In this section we provide some notation and terminology, as well as two reduction (or "cleaning") rules that can be applied to an arbitrary graph $G$. Throughout the article, we assume that any given graph $G$ of low diameter is *irreducible* with respect to these two reduction rules, i.e. that these reduction rules have been iteratively applied to $G$ until they can not be applied any more. Note that the iterative application of these reduction rules on a graph with $n$ vertices can be done in time polynomial in $n$.

**Notation.** We consider in this article simple undirected graphs with no loops or multiple edges. In a graph $G$, the edge between vertices $u$ and $v$ is denoted by $uv$. Given a graph $G = (V, E)$ and a vertex $u \in V$, denote by $N(u) = \{v \in V : uv \in E\}$ the set of neighbors (or the *open neighborhood*) of $u$ and by $N[u] = N(u) \cup \{u\}$ the *closed neighborhood* of $u$. Whenever the graph $G$ is not clear from the context, we will write $N_G(u)$ and $N_G[u]$, respectively. Denote by $\deg(u) = |N(u)|$ the *degree* of $u$ in $G$ and by $\delta(G) = \min\{\deg(u) : u \in V\}$ the *minimum degree* of $G$. Let $u$ and $v$ be two non-adjacent vertices of $G$. Then, $u$ and $v$ are called (false) *twins* if they have the same set of neighbors, i.e. if $N(u) = N(v)$. Furthermore, we call the vertices $u$ and $v$ *siblings* if $N(u) \subseteq N(v)$ or $N(v) \subseteq N(u)$; note that two twins are always siblings.

Given a graph $G = (V, E)$ and two vertices $u, v \in V$, we denote by $d(u, v)$ the *distance* of $u$ and $v$, i.e. the length of a shortest path between $u$ and $v$ in $G$. Furthermore, we denote by $diam(G) = \max\{d(u, v) : u, v \in V\}$ the *diameter* of $G$ and by $rad(G) = \min_{u \in V}\{\max\{d(u, v) : v \in V\}\}$ the *radius* of $G$. Given a subset $S \subseteq V$, $G[S]$ denotes the *induced* subgraph of $G$ on the vertices in $S$. We denote for simplicity by $G - S$ the induced subgraph $G[V \setminus S]$ of $G$. A complete graph (i.e. clique) with $t$ vertices is denoted by $K_t$. A graph $G$ that contains no $K_t$ as an induced subgraph is called *$K_t$-free*. Furthermore, a subset $D \subseteq V$ is a *dominating set* of $G$ if every vertex of $V \setminus D$ has at least one neighbor in $D$. For simplicity, we refer in the remainder of the article to a proper $k$-coloring of a graph $G$ just as a *$k$-coloring* of $G$. Throughout the article we perform several times the *merging* operation of two (or more) independent vertices, which is defined as follows: we *merge* the independent vertices $u_1, u_2, \ldots, u_t$ when we replace them by a new vertex $u_0$ with $N(u_0) = \cup_{i=1}^{k} N(u_i)$.

Observe that, whenever a graph $G$ contains a clique $K_4$ with four vertices as an induced subgraph, then $G$ is not 3-colorable. Furthermore, we can check easily in polynomial time (e.g. with brute-force) whether a given graph $G$ contains a $K_4$. Therefore we assume in the following that all given graphs are $K_4$-free. Furthermore, since a graph is 3-colorable if and only if all its connected components are 3-colorable, we assume in the following that all given graphs are connected. In order to present our two reduction rules of an arbitrary $K_4$-free graph $G$, recall first that the *diamond* graph is a graph with 4 vertices and 5 edges, i.e. it consists of a $K_4$ without one edge. Suppose that four vertices $u_1, u_2, u_3, u_4$ of a given graph $G = (V, E)$ induce a diamond graph, and assume without loss of generality that $u_1u_2 \notin E$. Then, it is easy to see that in any 3-coloring of $G$ (if such exists), $u_1$ and $u_2$ obtain necessarily the same color. Therefore we can merge $u_1$ and $u_2$ into one vertex, as the next reduction rule states, and the resulting graph is 3-colorable if and only if $G$ is 3-colorable.

**Reduction Rule 1 (diamond elimination).** *Let $G = (V, E)$ be a $K_4$-free graph. If the quadruple $\{u_1, u_2, u_3, u_4\}$ of vertices in $G$ induces a diamond graph, where $u_1u_2 \notin E$, then merge vertices $u_1$ and $u_2$.*

Note that, after performing a diamond elimination in a $K_4$-free graph $G$, we may introduce a new $K_4$ in the resulting graph. Suppose now that a graph $G$ has a pair of siblings $u$ and $v$ and assume without loss of generality that $N(u) \subseteq N(v)$. Then, we can extend any proper 3-coloring of $G - \{u\}$ (if such exists) to a proper 3-coloring of $G$ by assigning to $u$ the same color as $v$. Therefore, we can remove vertex $u$ from $G$, as the next reduction rule states, and the resulting graph $G - \{u\}$ is 3-colorable if and only if $G$ is 3-colorable.

**Reduction Rule 2 (siblings elimination).** *Let $G = (V, E)$ be a $K_4$-free graph and $u, v \in V$, such that $N(u) \subseteq N(v)$. Then remove $u$ from $G$.*

**Definition 1.** *Let $G = (V, E)$ be a $K_4$-free graph. If neither Reduction Rule 1 nor Reduction Rule 2 can be applied to $G$, then $G$ is* irreducible.

Due to Definition 1, a $K_4$-free graph is irreducible if and only if it is diamond-free and siblings-free. Given a $K_4$-free graph $G$ with $n$ vertices, clearly we can iteratively execute Reduction Rules 1 and 2 in time polynomial on $n$, until we either find a $K_4$ or none of the Reduction Rules 1 and 2 can be further applied. If we find a $K_4$, then clearly the initial graph $G$ is not 3-colorable. Otherwise, we transform $G$ in polynomial time into an irreducible ($K_4$-free) graph $G'$ of smaller or equal size, such that $G'$ is 3-colorable if and only if $G$ is 3-colorable.

**Observation 1.** *Let $G = (V, E)$ be a connected $K_4$-free graph and $G' = (V', E')$ be the irreducible graph obtained from $G$. If $G'$ has more than two vertices, then $\delta(G') \geq 2$, $diam(G') \leq diam(G)$, $rad(G') \leq rad(G)$, and $G'$ is 3-colorable if and only if $G$ is 3-colorable. Moreover, for every $u \in V'$, $N_{G'}(u)$ induces in $G'$ a graph with maximum degree 1.*

## 3 Algorithms for 3-Coloring on Graphs with Diameter 2

In this section we present our results on graphs with diameter 2. In particular, we provide in Section 3.1 our subexponential algorithm for 3-coloring on such graphs. We then provide, for every $n$, an example of an irreducible and triangle-free graph $G_n$ with $\Theta(n)$ vertices and diameter 2, which is 3-colorable, has minimum dominating set of size $\Theta(\sqrt{n})$, and its minimum degree is $\delta(G_n) = \Theta(\sqrt{n})$. Furthermore, we provide in Section 3.2 our polynomial algorithm for irreducible graphs $G$ with diameter 2, which have at least one vertex $v$, such that $G - N[v]$ is disconnected.

### 3.1 An $2^{O(\sqrt{n \log n})}$-Time Algorithm for Any Graph with Diameter 2

We first provide in the next lemma a well known algorithm that decides the 3-coloring problem on an arbitrary graph $G$, using a dominating set (DS) of $G$.

**Lemma 1 (the DS-approach).** *Let $G = (V, E)$ be a graph and $D \subseteq V$ be a dominating set of $G$. Then, the 3-coloring problem can be decided in $O^*(3^{|D|})$ time on $G$.*

In an arbitrary graph $G$ with $n$ vertices and minimum degree $\delta$, it is well known how to construct in polynomial time a dominating set $D$ with cardinality $|D| \leq n\frac{1+\ln(\delta+1)}{\delta+1}$ [2] (see also [1]). On the other hand, in a graph with diameter 2, the neighborhood of every vertex is a dominating set. Thus we can use Lemma 1 to provide in the next theorem an improved 3-coloring algorithm for the case of graphs with diameter 2.

**Theorem 1.** *Let $G = (V, E)$ be an irreducible graph with $n$ vertices. Let $diam(G) = 2$ and $\delta$ be the minimum degree of $G$. Then, the 3-coloring problem can be decided in $2^{O(\min\{\delta, \frac{n}{\delta}\log\delta\})}$ time on $G$.*

**Corollary 1.** *Let $G = (V, E)$ be an irreducible graph with $n$ vertices and let $diam(G) = 2$. Then, the 3-coloring problem can be decided in $2^{O(\sqrt{n \log n})}$ time on $G$.*

Given the statements of Lemma 1 and Theorem 1, a question that arises naturally is whether the worst case complexity of the algorithm of Theorem 1 is indeed $2^{O(\sqrt{n \log n})}$ (as given in Corollary 1). That is, do there exist 3-colorable irreducible graphs $G$ with $n$ vertices and $diam(G) = 2$, such that both $\delta(G)$ and the size of the minimum dominating set of $G$ are $\Theta(\sqrt{n \log n})$, or close to this value? We answer this question to the affirmative, thus proving that, in the case of 3-coloring of graphs with diameter 2, our analysis of the DS-approach (cf. Lemma 1 and Theorem 1) is asymptotically almost tight. In particular, we provide in the next theorem for every $n$ an example of an irreducible 3-colorable graph $G_n$ with $\Theta(n)$ vertices and $diam(G_n) = 2$, such that both $\delta(G_n)$ and the size of the minimum dominating set of $G$ are $\Theta(\sqrt{n})$. In addition, each of these graphs $G_n$ is triangle-free, as the next theorem states. The construction of

the graphs $G_n$ is based on a suitable and interesting matrix arrangement of the vertices of $G_n$.

**Theorem 2.** *Let $n \geq 1$. Then there exists an* irreducible *and* triangle-free 3-colorable *graph $G_n = (V_n, E_n)$ with $\Theta(n)$ vertices, where $diam(G_n) = 2$ and $\delta(G_n) = \Theta(\sqrt{n})$. Furthermore, the size of the minimum dominating set of $G_n$ is $\Theta(\sqrt{n})$.*

### 3.2 A Tractable Subclass of Graphs with Diameter 2

In this section we present a subclass of graphs with diameter 2, which admits an efficient algorithm for 3-coloring. We first introduce the definition of *locally decomposable* graphs.

**Definition 2.** *Let $G = (V, E)$ be a graph. If there exists a vertex $v_0 \in V$ such that $G - N[v_0]$ is disconnected, then $G$ is a* locally decomposable *graph.*

We prove in Theorem 3 that, given an irreducible and locally decomposable graph $G$ with $diam(G) = 2$, we can decide 3-coloring on $G$ in polynomial time. Note here that there exist instances of $K_4$-free graphs $G$ with diameter 2, for which $G - N[v]$ is connected for every vertex $v$ of $G$, but in the irreducible graph $G'$ obtained by $G$ (by iteratively applying the Reduction Rules 1 and 2), $G' - N_{G'}[v_0]$ becomes disconnected for some vertex $v_0$ of $G'$. That is, $G'$ may be locally decomposable, although $G$ is not. Therefore, if we provide as input to the algorithm of Theorem 3 the irreducible graph $G'$ instead of $G$, this algorithm decides in polynomial time the 3-coloring problem on $G'$ (and thus also on $G$). The crucial idea of this algorithm is that, since $G$ is irreducible and locally decomposable, we can prove in Theorem 3 that every connected component of $G - N[v_0]$ is bipartite, and that in every proper 3-coloring of $G$, all connected components of $G - N[v_0]$ are colored using only two colors.

**Theorem 3.** *Let $G = (V, E)$ be an irreducible graph with $n$ vertices and $diam(G) = 2$. If $G$ is a locally decomposable graph, then we can decide 3-coloring on $G$ in time polynomial on $n$.*

A question that arises now naturally by Theorem 3 is whether there exist any irreducible 3-colorable graph $G = (V, E)$ with $diam(G) = 2$, for which $G - N[v]$ is connected for *every* $v \in V$. A negative answer to this question would imply that we can decide the 3-coloring problem on *any* graph with diameter 2 in polynomial time using the algorithm of Theorem 3. However, the answer to that question is positive: for every $n \geq 1$, the graph $G_n = (V_n, E_n)$ that has been presented in Theorem 2 is irreducible, 3-colorable, has diameter 2, and $G_n - N[v]$ is connected for every $v \in V_n$. Therefore, the algorithm of Theorem 3 can not be used in a trivial way to decide in polynomial time the 3-coloring problem for an arbitrary graph of diameter 2. We leave the tractability of the 3-coloring problem of arbitrary diameter-2 graphs as an open problem.

## 4 Almost Tight Results for Graphs with Diameter 3

### 4.1 An $2^{O(\min\{\delta\Delta,\ \frac{n}{\delta}\log\delta\})}$-Time Algorithm for Any Graph with Diameter 3

In the next theorem we use the DS-approach of Lemma 1 to provide an improved 3-coloring algorithm for the case of graphs with diameter 3.

**Theorem 4.** *Let $G=(V,E)$ be an irreducible graph with $n$ vertices and $diam(G)=3$. Let $\delta$ and $\Delta$ be the minimum and the maximum degree of $G$, respectively. Then, the 3-coloring problem can be decided in $2^{O(\min\{\delta\Delta,\ \frac{n}{\delta}\log\delta\})}$ time on $G$.*

To the best of our knowledge, the algorithm of Theorem 4 is the first subexponential algorithm for graphs with diameter 3, whenever $\delta=\omega(1)$, as well as whenever $\delta=O(1)$ and $\Delta=o(n)$. As we will later prove in Section 4.3, the running time provided in Theorem 4 is asymptotically almost tight whenever $\delta=\Theta(n^{\varepsilon})$, for any $\varepsilon\in[\frac{1}{2},1)$.

### 4.2 The 3-Coloring Problem Is NP-Complete on Graphs with Diameter 3 and Radius 2

In this section we provide a reduction from the 3SAT problem to the 3-coloring problem of triangle-free graphs with diameter 3 and radius 2. Let $\phi$ be a 3-CNF formula with $n$ variables $x_1,x_2,\ldots,x_n$ and $m$ clauses $\alpha_1,\alpha_2,\ldots,\alpha_m$. We can assume in the following without loss of generality that each clause has three distinct literals. We now construct an irreducible and triangle-free graph $H_\phi=(V_\phi,E_\phi)$ with diameter 3 and radius 2, such that $\phi$ is satisfiable if and only if $H_\phi$ is 3-colorable. Before we construct $H_\phi$, we first construct an auxiliary graph $G_{n,m}$ that depends only on the number $n$ of the variables and the number $m$ of the clauses in $\phi$, rather than on $\phi$ itself.

We construct the graph $G_{n,m}=(V_{n,m},E_{n,m})$ as follows. Let $v_0$ be a vertex with $8m$ neighbors $v_1,v_2,\ldots,v_{8m}$, which induce an independent set. Consider also the sets $U=\{u_{i,j}: 1\leq i\leq n+5m, 1\leq j\leq 8m\}$ and $W=\{w_{i,j}: 1\leq i\leq n+5m, 1\leq j\leq 8m\}$ of vertices. Each of these sets has $(n+5m)8m$ vertices. The set $V_{n,m}$ of vertices of $G_{n,m}$ is defined as $V_{n,m}=U\cup W\cup\{v_0,v_1,v_2,\ldots,v_{8m}\}$. That is,$|V_{n,m}|=2\cdot(n+5m)8m+8m+1$, and thus $|V_{n,m}|=\Theta(m^2)$, since $m=\Omega(n)$.

The set $E_{n,m}$ of the edges of $G_{n,m}$ is defined as follows. Define first for every $j\in\{1,2,\ldots,8m\}$ the subsets $U_j=\{u_{1,j},u_{2,j},\ldots,u_{n+5m,j}\}$ and $W_j=\{w_{1,j},w_{2,j},\ldots,w_{n+5m,j}\}$ of $U$ and $W$, respectively. Then define $N(v_j)=\{v_0\}\cup U_j\cup W_j$ for every $j\in\{1,2,\ldots,8m\}$, where $N(v_j)$ denotes the set of neighbors of vertex $v_j$ in $G_{n,m}$. For simplicity of the presentation, we arrange the vertices of $U\cup W$ on a rectangle matrix of size $2(n+5m)\times 8m$, cf. Figure 1(a). In this matrix arrangement, the $(i,j)$th element is vertex $u_{i,j}$ if $i\leq n+5m$, and vertex $w_{i-n-5m,j}$ if $i\geq n+5m+1$. In particular, for every $j\in\{1,2,\ldots,8m\}$, the $j$th column of this matrix contains exactly the vertices of $U_j\cup W_j$, cf. Figure 1(a). Note that, for every $j\in\{1,2,\ldots,8m\}$, vertex $v_j$ is adjacent to all vertices

of the $j$th column of this matrix. Denote now by $\ell_i = \{u_{i,1}, u_{i,2}, \ldots, u_{i,8m}\}$ (resp. $\ell'_i = \{w_{i,1}, w_{i,2}, \ldots, w_{i,8m}\}$) the $i$th (resp. the $(n+5m+i)$th) row of this matrix, cf. Figure 1(a). For every $i \in \{1, 2, \ldots, n+5m\}$, the vertices of $\ell_i$ and of $\ell'_i$ induce two independent sets in $G_{n,m}$. We then add between the vertices of $\ell_i$ and the vertices of $\ell'_i$ all possible edges, except those of $\{u_{i,j}w_{i,j} : 1 \leq j \leq 8m\}$. That is, we add all possible $(8m)^2 - 8m$ edges between the vertices of $\ell_i$ and of $\ell'_i$, such that they induce a complete bipartite graph without a perfect matching between the vertices of $\ell_i$ and of $\ell'_i$, cf. Figure 1(b). Note by the construction of $G_{n,m}$ that both $U$ and $W$ are independent sets in $G_{n,m}$. Furthermore note that the minimum degree in $G_{n,m}$ is $\delta(G_{n,m}) = \Theta(m)$ and the maximum degree is $\Delta(G_{n,m}) = \Theta(n+m)$. Thus, since $m = \Omega(n)$, we have that $\delta(G_{n,m}) = \Delta(G_{n,m}) = \Theta(m)$. The construction of the graph $G_{n,m}$ is illustrated in Figure 1. Moreover, we can prove that $G_{n,m}$ has diameter 3 and radius 2, and furthermore that it is irreducible and triangle-free.

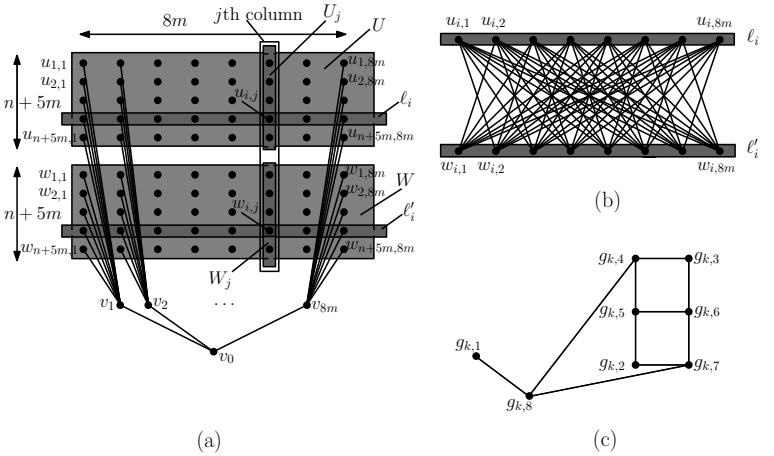



**Fig. 1.** (a) The $2(n+5m) \times 8m$-matrix arrangement of the vertices $U \cup W$ of $G_{n,m}$ and their connections with the vertices $\{v_0, v_1, v_2, \ldots, v_{8m}\}$, (b) the edges between the vertices of the $i$th row $\ell_i$ and the $(n+5m+i)$th row $\ell'_i$ in this matrix, and (c) the gadget with 8 vertices and 10 edges that we associate in $H_\phi$ to the clause $\alpha_k$ of $\phi$, where $1 \leq k \leq m$.

We now construct the graph $H_\phi = (V_\phi, E_\phi)$ from $\phi$ by adding $10m$ edges to $G_{n,m}$ as follows. Let $k \in \{1, 2, \ldots, m\}$ and consider the clause $\alpha_k = (l_{k,1} \vee l_{k,2} \vee l_{k,3})$, where $l_{k,p} \in \{x_{i_{k,p}}, \overline{x_{i_{k,p}}}\}$ for $p \in \{1,2,3\}$ and $i_{k,1}, i_{k,2}, i_{k,3} \in \{1, 2, \ldots, n\}$. For this clause $\alpha_k$, we add on the vertices of $G_{n,m}$ an isomorphic copy of the gadget in Figure 1(c), which has 8 vertices and 10 edges, as follows. Let $p \in \{1,2,3\}$. The literal $l_{k,p}$ corresponds to vertex $g_{k,p}$ of this gadget. If $l_{k,p} = x_{i_{k,p}}$, we set $g_{k,p} = u_{i_{k,p},8k+1-p}$. Otherwise, if $l_{k,p} = \overline{x_{i_{k,p}}}$, we set $g_p = w_{i_{k,p},8k+1-p}$. Furthermore, for $p \in \{4, \ldots, 8\}$, we set $g_{k,p} = u_{n+5k+4-p,8k+1-p}$.

Note that, by construction, the graphs $H_\phi$ and $G_{n,m}$ have the same vertex set, i.e. $V_\phi = V_{n,m}$, and that $E_{n,m} \subset E_\phi$. Therefore $diam(H_\phi) = 3$ and $rad(H_\phi) = 2$, since $diam(G_{n,m}) = 3$ and $rad(G_{n,m}) = 2$. Observe now that every positive literal of $\phi$ is associated to a vertex of $U$, while every negative literal of $\phi$ is associated to a vertex of $W$. In particular, each of the $3m$ literals of $\phi$ corresponds by this construction to a different column in the matrix arrangement of the vertices of $U \cup W$. If a literal of $\phi$ is the variable $x_i$ (resp. the negated variable $\overline{x_i}$), where $1 \leq i \leq n$, then the vertex of $U$ (resp. $W$) that is associated to this literal lies in the $i$th row $\ell_i$ (resp. in the $(n + 5m + i)$th row $\ell'_i$) of the matrix. Moreover, note by the above construction that each of the $8m$ vertices $\{q_{k,1}, q_{k,2}, \ldots, q_{k,8}\}_{k=1}^{m}$ corresponds to a different column in the matrix of the vertices of $U \cup W$. Finally, each of the $5m$ vertices $\{q_{k,4}, q_{k,5}, q_{k,6}, q_{k,7}, q_{k,8}\}_{k=1}^{m}$ corresponds to a different row in the matrix of the vertices of $U$.

**Observation 2.** *The gadget of Figure 1(c) has no proper* 2*-coloring, as it contains an induced cycle of length* 5*.*

**Observation 3.** *Consider the gadget of Figure 1(c). If we assign to vertices* $g_{k,1}, g_{k,2}, g_{k,3}$ *the same color, we can not extend this coloring to a proper* 3*-coloring of the gadget. Furthermore, if we assign to vertices* $g_{k,1}, g_{k,2}, g_{k,3}$ *in total two or three colors, then we can extend this coloring to a proper* 3*-coloring of the gadget.*

**Observation 4.** *For every* $i \in \{1, 2, \ldots, n+5m\}$*, there exists no pair of adjacent vertices in the same row* $\ell_i$ *or* $\ell'_i$ *in* $H_\phi$*.*

**Theorem 5.** *The formula* $\phi$ *is satisfiable if and only if* $H_\phi$ *is* 3*-colorable.*

Moreover we can prove that $H_\phi$ is irreducible and triangle-free, and thus we conclude the main theorem of this section.

**Theorem 6.** *The* 3*-coloring problem is NP-complete on irreducible and triangle-free graphs with diameter* 3 *and radius* 2*.*

### 4.3 Lower Time Complexity Bounds and General NP-Completeness Results

In this section we present our three different amplification techniques of the reduction of Theorem 5. In particular, using these three amplifications we extend for every $\varepsilon \in [0, 1)$ the result of Theorem 6 (by providing both NP-completeness and lower time complexity bounds) to irreducible triangle-free graphs with diameter 3 and radius 2 and minimum degree $\delta = \Theta(n^\varepsilon)$. Our extended NP-completeness results, as well as our lower time complexity bounds are given in Tables 1 and 2. A detailed presentation of the results in this section can be found in [19].

## References


1. Alon, N.: Transversal numbers of uniform hypergraphs. Graphs and Combinatorics 6, 1–4 (1990)
2. Alon, N., Spencer, J.H.: The Probabilistic Method, 3rd edn. John Wiley & Sons (2008)
3. Babai, L., Luks, E.M.: Canonical labeling of graphs. In: Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC), pp. 171–183 (1983)
4. Beigel, R., Eppstein, D.: 3-coloring in time $O(1.3289^n)$. Journal of Algorithms 54(2), 168–204 (2005)
5. Bodirsky, M., Kára, J., Martin, B.: The complexity of surjective homomorphism problems – A survey. CoRR (2011), http://arxiv.org/abs/1104.5257
6. Bollobás, B.: The diameter of random graphs. Transactions of the American Mathematical Society 267(1), 41–52 (1981)
7. Broersma, H., Fomin, F.V., Golovach, P.A., Paulusma, D.: Three complexity results on coloring $P_k$-free graphs. In: Proceedings of the 20th International Workshop on Combinatorial Algorithms, pp. 95–104 (2009)
8. Courcelle, B.: The monadic second-order logic of graphs I: Recognizable sets of finite graphs. Information and Computation 85(1), 12–75 (1990)
9. Fomin, F.V., Kratsch, D.: Exact exponential algorithms. Texts in Theoretical Computer Science. An EATCS Series. Springer (2010)
10. Garey, M.R., Johnson, D.S.: Computers and intractability: A guide to the theory of NP-completeness. W.H. Freeman (1979)
11. Grötschel, M., Lovász, L., Schrijver, A.: Polynomial algorithms for perfect graphs. Topics on Perfect Graphs 88, 325–356 (1984)
12. Hoàng, C.T., Kamiński, M., Lozin, V.V., Sawada, J., Shu, X.: Deciding $k$-colorability of $P_5$-free graphs in polynomial time. Algorithmica 57(1), 74–81 (2010)
13. Holyer, I.: The NP-completeness of edge-coloring. SIAM Journal on Computing 10, 718–720 (1981)
14. Impagliazzo, R., Paturi, R.: On the complexity of $k$-SAT. Journal of Computer and System Sciences 62(2), 367–375 (2001)
15. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? Journal of Computer and System Sciences 63(4), 512–530 (2001)
16. Kamiński, M.: Open problems from algorithmic graph theory. In: 7th Slovenian International Conference on Graph Theory (2011), http://rutcor.rutgers.edu/~mkaminski/AGT/openproblemsAGT.pdf
17. Lokshtanov, D., Marx, D., Saurabh, S.: Lower bounds based on the Exponential Time Hypothesis. Bulletin of the EATCS 84, 41–71 (2011)
18. Maffray, F., Preissmann, M.: On the NP-completeness of the $k$-colorability problem for triangle-free graphs. Discrete Mathematics 162, 313–317 (1996)
19. Mertzios, G.B., Spirakis, P.G.: Algorithms and almost tight results for 3-colorability of small diameter graphs. CoRR (2012), http://arxiv.org/abs/1202.4665
20. Narayanaswamy, N., Subramanian, C.: Dominating set based exact algorithms for 3-coloring. Information Processing Letters 111, 251–255 (2011)
21. Papadimitriou, C.H.: Computational complexity. Addison-Wesley (1994)
22. Randerath, B., Schiermeyer, I.: 3-colorability $\in P$ for $P_6$-free graphs. Discrete Applied Mathematics 136(2-3), 299–313 (2004)
23. Stacho, J.: 3-colouring AT-free graphs in polynomial time. Algorithmica (to appear)

# Approximating the k-Splittable Capacitated Network Design Problem

Ehab Morsy

Division of Mathematical Sciences, Nanyang Technological University,
Singapore 637371, and Department of Mathematics,
Suez Canal University, Ismailia 22541, Egypt
ehabmorsy@gmail.com

**Abstract.** We consider the *k-splittable capacitated network design problem* (kSCND) in a graph $G = (V,E)$ with edge weight $w(e) \geq 0$, $e \in E$. We are given a vertex $s \in V$ designated as a sink, a cable capacity $\lambda > 0$, and a source set $S \subseteq V$ with demand $q(v) \geq 0$, $v \in S$. For any edge $e \in E$, we are allowed to install an integer number $h(e)$ of copies of $e$. The kSCND asks to simultaneously send demand $q(v)$ from each source $v \in S$ along at most $k$ paths to the sink $s$. A set of such paths can pass through a single copy of an edge in $G$ as long as the total demand along the paths does not exceed the cable capacity $\lambda$. The objective is to find a set $\mathcal{P}$ of paths of $G$ that minimizes the installing cost $\sum_{e \in E} h(e)w(e)$. In this paper, we propose a $((k+1)/k + \rho_{ST})$-approximation algorithm to the kSCND, where $\rho_{ST}$ is any approximation ratio achievable for the *Steiner tree problem*.

**Keywords:** approximation algorithm, graph algorithm, routing problems, network optimization.

## 1 Introduction

We study a problem of finding routings from a set of sources to a single sink in a network with an edge installing cost. This problem is a fundamental and economically significant one that arises in hierarchical design of telecommunication networks [3] and transportation networks [8,9]. In telecommunication networks this corresponds to installing transmission facilities such as fiber-optic cables, which represent the edges of the network. In other applications, optical cables may be replaced by pipes, trucks, and so on.

Consider the *capacitated network design problem* (CND), which can be stated as follows. We are given an undirected graph $G$ such that each edge $e \in E(G)$ is weighted by a nonnegative real $w(e)$, a subset $S \subseteq V(G)$ of sources, and a vertex $s \in V(G)$ designated as a sink. Each source $v \in S$ has a nonnegative demand $q(v)$, all of which must be routed to $s$ through a *single* path. A cable with a fixed capacity $\lambda$ is available for installing on the edges of the graph, where installing $i$ copies of the cable on edge $e = (u,v)$ costs $iw(e)$ and provides $i\lambda$ capacity between vertices $u$ and $v$. The CND asks to find a minimum cost installation of cables that provides sufficient capacity to route all of the demands simultaneously to $s$. The problem requires choosing a single path from each source to the sink and deciding the number of cables to be installed on each

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 344–355, 2013.

edge such that all the demands are routed without exceeding the installed cable capacities. Demands of different sources may share the capacity on the installed cables and the capacity installed on an edge has to be at least as much as the total demand routed through this edge.

For the CND problem, Mansour and Peleg [7] gave an $O(\log n)$-approximation algorithm for a graph with $n$ vertices. Salman et al. [8] designed a 7-approximation algorithm for the CND based on a construction from [5]. Afterwards Hassin et al. [4] gave a $(2+\rho_{ST})$-approximation algorithm, where $\rho_{ST}$ is any approximation ratio achievable for the Steiner tree problem. By using of a slight intricate version of this algorithm, they improved the approximation ratio to $(1+\rho_{ST})$ when every source has unit demand. The current best approximation ratio for the Steiner tree problem is at most $\ln 4$ [2]. When all non-sink vertices are sources, the approximation ratio of Hassin et al. [4] becomes 3 for general demands and 2 for unit demands, since the Steiner tree problem in this case is a minimum spanning tree problem.

In this paper, we study a more flexible version of the CND, in which the demand of each source is *splittable* in the sense that it is allowed to be routed to the sink along at most $k$ paths for a specified integer $k$. That is, we have a single cable type with a fixed capacity $\lambda > 0$ for all edges, and we are interested in constructing a demand assignment $\mathcal{D}$ and a set $\mathcal{P}$ of paths such that, for every source $v$, the pair $(\mathcal{P}_v \subseteq \mathcal{P}, \mathcal{D}_v \subseteq \mathcal{D})$ of paths and demand assignment of $v$ identifies the amount of demand of $v$ to be routed to a single sink $s$ through each path in $\mathcal{P}_v$. However, the demand of each source is allowed to be routed to $s$ along at most $k$ paths of $\mathcal{P}$. The cost of installing a copy of an edge $e$ is represented by the weight of $e$. A subset of paths of $\mathcal{P}$ can pass through a single copy of an edge $e$ as long as the total demand sent along these paths does not exceed the cable capacity $\lambda$; any integer number of copies of $e$ are allowed to be installed. The cost of a pair $(\mathcal{P}, \mathcal{D})$ is defined by the minimum cost of installing copies of edges such that the demand of each source can be routed to the sink under the edge capacity constraint, i.e.,

$$cost(\mathcal{P}, \mathcal{D}) = \sum_{e \in E(G)} \lceil q_e/\lambda \rceil w(e),$$

where $q_e$ is the total flow through $e$. The goal is to find a pair $(\mathcal{P}, \mathcal{D})$ of paths and demand assignment that minimizes $cost(\mathcal{P}, \mathcal{D})$. We call this problem, the *k-splittable capacitated Network Design problem* (kSCND). The kSCND can be formally defined as follows, where $\mathbb{R}_+$ denotes the set of nonnegative reals.

**k-Splittable Capacitated Network Design Problem (kSCND)**

**Input:** A connected graph $G$, an edge weight function $w : E(G) \to \mathbb{R}_+$, a cable capacity $\lambda > 0$, a set $S \subseteq V(G)$ of sources, a demand function $q : S \to \mathbb{R}_+$, a sink $s \in V(G)$, and a positive integer $k$.

**Feasible Solution:** For each $v \in S$, a set $\mathcal{P}_v$ of at most $k$ paths of $G$ and a demand assignment $\mathcal{D}_v = \{q_t(v) \mid P_t \in \mathcal{P}_v\}$ such that (i) $\sum_{P_t \in \mathcal{P}_v} q_t(v) = q(v)$, and (ii) For all $P_t \in \mathcal{P}_v$, it holds $\{s, v\} \subseteq V(P_t)$.

**Goal:** Find a feasible solution $(\mathcal{P} = \cup_{v \in S} \mathcal{P}_v, \mathcal{D} = \cup_{v \in S} \mathcal{D}_v)$ that minimizes $cost(\mathcal{P}, \mathcal{D})$.

Similar flow routing problems have been studied before [1], [6]. The single-source unspittable flow problem asks to route demands of a specified set of sinks simultaneously

from a single source in a directed graph with edge capacities and costs at the minimum possible cost without violating edge capacity. The demand of each vertex must be routed along a single path. Baier et al. [1] has studied a relaxation of the problem, in which the demand of each sink is allowed to be routed along $k$ paths.

An instance of the CND is reduced to that of the kSCND with $k = 1$. Moreover, a unit demand instance of the CND [4] can be regarded as an instance of the kSCND with $k \geq \max_{v \in S} q(v)$. In this paper we assume that $2 \leq k < \max_{v \in S} q(v)$, and design a $((k+1)/k + \rho_{ST})$-approximation algorithm to the kSCND. Our result nearly matches with the bounds $(2 + \rho_{ST})$ for the CND and $(1 + \rho_{ST})$ for the CND with unit demand by Hassin et al. [4] by setting $k = 1$ and $\infty$, respectively.

The rest of this paper is organized as follows. Section 2 introduces terminologies on graphs and two lower bounds on the optimal value of the kSCND. Section 3 gives a framework of our approximation algorithm for the kSCND and analyzing its approximation ratio based on a crucial theorem on routings over trees. In Section 4, we give a proof to the theorem by presenting a procedure for realizing a demand assignment in the theorem. Section 5 makes some concluding remarks.

## 2 Preliminaries

This section introduces some notations and definitions. Let $G$ be a simple undirected graph. We denote by $V(G)$ and $E(G)$ the sets of vertices and edges in $G$, respectively. An edge-weighted graph is a pair $(G, w)$ of a graph $G$ and a nonnegative weight function $w : E(G) \to \mathbb{R}_+$. The weight of a shortest path between two vertices $u$ and $v$ in $(G, w)$ is denoted by $d_{(G,w)}(u, v)$. Given a demand function $q : V(G) \to \mathbb{R}_+$ and a subgraph $H$ of $G$ with vertex set $V(H) \subseteq V(G)$, we use $q(H)$ and $q(V(H))$ interchangeably to denote the sum $\sum_{v \in V(H)} q(v)$ of demands of all vertices in $V(H)$.

Let $T$ be a tree. A *subtree* of $T$ is a connected subgraph of $T$. For a subset $X \subseteq V(T)$ of vertices, let $T\langle X \rangle$ denote the minimal subtree of $T$ that contains $X$ (note that $T\langle X \rangle$ is uniquely determined). Now let $T$ be a rooted tree. We denote by $L(T)$ the set of leaves in $T$. For a vertex $v$ in $T$, let $Ch(v)$ and $D(v)$ (or $D_T(v)$) denote the sets of children and descendants of $v$, respectively, where $D(v)$ includes $v$. A *subtree* $T_v$ *rooted* at a vertex $v$ is the subtree induced by $D(v)$, i.e., $T_v = T\langle D(v) \rangle$. For an edge $e = (u, v)$ in a rooted tree $T$, where $u \in Ch(v)$, the subtree induced by $\{v\} \cup D(u)$ is denoted by $T_e$, and is called a *branch* of $T_v$. For a rooted tree $T_v$, the *depth* of a vertex $u$ in $T_v$ is the length (the number of edges) of the path from $v$ to $u$.

The following lower bound has been proved and used to derive approximation algorithms to the CND in [4].

**Lemma 1.** *For an instance $I = (G = (V, E), w, \lambda, S, q, s, k)$ of the kSCND, let $OPT(I)$ be the weight of an optimal solution to $I$, and $T^*$ be a minimum weight tree that spans $S \cup \{s\}$ in $G$. Then*

$$\max\left\{ w(T^*), \frac{1}{\lambda} \sum_{t \in S} q(t) d_{(G,w)}(s, t) \right\} \leq OPT(I),$$

*where $w(T^*)$ is the sum of weights of edges in $T^*$.* □

## 3 Approximation Algorithm to the kSCND

This section describes a framework of our approximation algorithm for the kSCND and then analyzes its approximation ratio.

### 3.1 Overview

We first try to route a sufficiently large amount of demand from a source $v$ to sink $s$ along a shortest path $SP(s,v)$ so that at least $\lambda k/(k+1)$ of the capacity $\lambda$ of each edge is consumed by the routing. We next construct routing for the sources with the remaining small demands along a tree $T$ that spans these sources in such a way that the demand of each source $v$ is routed to at most $k$ (or $k-1$) hub vertices in $T$ and then the total demand collected at each hub vertex $t$ is routed to $s$ along a shorted path $SP(s,t)$. The crucial point in this approach is how to determine appropriate hub vertices to route small demands along $T$ under the capacity constraint. To attain such a routing along $T$, we design a sophisticated procedure, which will be described in Section 4. We here summarize mathematical properties of the procedure as follows.

**Theorem 1.** *Given a tree $T$ rooted at $s$, an edge capacity $\lambda > 0$, a positive integer $k$, a source set $S \subseteq V(T)$, a vertex weight function $d : S \to \mathbb{R}_+$, and a demand function $q : S \to \mathbb{R}_+$ such that $q(v) < \lambda k/(k+1)$ for all $v \in S$, there are a set $\mathcal{H} = \cup_{v \in S} \mathcal{H}_v$ of hub vertices and a demand assignment $\mathcal{D} = \cup_{v \in S} \mathcal{D}_v$, where $\mathcal{D}_v = \{q_t(v) \mid t \in \mathcal{H}_v\}$ and $\sum_{t \in \mathcal{H}_v} q_t(v) = q(v)$ for every $v \in S$, such that:*

(i) *For each $v \in S$, it holds $|\mathcal{H}_v| \leq \lceil q(v)(k+1)/\lambda \rceil$ $(\leq k)$.*
(ii) *It holds $\sum_{v \in S} q(v)d(v) \geq \lambda k/(k+1) \sum_{t \in \mathcal{H} - \{s\}} d(t)$.*
(iii) *When demands of all sources in $S$ are routed to their hub vertices simultaneously, the total amount of these flows on each edge of $T$ is bounded from above by $\lambda$.* □

A proof of the theorem will be given in Section 4. Assuming Theorem 1, we describe our approximation algorithm and analyze its performance guarantee.

### 3.2 Algorithm and Analysis

We first handle the vertices $v$ of demand of at least $\lambda k/(k+1)$. For each source $v \in S$ with $\lambda k/(k+1) \leq q(v) \leq \lambda$, we route the total demand of $v$ through a shortest path to $s$ by installing a copy of each edge in this path. For each source $v \in S$ with $q(v) > \lambda$, we route $\min\{\lfloor q(v)/(\lambda k/(k+1)) \rfloor \lambda, q(v)\}$ of the total demand of $v$ through a shortest path to $s$ by installing $\lfloor q(v)/(\lambda k/(k+1)) \rfloor$ copies of each edge in this path. Note that, in the latter case, the remaining amount of demand at $v$ is less than $\lambda(k-1)/(k+1)$. The basic idea of the rest of the algorithm is to produce a tree $T$ of minimum cost including $s$ and all vertices in $S$ with nonzero remaining demand, find a set $\mathcal{H}$ of hub vertices in $T$, and then assign the remaining demand of each source $v \in S$ to a subset $\mathcal{H}_v$ of at most $\lceil q(v)(k+1)/\lambda \rceil$ hub vertices of $\mathcal{H}$ such that when the demands of all sources are routed to their hub vertices simultaneously, the amount of these flows on each edge of $T$ is at most $\lambda$. Next, for each hub vertex $t \in \mathcal{H}$, we join $t$ to $s$ by installing an appropriate

number of copies of each edge in a shortest path between $s$ and $t$ in $G$. Finally, for each source $v \in S$, we define a set $\mathcal{P}_v$ of at most $k$ paths in $G$ and a demand assignment $\mathcal{D}_v$ that identifies the amount of demand of $q(v)$ to be routed to $s$ through each path in $\mathcal{P}_v$.

**Algorithm.** APPROXKSCND
**Input:** An instance $I = (G, w, \lambda, S, q, k, s)$ of the kSCND.
**Output:** A solution $(\mathcal{P}, \mathcal{D})$ to $I$.

**Step 1.** Let $S_1 = \{v \in S \mid \lambda k/(k+1) \leq q(v) \leq \lambda\}$, $S_2 = \{v \in S \mid q(v) > \lambda\}$, and $S_3 = S - (S_1 \cup S_2)$.
For each $v \in S_1$,
Let $q_v(v) = q(v)$ and $q'(v) = 0$.
Choose a shortest path $SP(s,v)$ between $s$ and $v$ in $(G,w)$ and join $v$ to $s$ by installing a copy of each edge in $SP(s,v)$, and let $P_v = SP(s,v)$.
For each $v \in S_2$,
Let $q_v(v) = \min\{\lfloor q(v)/(\lambda k/(k+1))\rfloor \lambda, q(v)\}$ and $q'(v) = q(v) - q_v(v)$.
Choose a shortest path $SP(s,v)$ between $s$ and $v$ in $(G,w)$ and join $v$ to $s$ by installing $\lfloor q(v)/(\lambda k/(k+1))\rfloor$ copies of each edge in $SP(s,v)$, and let $P_v = SP(s,v)$.
For each $v \in S_3$, let $q'(v) = q(v)$.

**Step 2.** Let $S' = \{v \in S_2 \mid q'(v) > 0\} \cup S_3$.
Compute a Steiner tree $T$ that spans $S' \cup \{s\}$ in $G$.
Regard $T$ as a tree rooted at $s$, and define $d : V(G) \to \mathbb{R}_+$ by setting

$$d(v) := d_{(G,w)}(s,v) \text{ for each vertex } v.$$

**Step 3.** Apply Theorem 1 to $(T, \lambda, S', s, q', d, k)$ to obtain, for each $v \in S'$, a set $\mathcal{H}_v$ of hub vertices and a demand assignment $\mathcal{D}_v = \{q_t(v) \mid t \in \mathcal{H}_v\}$ that satisfy the conditions of the theorem.

**Step 4.** For each $t \in \mathcal{H} = \cup_{v \in S'} \mathcal{H}_v$, choose a shortest path $SP(s,t)$ between $s$ and $t$ in $(G,w)$ and join $t$ to $s$ by installing a copy of each edge in $SP(s,t)$.
For each $v \in S'$ and $t \in \mathcal{H}_v$, let $P_{vt}$ be the path consisting of $SP(s,t)$ and the path between $v$ and $t$ along $T$.

**Step 5.** For each $v \in S_1$, let $\mathcal{P}_v = \{P_v\}$ and $\mathcal{D}_v = \{q_v(v)\}$.
For each $v \in S_2$, let $\mathcal{P}_v = \{P_{vt} \mid t \in \mathcal{H}_v\} \cup \{P_v\}$ and $\mathcal{D}_v = \{q_t(v) \mid t \in \mathcal{H}_v\} \cup \{q_v(v)\}$.
For each $v \in S_3$, let $\mathcal{P}_v = \{P_{vt} \mid t \in \mathcal{H}_v\}$ and $\mathcal{D}_v = \{q_t(v) \mid t \in \mathcal{H}_v\}$.
Output $(\mathcal{P} = \cup_{v \in S} \mathcal{P}_v, \mathcal{D} = \cup_{v \in S} \mathcal{D}_v)$. □

Note that, for each vertex $v \in S_1$, we install one copy of each edge in the shortest path $P_v$ between $v$ and $s$ to send an amount of demand of at least $\lambda k/(k+1)$ to $s$. By Theorem 1(i), the total demand of each vertex $v \in S_3$ is routed to $s$ through at most $k$ paths since $q(v) < \lambda k/(k+1)$. Now, consider a vertex $v \in S_2$. We install $\lfloor q(v)/(\lambda k/(k+1))\rfloor$ copies of each edge in the shortest path $P_v$ between $v$ and $s$ to send $q_v(v) = \min\{\lfloor q(v)/(\lambda k/(k+1))\rfloor \lambda, q(v)\}$ to $s$, and hence the average demand sent through each copy of $P_v$ is at least $\lambda k/(k+1)$. If $q(v) \leq \lfloor q(v)/(\lambda k/(k+1))\rfloor \lambda$, then the total demand of $q(v)$ is sent to $s$ through $P_v$ and hence $q'(v) = 0$. Assume that

$q(v) > \lfloor q(v)/(\lambda k/(k+1))\rfloor \lambda$ holds. Then $q(v) < (\lfloor q(v)/(\lambda k/(k+1))\rfloor + 1)\lambda k/(k+1)$ holds, and hence

$$q'(v) = q(v) - q_v(v) < (\lfloor q(v)/(\lambda k/(k+1))\rfloor + 1)\lambda k/(k+1) - \lfloor q(v)/(\lambda k/(k+1))\rfloor \lambda$$
$$\leq \lambda(k-1)/(k+1),$$

since $\lfloor q(v)/(\lambda k/(k+1))\rfloor \geq 1$. Therefore, by Theorem 1(i), the total demand in $q'(v)$ is routed to $s$ through at most $(k-1)$ paths, and hence the total amount of demand in $q(v)$ is routed to $s$ through at most $1+(k-1) = k$ paths. This discussion and Theorem 1(ii) imply that

$$\sum_{v\in S} q(v)d(v) \geq \frac{\lambda k}{k+1}[\sum_{v\in S_1} d(v) + \sum_{v\in S_2} \lfloor q(v)/(\lambda k/(k+1))\rfloor d(v) + \sum_{t\in\mathcal{H}-\{s\}} d(t)]. \quad (1)$$

By the construction of $\mathcal{P}$ and Theorem 1(iii), we conclude that

$$cost(\mathcal{P},\mathcal{D}) \leq w(T) + \sum_{v\in S_1} d(v) + \sum_{v\in S_2} \lfloor q(v)/(\lambda k/(k+1))\rfloor d(v) + \sum_{t\in\mathcal{H}-\{s\}} d(t).$$

Now, let $OPT(I)$ denote the weight of an optimal solution. For a minimum Steiner tree $T^*$ that spans $S\cup\{s\}$, we have $w(T) \leq \rho_{ST} w(T^*)$ and $w(T^*) \leq OPT(I)$ by Lemma 1. Hence $w(T) \leq \rho_{ST}\cdot OPT(I)$ holds. On the other hand, Equation (1) implies that

$$OPT(I) \geq \sum_{v\in S} q(v)d(v)/\lambda$$
$$\geq \frac{k}{k+1}[\sum_{v\in S_1} d(v) + \sum_{v\in S_2} \lfloor q(v)/(\lambda k/(k+1))\rfloor d(v) + \sum_{t\in\mathcal{H}-\{s\}} d(t)].$$

This completes the proof of the following theorem.

**Theorem 2.** *For an instance $I = (G,w,\lambda,S,q,s,k)$ of the* kSCND*, algorithm* APPROX-kSCND *delivers a $((k+1)/k+\rho_{ST})$-approximate solution $(\mathcal{P},\mathcal{D})$.* □

## 4 Demand Assignment in a Tree

The purpose of this section is to prove Theorem 1. To prove the theorem, we can assume without loss of generality that in a given tree $T$, (i) all terminals are leaves, i.e., $S = L(T)$, by introducing a new edge for each non-leaf terminal, and (ii) $|Ch(v)| = 2$ holds for every non-leaf $v \in V(T)$, i.e., $T$ is a binary tree rooted at $s$, by replicating internal vertices of degree more than 3, so that each copy of a vertex has the same vertex weight with the original vertex and the copies of the same vertex induce a connected subgraph.

We present an algorithm for finding a desired demand assignment of the source set $S$ of a tree. Namely, for each source $v \in S$, we select a set $\mathcal{H}_v$ of at most $\lceil (k+1)q(v)/\lambda\rceil$ hub vertices from $S$ and assign the demand $q(v)$ of each source $v \in S$ to some hub vertices in $\mathcal{H}_v$ such that the routing of the demands of all sources to their hub vertices according to the assignment does not violate edge capacity at any edge in the tree.

The proposed algorithm for proving Theorem 1 goes through two main stages. In the first stage (bottom-up stage), for each source $v \in S$, we find a set $\mathcal{H}_v$ of hub vertices and a demand assignment $\mathcal{D}_v$ that satisfies Conditions (i)-(ii) of the theorem. The second stage (top-down stage) modifies $\mathcal{H}_v$ of some source $v$ by replacing vertices in $\mathcal{H}_v$ with other hub vertices of $\mathcal{H} = \cup_{v \in S} \mathcal{H}_v$ and reassign the demand of $v$ to the modifed hub vertex set so that when the demands of all sources are routed to their hub vertices simultaneously, the amount of these flows on each edge of $T$ is at most $\lambda$ (Theorem 1(iii)).

The algorithm may assign the demand of a source vertex to more than one hub vertex. At any intermediate iteration of the algorithm the demand of any source vertex refers to the current demand of the vertex (unassigned demands at this time).

### 4.1 The First Stage

In this section, we give an iterative algorithm for the first stage that delivers $\mathcal{H}$ and $\mathcal{D}$ which satisfy Theorem 1(i)-(ii). Given a tree $(\widehat{T}, \lambda, S, q, d, k, s)$ defined in Theorem 1, the algorithm called DEMANDASSIGNMENT is outlined as follows. Recall that $q_t(u) \in \mathcal{D}_u$ denotes the amount of demand from a source $u \in S$ assigned to a hub vertex $t \in \mathcal{H}_u$. The demand $q(u)$ of a source $u$ will decrease during an execution of the first stage as part of its demand is assigned to hub vertices; we call a source $u$ *positive* while it has a nonzero remaining demand during an execution of the algorithm. At the begining of each iteration we compute the minimal subtree $T$ of $\widehat{T}$ that spans $s$ and all the positive sources, i.e., $T = \widehat{T}\langle \{u \in S \mid q(u) > 0\} \cup \{s\} \rangle$. For each vertex $u$ during an execution, we use the following notations: $S_u$ denotes the set of positive sources in the descendants of $u$, i.e., $S_u = D_T(u) \cap \{v \in S \mid q(v) > 0\}$, and we choose a positive source $t_u \in S_u$ with minimum weight $d(t_u)$, which is used as a hub vertex of certain vertex $u$ in the first stage. To simplify the presentation and avoid tedious technical discussions we assume that the source $t_u$ of the minimum weight in $S_u$ is unique (at the end of this section we show that our results can be extended to the general case). Let $T_u$ denote the minimal rooted subtree of $T$ that contains $S_u$, i.e., $T_u = T\langle S_u \rangle$.

During an execution of the algorithm, we compute the following data in the current tree $T$: A non-source vertex $v$ which satisfies a certain criteria will be chosen by the algorithm, and $\mathcal{C}$ denotes the set of such chosen vertices during an execution of the algorithm. In particular, the set $\mathcal{C}$ at the begining of the current iteration when a vertex $u$ is chosen is denoted by $\mathcal{C}_u$. For each vertex $u$ in $T$, a non-source vertex $w \in \mathcal{C}_u - \{u\}$ is called an *immediate support* vertex of $u$ if $t_u$ (the source of the minimum weight in $S_u$) is contained in $S_w$, or is called a *support* vertex of $u$ if there is a sequence of vertices $u_0\ (= u), u_1, \ldots, u_j\ (= w) \in \mathcal{C}_u$ such that each $u_{i+1}$, $i = 0, 1, \ldots, j-1$, is an immediate support vertex of $u_i$.

For each vertex $u$ in $T$, let $\kappa_u$ denote the number of the hub vertices $t_w$ of all support vertices $w$ of $u$, and $Q_u$ denote the summation of demands assigned to the hub vertices $t_w$ of all support vertices $w$ of $u$, i.e.,

$$Q_u = \sum \{ \sum_{z \in S_w} q_{t_w}(z) \mid \text{ support vertices } w \text{ of } u \}.$$

By definition, $\kappa_u$ (resp., $Q_u$) can be computed recursively as the summation of $\kappa_w + 1$ (resp., $Q_w + \sum_{z \in S_w} q_{t_w}(z)$) overall immediate support vertices $w \in \mathcal{C}_u$ of $u$.

We need to know these values $\kappa_u$ and $Q_u$ in order to maintain the following conditions at the end of the iteration of the algorithm for each vertex $v \in \mathcal{C}$.

$$\text{any nonzero amount of demand of a positive source } z \in S_v \text{ assigned to } t_v \text{ is } q_{t_v}(z) = q(z) \text{ or } q_{t_v}(z) \geq \lambda/(k+1), \tag{2}$$

$$\text{the total demands } \textstyle\sum_{z \in S_v} q_{t_v}(z) \text{ assigned to } t_v \text{ satisfies } (\kappa_v+1)\lambda k/(k+1) \leq Q_v + \sum_{z \in S_v} q_{t_v}(z) < (\kappa_v+1)\lambda k/(k+1) + \lambda/(k+1), \tag{3}$$

$$Q_u + q(S_u) < (\kappa_u+1)\lambda k/(k+1) \text{ for each vertex } u \in V(T'_v) \text{ for the subtree } T' \text{ of } \widehat{T} \text{ spanning } s \text{ and all positive sources.} \tag{4}$$

To maintain the above conditions (2)-(4), we keep the minimal subtree $T$ of $\widehat{T}$ that spans $s$ and all positive sources in $S$, and choose a new vertex $v$ to be included into $\mathcal{C}$ as a non-source vertex $v$ in $T$ with maximum depth such that $q(S_v) + Q_v$ is at least $(\kappa_v+1)\lambda k/(k+1)$. When we add $v$ to $\mathcal{C}$, we assign an amount of demand of positive sources in $S_v$ to $t_v$ so that the above conditions hold. More formally, for the two children $x_v$ and $y_v$ of $v$ in $T$ where $t_v$ is a descendant of $x_v$, we assign the entire demand $q(x)$ of every positive source $x \in S_v \cap V(T_{x_v})$ to $t_v$, i.e., $q_{t_v}(x) = q(x)$. We then assign the demands of positive sources in $S_v \cap V(T_{y_v})$ to $t_v$. We repeatedly choose a positive source $y \in S_v \cap V(T_{y_v})$ such that, for each vertex $u \in V(T_{y_v})$, the source of the minimum vertex weight $d$ in $T_u$ is chosen as $y$ only when $T_u$ contains no other positive source, and then assign $\min\{q(y), \lambda/(k+1)\}$ of the demand of $y$ to $t_v$ as long as $Q_v + \sum_{z \in S_v} q_{t_v}(z) < (\kappa_v+1)\lambda k/(k+1)$ (a source $y \in S_v \cap V(T_{y_v})$ with $q(y) > \lambda/(k+1)$ may be chosen more than once).

**Algorithm.** DEMANDASSIGNMENT
**Input:** A binary tree $\widehat{T}$ rooted at $s$, a capacity $\lambda$ of each edge, a set $S = L(\widehat{T})$ of sources, a positive integer $k$, a demand function $q : S \to \mathbb{R}_+$ such that $q(u) < \lambda k/(k+1)$, $u \in S$, and a vertex weight function $d : S \to \mathbb{R}_+$.
**Output:** A set $\mathcal{H} = \cup_{u \in S} \mathcal{H}_u$ of hub vertices and a demand assignment $\mathcal{D} = \cup_{u \in S} \mathcal{D}_u$ that satisfy the conditions in Theorem 1(i)-(ii).
**Initialize** $\mathcal{H}_u := \mathcal{D}_u := \emptyset$ and $Q_u := \kappa_u := 0$ for all $u \in S$; $T := \widehat{T}$; $\mathcal{C} := \emptyset$;
1 **while** $q(S) \geq \lambda k/(k+1)$ **do**
2 $T := T\langle\{u \in S \mid q(u) > 0\} \cup \{s\}\rangle$;
3 Choose a non-source vertex $v \in V(T)$ with maximum depth such that $q(S_v) \geq (\kappa_v+1)\lambda k/(k+1) - Q_v$, and let $\mathcal{C} := \mathcal{C} \cup \{v\}$;
/* the start of assignment to hub vertex $t_v$ */
4 Let $x_v$ and $y_v$ be the two children of $v$ in $T$ where $t_v \in V(T_{x_v})$;
5 Let $q_{t_v}(x) := q(x)$ and $q(x) := 0$ for all $x \in S_v \cap V(T_{x_v})$;
6 Initialize $q_{t_v}(y) := 0$ for all $y \in S_v \cap V(T_{y_v})$;
7 **while** $Q_v + \sum_{z \in S_v} q_{t_v}(z) < (\kappa_v+1)\lambda k/(k+1)$ **do**
8 Choose a positive source $y \in V(T_{y_v})$ such that, for each vertex $u \in V(T_{y_v})$, the source of the minimum vertex weight $d$ in $T_u$ is chosen as $y$ only when $T_u$ contains no other positive source;
9 $q_{t_v}(y) := q_{t_v}(y) + \min\{q(y), \lambda/(k+1)\}$;

10 $q(y) := q(y) - \min\{q(y), \lambda/(k+1)\}$
11 **endwhile**;
/* the end of assignment to hub vertex $t_v$ */
12 Let $\mathcal{H}_u := \mathcal{H}_u \cup \{t_v\}$; $\mathcal{D}_u := \mathcal{D}_u \cup \{q_{t_v}(u)\}$ for all $u \in S_v$ with $q_{t_v}(u) > 0$
13 **endwhile**;
/* assignment to $s$ */
14 Let $q_s(u) := q(u)$; $\mathcal{H}_u := \mathcal{H}_u \cup \{s\}$; $\mathcal{D}_u := \mathcal{D}_u \cup \{q_s(u)\}$ for all positive sources $u \in S$.

Algorithm DEMANDASSIGNMENT correctly delivers a desired pair of $\mathcal{H}$ and $\mathcal{D}$ which satisfies Theorem 1(i)-(ii) (the proof is omitted due to space constraints).

We observe that, for each vertex $u$, the data $S_u$, $t_u$, $Q_u$, $\kappa_u$, and the set of support vertices of $u$ may change during the execution of the algorithm since their values are based on the current tree $T$. Throughout the rest of this paper, for each vertex $u \in \mathcal{C}$, we assume that such data refer to their values computed at the time the vertex $u$ is chosen in line 3 of the outer while-loop of the algorithm.

Throughout the execution of Algorithm DEMANDASSIGNMENT, for each vertex $u$ in the current tree $T$, we assumed that the vertex of the minimum weight $d$ in $S_u$ is unique. We can generalize our results to work for the general case as well by fixing an arbitrary vertex among the sources of the minimum weight in $S_u$ as $t_u$ (if there are more than one) throughout the execution of the algorithm as long as it is positive. Moreover, for each descendant $w$ of $u$, if $t_u \in S_w$ and $d(t_u) \leq d(z)$ for all $z \in S_w$, then we let $t_w = t_u$.

### 4.2 The Second Stage

Note that the demand assignment $(\mathcal{H}, \mathcal{D})$ output from Algorithm DEMANDASSIGNMENT may violate Condition (iii) of Theorem 1. As the second stage, we reassign the demand of some sources to a different subset of the same set $\mathcal{H}$ of hub vertices so that Theorem 1(iii) is satisfied. In this section, we design an algorithm for this stage.

Let $(T, \lambda, S, q, d, k, s)$ be a tree defined in Theorem 1, and let $(\mathcal{H}, \mathcal{D})$ and $\mathcal{C}$ denote the demand assignment and the final set of chosen vertices output from Algorithm DEMANDASSIGNMENT. For each vertex $v \in \mathcal{C}$, we define the following data that will be used throughout this section. Let $I_v$ denote the set of immediate support vertices of $v$, and define $\mathcal{F}_v = \{t_u \mid u \in I_v\} \cup \{t_v\}$. Let $\widetilde{S}_v$ denote the set of all sources with nonzero demands that have been assigned to some hub vertex in $\mathcal{F}_v$.

In the next lemma, the total demands $q^v(u) = \sum_{t \in \mathcal{F}_v \cap \mathcal{H}_u} q_t(u)$ of all vertices $u \in \widetilde{S}_v$ (that have been assigned to $\mathcal{F}_v$) will be reassigned to the same set of hub vertices $\mathcal{F}_v$ such that when the total demands $q^v(u)$ of all vertices $u \in \widetilde{S}_v$ are routed to their new hub vertices the flow on each edge of the subtree

$$T^v = T\left\langle (\widetilde{S}_v \cup \{v\}) - \cup_{u \in I_v} V(T_{x_u}) - \cup_{u \in C_v - I_v} S_u \right\rangle$$

is at most $\lambda$.

**Lemma 2.** *The amount of demand $q^v(u)$ of each vertex $u \in \widetilde{S}_v$ can be reassigned to a subset of hub vertices of $\mathcal{F}_v$ such that the resulting demand reassignment $\mathcal{D}^v_u = \{q_t(u) > 0 \mid t \in \mathcal{F}_v\}$ of $q^v(u)$ satisfies:*

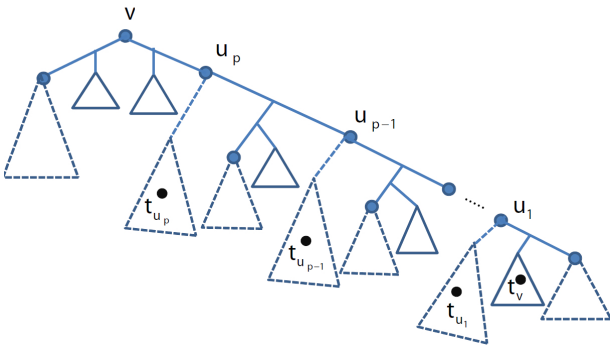



**Fig. 1.** The subtree induced by solid edges form $T^v$ in Lemma 2 ($u_1, u_2, \ldots, u_p$ are the immediate support vertices of $v$)

(i) *For every* $u \in \widetilde{S}_v$ *and* $t \in \mathcal{F}_v$, *if* $q_t(u) > 0$, *then either* $q_t(u) = q^v(u)$, *or* $q_t(u) \geq \lambda/(k+1)$,
(ii) *For every* $u \in I_v$, *the total demand in* $Q_u$ *plus that reassigned to* $t_u$ *is less than* $(\kappa_u + 1)\lambda k/(k+1) + \lambda/(k+1)$, *and*
(iii) *when the total demands* $q^v(u)$ *of all vertices* $u \in \widetilde{S}_v$ *are routed to their hub vertices the flow on each edge of* $T^v$ *is at most* $\lambda$. □

The proof of the above lemma is omitted due to space constraints.

Based on the previous lemma we present the following algorithm that update the demand assignment $(\mathcal{H}, \mathcal{D})$ output from Algorithm DEMANDASSIGNMENT in order to satisfy Theorem 1(iii).

**Algorithm.** DEMANDREASSIGNMENT
**Input:** A tree $(T, \lambda, S, q, d, k, s)$ defined in Theorem 1. The set $\mathcal{C}$ and the demand assignment $(\mathcal{H}' = \cup_{u \in S} \mathcal{H}'_u, \mathcal{D}' = \cup_{u \in S} \mathcal{D}'_u)$ output from Algorithm DEMANDASSIGNMENT.
**Output:** A demand assignment $(\mathcal{H} = \cup_{u \in S} \mathcal{H}_u, \mathcal{D} = \cup_{u \in S} \mathcal{D}_u)$ that satisfies the conditions in Theorem 1.
**Initialize** $\mathcal{H}_u := \mathcal{H}'_u$; $\mathcal{D}_u := \mathcal{D}'_u$ for all $u \in S$; $\mathcal{C}' := \mathcal{C}$.
1 **while** $\mathcal{C}' \neq \emptyset$ **do**
2 Choose the closest vertex $v \in \mathcal{C}'$ to $s$;
3 **if** the flow on an edge of $T^v$ exceeds $\lambda$ when the total demands $q^v(u)$ of all vertices $u \in \widetilde{S}_v$ are routed to their hub vertices **then**
4 Apply Lemma 2 to $(I_v, \widetilde{S}_v)$ to get, for each vertex $u \in \widetilde{S}_v$, a demand reassignment $\mathcal{D}^v_u = \{q_t(u) > 0 \mid t \in \mathcal{F}_v\}$;
5 Let $\mathcal{D}_u := (\mathcal{D}_u - \{q_t(u) \mid t \in \mathcal{F}_v \cap \mathcal{H}'_u\}) \cup \mathcal{D}^v_u$ for all $u \in \widetilde{S}_v$;
6 Let $\mathcal{H}_u := (\mathcal{H}_u - \mathcal{F}_v \cap \mathcal{H}'_u) \cup \{t \in \mathcal{F}_v \mid q_t(u) > 0\}$ for all $u \in \widetilde{S}_v$
7 **endif**;
8 Let $\mathcal{C}' = \mathcal{C}' - \{v\}$
9 **endwhile**.

We show that the demand assignment $(\mathcal{H}, \mathcal{D})$ output from Algorithm DEMANDREASSIGNMENT satisfies the conditions in Theorem 1. Theorem 1(i) follows directly from Lemma 2(i). Note that the set $\mathcal{H}$ of hub vertices output from Algorithm DEMANDASSIGNMENT remains unchanged during the execution of Algorithm DEMANDREASSIGNMENT, and hence Condition (ii) holds for $(\mathcal{H}, \mathcal{D})$. It remains to show that Theorem 1(iii) holds.

**Proof of Theorem 1(iii).** First we show that, at the end of Algorithm DEMANDREASSIGNMENT, when the total demands of all vertices in $S$ are routed to their hub vertices the flow on each edge of the subtree $T^s = T\langle (S \cup \{s\}) - \cup_{v \in \mathcal{C}} S_v \rangle$ is at most $\lambda k/(k+1)$. Note that, after the final iteration of Algorithm DEMANDASSIGNMENT, the total amount of demands of all positive sources that have been assigned to $s$ is less than $\lambda k/(k+1)$. On the other hand, by the construction of DEMANDASSIGNMENT, no demands of any sources in $T - T^s$ (resp., $T^s$) have been assigned to any vertex in $V(T^s) - \{s\}$ (resp., $T - T^s$) during the execution of all iterations of the while-loop. Therefore, at the end of Algorithm DEMANDASSIGNMENT, when the total demands of all vertices in $S$ are routed to their hub vertices the flow on each edge of $T^s$ is less than $\lambda k/(k+1)$. We observe that the flow on edges of $T^s$ remains unchanged during the execution of Algorithm DEMANDREASSIGNMENT, and hence the above bound on the flow on each edge in $T^s$ is maintained at the end of DEMANDREASSIGNMENT.

Now, consider a vertex $v \in \mathcal{C}$ chosen during the execution of an arbitrary iteration of Algorithm DEMANDREASSIGNMENT. Assume that when the total demands $q^v(u)$ of all vertices $u \in \widetilde{S}_v$ are routed to their hub vertices the flow on some edges in $T^v$ exceed $\lambda$. We apply Lemma 2 to $(I_v, \widetilde{S}_v)$ to reassign the total demands $q^v(u)$ of all vertices $u \in \widetilde{S}_v$ to the set $\mathcal{F}_v$ of hub vertices such that when these demands are routed to their new hub vertices the total flow on each edge of the subtree $T^v$ is at most $\lambda$. Define the subtree $T^{y_v} = T\langle S_v \cap V(T_{y_v}) \cup \{v\} - \cup_{u \in C_v} S_u \rangle$. We observe that when the total demands of all vertices in $S$ are routed to their hub vertices the flow on each edge of the subtree $T^v - T^{y_v}$ is from $q^v(u)$, $u \in \widetilde{S}_v$ (which is at most $\lambda$ by Lemma 2), and hence it remains to show that the flow on each edge of the subtree $T^{y_v}$ is at most $\lambda$. Note that, at the time $v$ has been chosen in Algorithm DEMANDASSIGNMENT, the total demand in $S_v \cap V(T_{y_v})$ is less than $\lambda k/(k+1)$. On the other hand, by the construction of Algorithm DEMANDASSIGNMENT, no demands have been assigned to any vertex in $T^{y_v}$ during the execution of all iterations prior to the iteration in which $v$ has been chosen in DEMANDASSIGNMENT. Therefore, we have the following two cases.

In the first case, $v$ is not an immediate support vertex of any other vertex in $\mathcal{C}$. Then there is no demands from $T - T_v$ have been assigned to any vertex in $T^{y_v}$ during the execution of Algorithm DEMANDASSIGNMENT, and hence when the total demands of all vertices in $S$ are routed to their hub vertices the flow on each edge in $T^{y_v}$ is less than $\lambda k/(k+1)$.

In the second case, $v$ is an immediate support vertex of some vertex $v' \in \mathcal{C}$. Clearly, $v \in V(T_{v'})$ and Lemma 2 was applied to $(I_{v'}, \widetilde{S}_{v'})$ in a previous iteration of Algorithm DEMANDREASSIGNMENT, and hence when the total demands of all vertices in $S$ are routed to their hub vertices the flow on each edge of the subtree $T^{y_v}$ is at most $\lambda$.

We observe that, during the execution of the subsequent iterations of Algorithm DEMANDREASSIGNMENT, the flow on each edge of $T^v$ does not change. This completes the proof of Theorem 1(iii). □

## 5 Concluding Remarks

In this paper, we have studied the k-splittable capacitated network design problem (kSCND), a problem of finding a routing from a set of sources to a single sink in a network with an edge installing cost. The kSCND is a relaxation version of the capacitated network design problem (CND), in which the demand of each source can be routed to the sink along at most $k$ paths. We have designed a $((k+1)/k+\rho_{ST})$-approximation algorithm for the kSCND, where $\rho_{ST}$ is any approximation ratio achievable for the Steiner tree problem. A future work may include an interesting version of the kSCND, in which the demand of each vertex $v$ is allowed to be routed to a single sink through at most $k_v$ paths.

## References

1. Baier, G., Kohler, E., Skutella, M.: The k-splittable flow problem. Algorithmica 42, 231–248 (2005)
2. Byrka, J., Grandoni, F., Rothvoß, T., Sanità, L.: An improved LP-based approximation for steiner tree. In: ACM Symposium on the Theory of Computing (STOC), pp. 583–592 (2010)
3. Grandoni, F., Italiano, G.F.: Improved Approximation for Single-Sink Buy-at-Bulk. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 111–120. Springer, Heidelberg (2006)
4. Hassin, R., Ravi, R., Salman, F.S.: Approximation algorithms for a capacitated network design problem. Algorithmica 38, 417–431 (2004)
5. Khuller, S., Raghavachari, B., Young, N.N.: Balancing minimum spanning and shortest path trees. Algorithmica 14, 305–322 (1993)
6. Kolliopoulos, S.G.: Minimum-cost single source 2-splittable flow. Information Processing Letters 94, 15–18 (2005)
7. Mansour, Y., Peleg, D.: An approximation algorithm for minimum-cost network design, Tech. Report Cs94-22, The Weizman Institute of Science, Rehovot (1994); Also Presented at the DIMACS Workshop on Robust Communication Network (1998)
8. Salman, F.S., Cheriyan, J., Ravi, R., Subramanian, S.: Approximating the single-sink link-installation problem in network design. SIAM J. Optim. 11, 595–610 (2000)
9. Zheng, H.-Z., Chu, D.-H., Zhan, D.-C.: Effective algorithm CFL based on Steiner tree and UFL problem. IJCSNS 6(9A), 24–27 (2006)

# Mixed Hypergraphs for Linear-Time Construction of Denser Hashing-Based Data Structures

Michael Rink*

Fakultät für Informatik und Automatisierung, Technische Universität Ilmenau
michael.rink@tu-ilmenau.de

**Abstract.** There are several hashing-based data structures whose space utilization (keys per table cells) directly depends on the edge density threshold for the appearance of a 2-core in some underlying random $k$-uniform hypergraph. We show that by modifying these data structures such that the $k$-uniform hypergraphs are replaced by certain non-uniform hypergraphs their space utilization can be improved. These non-uniform hypergraphs are a mixture of uniform hypergraphs each with a linear number of edges but with different edge sizes. In the case of two different edge sizes we give a solution for the optimal (expected) number of edges of each size such that the 2-core threshold for the resulting mixed hypergraph is maximized. For suitable edge sizes we obtain optimal thresholds for mixed hypergraphs up to 0.920, improving the maximum 2-core threshold for any random $k$-uniform hypergraph, which is about 0.818.

## 1 Introduction

*Motivation: Matchings and Linear Systems.* Consider a random bipartite graph $G$ with $m$ left nodes and $n$ right nodes, where left node $i$ has $k_i$ right neighbors, and the $k_i$'s are small, in particular $\sum_{i=1}^{m} k_i = O(m)$. Let $\boldsymbol{M}$ from $\{0,1\}^{m\times n}$ be the biadjacency matrix of $G$. The following algorithmic problems have several applications: $(i)$ We wish to find in linear time a matching in $G$ that covers all left nodes, or $(ii)$ a solution $\boldsymbol{a}$ of the linear system $\boldsymbol{M}\cdot\boldsymbol{a} = \boldsymbol{v}$, for some vector $\boldsymbol{v}$. A sufficient condition for both is that $\boldsymbol{M}$ can be transformed into row echelon form by row and column exchanges only (no addition or multiplication required). The question whether such a transformation is possible is conveniently expressed in terms of the hypergraph $H$ that corresponds to $G$ and $\boldsymbol{M} = (M_{i,j})_{i,j}$. This hypergraph has node set $\{1,2,\ldots,n\}$ and $m$ edges $e_i = \{j \mid M_{i,j} = 1\}$, $1 \le i \le m$. The matrix can be transformed if and only if $H$ has "an empty 2-core" in the sense of the following definition.

*The* 2*-core.* The 2-core of a hypergraph $H$ is the largest induced sub-hypergraph (possibly empty) that has minimum degree at least 2. It can be obtained via a simple peeling procedure (Algorithm 1) that successively removes nodes of degree 1 together with their incident edge. If $H$ is given as a list of its edges the

* Research supported by DFG grant DI 412/10-2.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 356–368, 2013.

2-core and a peeling order of edges and nodes can be found in linear time; and if the 2-core is empty a matching in $G$ as well as a solution of $\boldsymbol{M} \cdot \boldsymbol{a} = \boldsymbol{v}$ can be determined in linear time from the peeling order. The central question considered in this paper is how to choose $k_1, k_2, \ldots, k_m$ so as to get a high probability for a random hypergraph being completely peelable, i.e., having an empty 2-core. The starting point for our considerations are random $k$-uniform hypergraphs.

**Algorithm 1.** Peeling

**In**: Hypergraph $H$ **Out**: 2-core of $H$
**while** *$H$ has a node $v$ of degree $\leq 1$* **do**
  **if** *$v$ is incident to an edge $e$* **then**
    remove $e$ from $H$
  remove $v$ from $H$
**return** $H$

*Uniform Hypergraphs.* Let $H^k_{n,p}$ be a random hypergraph with $n$ nodes where each of the possible $\binom{n}{k}$ edges of size $k$ is present with probability $p$ independent of the other edges. In the case that the expected number of edges equals $c \cdot n$ for some constant $c > 0$, the following theorem (conjectured e.g. in [16], rigorously proved in [18] and independently in [12]) gives the threshold for the appearance of a 2-core in $H^k_{n,p}$. Let

$$t(\lambda, k) = \frac{\lambda}{k \cdot \left( \Pr\left( \mathrm{Po}\left[\lambda\right] \geq 1 \right) \right)^{k-1}} , \tag{1}$$

where $\mathrm{Po}[\lambda]$ denotes a Poisson random variable with mean $\lambda$.

**Theorem 1 ([18, Theorem 1.2]).** *Let $k \geq 3$ be constant, and let $c^*(k) = \min_{\lambda>0} t(\lambda, k)$. Then for $p = c \cdot n / \binom{n}{k}$ with probability $1 - o(1)$ for $n \to \infty$ the following holds:*

*(i) if $c < c^*(k)$ then $H^k_{n,p}$ has an empty 2-core,*
*(ii) if $c > c^*(k)$ then $H^k_{n,p}$ has a non-empty 2-core.*

*Remark 1.* Actually this is only a special case of [18, Theorem 1.2] which covers $\ell$-cores for $k$-uniform hypergraphs for all $\ell \geq 2$, $k$ and $\ell$ not both equal to 2.

Consider hypergraphs $H^k_{n,p}$ with $p = c \cdot n / \binom{n}{k}$ and $k \geq 3$, as in Theorem 1. Since $\arg\min_{\lambda>0} t(\lambda, k) \approx \ln(k) + \ln(\ln(k))$, see [16, Section 4.3], the 2-core threshold $c^*(k)$ can be approximated via

$$c^*(k) \approx \frac{\ln(k) + \ln(\ln(k))}{k \cdot \left(1 - \frac{1}{k \cdot \ln(k)}\right)^{k-1}} . \tag{2}$$

The function $c^*(k)$ is monotonically decreasing. Hence the maximum 2-core threshold among all $k$-uniform hypergraphs is $c^*(3)$, which is about 0.818 [11,16, conjecture], [5, proof].

*Mixed Hypergraphs.* Let $H^{\boldsymbol{k}}_{n,\boldsymbol{p}}$ be a random hypergraph where each of the possible $\binom{n}{k_i}$ edges is present with probability $p_i$, given via the vectors $\boldsymbol{k} = (k_1, k_2, \ldots, k_s)$ and $\boldsymbol{p} = (p_1, p_2, \ldots, p_s)$. In other words, $H^{\boldsymbol{k}}_{n,\boldsymbol{p}}$ is a mixture of hypergraphs $H^k_{n,p}$ on $n$ nodes for different values of $p$ and $k$.

While studying hypergraphs in the context of cuckoo hashing the authors of [7] described how to extend the analysis of cores of $k$-uniform hypergraphs, with (expected) linear number of edges, to mixed hypergraphs, which directly leads to the following theorem. For $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_s) \in [0,1]^s$ with $\sum_{i=1}^s \alpha_i = 1$ let

$$t(\lambda, \boldsymbol{k}, \boldsymbol{\alpha}) = \frac{\lambda}{\sum\limits_{i=1}^{s} \alpha_i \cdot k_i \cdot \left(\Pr\left(\mathrm{Po}\left[\lambda\right] \geq 1\right)\right)^{k_i - 1}} \,. \tag{3}$$

**Theorem 2 (generalization of Theorem 1, implied by [7]).** *Let $s \geq 1$ be constant. For each $1 \leq i \leq s$ let $k_i \geq 3$ be constant, and let $\alpha_i \in [0,1]$ be constant, where $\sum_{i=1}^s \alpha_i = 1$. Furthermore let $c^*(\boldsymbol{k}, \boldsymbol{\alpha}) = \min_{\lambda > 0} t(\lambda, \boldsymbol{k}, \boldsymbol{\alpha})$. Then for $p_i = \alpha_i \cdot c \cdot n / \binom{n}{k_i}$ with probability $1 - o(1)$ for $n \to \infty$ the following holds:*

*(i) if $c < c^*(\boldsymbol{k}, \boldsymbol{\alpha})$ then $H^{\boldsymbol{k}}_{n,\boldsymbol{p}}$ has an empty 2-core,*
*(ii) if $c > c^*(\boldsymbol{k}, \boldsymbol{\alpha})$ then $H^{\boldsymbol{k}}_{n,\boldsymbol{p}}$ has a non-empty 2-core.*

Using ideas from [7, Section 4] this theorem can be proved along the lines of [18, Theorem 1.2] utilizing that $H^{\boldsymbol{k}}_{n,\boldsymbol{p}}$ is a mixture of a constant number of independent hypergraphs.

*Remark 2.* Analogous to Theorem 1, Theorem 2 can also be generalized such that it covers $\ell$-cores for all $\ell \geq 2$.

Often the 2-core threshold is given for hypergraph models slightly different from $H^k_{n,p}$ or $H^{\boldsymbol{k}}_{n,\boldsymbol{p}}$, as for example assumed in the opening of the introduction. A justification that some "common" hypergraph models are equivalent in terms of this threshold is given in Section 1.2.

Now consider hypergraphs $H^{\boldsymbol{k}}_{n,\boldsymbol{p}}$ as in Theorem 2, i.e., with $\alpha_i \cdot c \cdot n$ edges of size $k_i$ in expectation, $k_i \geq 3$. One can ask the following questions.

1. Assume $\boldsymbol{k}$ is given. What is the optimal vector $\boldsymbol{\alpha}^*$ such that the threshold $c^*(\boldsymbol{k}, \boldsymbol{\alpha}^*) =: c^*(\boldsymbol{k})$ is maximal among all thresholds $c^*(\boldsymbol{k}, \boldsymbol{\alpha})$? In other words, we want to solve the following optimization problem

$$c^*(\boldsymbol{k}) = \min_{\lambda > 0} t(\lambda, \boldsymbol{k}, \boldsymbol{\alpha}^*) = \max_{\boldsymbol{\alpha}} \min_{\lambda > 0} t(\lambda, \boldsymbol{k}, \boldsymbol{\alpha}) \,. \tag{4}$$

2. Is there a $\boldsymbol{k}$ such that $\boldsymbol{\alpha}^*$ gives some $c^*(\boldsymbol{k})$ that exceeds $c^*(3)$, the maximum 2-core threshold among all $k$-uniform hypergraphs (not mixed).

### 1.1 Results

We give the solution for the non-linear optimization problem (4) for $s = 2$. That is for each $\boldsymbol{k} = (k_1, k_2)$ we either give optimal solutions $\boldsymbol{\alpha}^* = (\alpha^*, 1 - \alpha^*)$ and $c^*(\boldsymbol{k})$ in analytical form or identify a subset of the interval $(0, 1]$ where we can

use binary search to determine $\alpha^*$ and therefore $c^*(\boldsymbol{k})$ numerically with arbitrary precision. Interestingly, it turns out that for adequate edge sizes $k_1$ and $k_2$ the maximum 2-core threshold $c^*(\boldsymbol{k})$ exceeds the maximum 2-core threshold $c^*(k)$ for $k$-uniform hypergraphs. The following table lists some values.

**Table 1.** Optimal 2-core thresholds $c^*(\boldsymbol{k})$, $\boldsymbol{k} = (k_1, k_2)$, and $\boldsymbol{\alpha}^* = (\alpha^*, 1-\alpha^*)$, and $\bar{k} = \alpha^* \cdot k_1 + (1-\alpha^*) \cdot k_2$. The values are rounded to the nearest multiple of $10^{-5}$.

| $(k_1, k_2)$ | $(3,3)$ | $(3,4)$ | $(3,6)$ | $(3,8)$ | $(3,10)$ | $(3,12)$ | $(3,14)$ | $(3,16)$ | $(3,21)$ |
|---|---|---|---|---|---|---|---|---|---|
| $c^*$ | 0.81847 | 0.82151 | 0.83520 | 0.85138 | 0.86752 | 0.88298 | 0.89761 | 0.91089 | 0.92004 |
| $\alpha^*$ | 1 | 0.83596 | 0.85419 | 0.86512 | 0.87315 | 0.87946 | 0.88464 | 0.88684 | 0.88743 |
| $\bar{k}$ | 3 | 3.16404 | 3.43744 | 3.67439 | 3.88795 | 4.08482 | 4.26898 | 4.47102 | 5.02626 |

More comprehensive tables for parameters $3 \le k_1 \le 6$ and $k_1 \le k_2 \le 50$ are given in the full version of this paper [19]. The maximum threshold found is about 0.92 for $\boldsymbol{k} = (3, 21)$.

*Remark 3.* In contrast, for any pair of hypergraphs $H^{\boldsymbol{k}}_{n,\boldsymbol{p}}$ and $H^{k}_{n,p}$ as above the following holds. Let $\hat{c}$ be the threshold where the edge density of the 2-core switches from below 1 to above 1. If $\sum_{i=1}^{s} \alpha_i \cdot k_i \le k$, then we have $\hat{c}(\boldsymbol{k}) \le \hat{c}(k)$.

### 1.2 Extensions to Other Hypergraph Models

While Theorems 1 and 2 are stated for hypergraphs $H^{\boldsymbol{k}}_{n,\boldsymbol{p}}$, one often considers slightly different hypergraphs, e.g. in the analysis of data structures.

Let $H^{\boldsymbol{k}}_{n,m,\boldsymbol{\alpha}}$ and $\tilde{H}^{\boldsymbol{k}}_{n,m,\boldsymbol{\alpha}}$ be random hypergraphs with $n$ nodes and $m$ edges, where for each $1 \le i \le s$, a fraction of $\alpha_i$ of the edges are fully randomly chosen from the set of all possible edges of size $k_i$. In the case of $H^{\boldsymbol{k}}_{n,m,\boldsymbol{\alpha}}$ the random edge choices are made *without* replacement and in the case of $\tilde{H}^{\boldsymbol{k}}_{n,m,\boldsymbol{\alpha}}$ the random edge choices are made *with* replacement. Using standard arguments, one sees that if $m = c \cdot n, k_i \ge 3$, and $p_i = \alpha_i \cdot c \cdot n / \binom{n}{k_i}$ as in the situation of Theorem 2, the 2-core threshold of $H^{\boldsymbol{k}}_{n,m,\boldsymbol{\alpha}}$ is the same as for $H^{\boldsymbol{k}}_{n,\boldsymbol{p}}$ (see e.g. [9, analogous to Proposition 2]), and the 2-core threshold of $\tilde{H}^{\boldsymbol{k}}_{n,m,\boldsymbol{\alpha}}$ is the same as for $H^{\boldsymbol{k}}_{n,m,\boldsymbol{\alpha}}$ (see e.g. [9, analogous to Proposition 1]).

### 1.3 Related Work

Non-uniform hypergraphs have proven very useful in the design of erasure correcting codes, such as Tornado codes [15,14], LT codes [13], Online codes [17], and Raptor codes [20]. Each of these codes heavily rely on one or more hypergraphs where the hyperedges correspond to variables (input/message symbols) and the nodes correspond to constraints on these variables (encoding/check symbols). An essential part of the decoding process of an encoded message is the application of a procedure that can be interpreted as peeling the hypergraph (see Algorithm 1) associated with the recovery process, where it is required that the result is an empty 2-core. Given $m$ message symbols, carefully designed non-uniform

hypergraphs allow, in contrast to uniform ones, to gain codes where in the example of Tornado, Online, and Raptor codes a random set of $(1+\varepsilon)\cdot m$ encoding symbols are necessary to decode the whole message in linear time (with high probability), and in the case of LT codes a random set of $m+o(m)$ encoding symbols are necessary to decode the whole message in time proportional to $m\cdot\ln(m)$ (with high probability). Tornado codes use explicit underlying hypergraphs designed for a given fixed code rate, whereas LT codes and its improvements, Online and Raptor codes, use implicit graph constructions to generate essentially infinite hypergraphs resulting in what is called rateless codes. In the case of Tornado codes the size of the hyperedges as well as the degree of the nodes follow precalculated sequences that are optimized to obtain the desired properties. In the case of LT codes, as well as in the last stage of Raptor and Online codes each node chooses its degree at random according to some fixed distribution, and then selects its incident hyperedges uniformly at random. (For Online codes also a skewed selection of the hyperedges is discussed, see [17, Section 7].) While the construction of the non-uniform hypergraph used for these codes is not quite the same as for $\tilde{H}^{\boldsymbol{k}}_{n,m,\boldsymbol{\alpha}}$ (or $H^{\boldsymbol{k}}_{n,\boldsymbol{p}}$, $H^{\boldsymbol{k}}_{n,m,\boldsymbol{\alpha}}$), since, among other reasons, the degree of the nodes is part of the design, they are similar enough to seemingly make the optimization methods / heuristics of [14] applicable, see footnote [10, page 796 (5)]. Having said that, compared to e.g. [14], our optimization problem is easier in the sense that it has fewer free parameters and harder in the sense that we are seeking a global optimum.

### 1.4 Overview of the Paper

In the next section we discuss the effect of our results on three hashing-based data structures. Afterwards, we give our main theorem that shows how to determine optimal 2-core thresholds for mixed hypergraphs with two different edge sizes. It follows a section with experimental evaluation of the appearance of 2-cores for a few selected mixed hypergraphs, which underpins our theoretical results. We conclude with a short summary and an open question.

## 2 Some Applications to Hashing-Based Data Structures

Several hashing-based data structures are closely related to the 2-core threshold of $k$-uniform hypergraphs (often one considers $\tilde{H}^{\boldsymbol{k}}_{n,m,\boldsymbol{\alpha}}$ for $s=\alpha_1=1$ and $k_1=3$). More precisely, the space usage of these data structures is inversely proportional to the 2-core threshold $c^*(k)$, while the evaluation time is proportional to the edge size $k$. By showing that the value of $c^*(3)$ can be improved using mixed hypergraphs instead of uniform ones, our result opens a new possibility for a space–time tradeoff regarding these data structures, allowing to further reduce their space needs at the cost of a constant increase in the evaluation time. Below we briefly sketch three data structures and discuss possible improvements, where we make use of the following definitions.

Let $\boldsymbol{a} = (a_1, a_2, \ldots, a_n)$ be a vector with $n$ cells each of size $r$ bits. Let $S = \{x_1, x_2, \ldots, x_m\}$ be a set of $m$ keys, where $S$ is subset of some universe $U$ and it holds $m = c \cdot n$ for some constant $c < 1$. The vector cells correspond to nodes of a hypergraph and the keys from $S$ are mapped via some function $\varphi$ to a sequence of vector cells and therefore correspond to hyperedges. We identify cells (and nodes) via their index $i$, $1 \leq i \leq n$, where $a_i$ stands for the value of cell $i$. The following three data structures essentially consist of a vector $\boldsymbol{a}$ and a mapping $\varphi$. For each data structure we compare their performance, depending if $\varphi$ realizes a uniform or a mixed hypergraph. In the case of a uniform hypergraph each key $x_j$ is mapped to $k = 3$ random nodes $\varphi(x_j) = (g_0(x_j), g_1(x_j), g_2(x_j))$ via functions $g_0, g_1, g_2 : U \to \{1, 2, \ldots, n\}$. In the case of a mixed hypergraph, for example, a fraction of $\alpha^* = 0.88684$ keys are mapped to 3 random nodes using functions $g_0, g_1, g_2$ and a fraction of $1 - \alpha^*$ keys are mapped to 16 random nodes via functions $g'_0, g'_1, \ldots, g'_{15} : U \to \{1, 2, \ldots, n\}$. We fix $c$ below the 2-core threshold to $c = c^* - 0.005$, which gives $c = 0.813$ in the uniform case and $c = 0.906$ in the mixed case, cf. Table 1. The reason why we use a rather small distance of 0.005 is that for large $m$ one observes a fairly sharp phase transition from "empty 2-core" to "non-empty 2-core" in experiments, cf. Section 4.

### 2.1 Invertible Bloom Lookup Table

The invertible Bloom Lookup Table [10] (IBLT) is a Bloom filter data structure that, amongst others, supports a complete listing of the inserted elements (with high probability). We restrict ourselves to the case where the IBLT is optimized for the listing operation and we assume without loss of generality that the keys from $S$ are integers. Each vector cell contains a summation counter and a quantity counter, initialized with 0. The keys arrive one by one and are inserted into the IBLT. Inserting a key $x_j$ adds its value to the summation counter and increments the quantity counter at each of the cells given via $\varphi(x_j)$. To list the inserted elements of the IBLT one essentially uses the standard peeling process for finding the 2-core of the underlying hypergraph (see Algorithm 1). While there exists a cell where the quantity counter has value 1, extract the value of the summation counter of this cell which gives some element $x_j$. Determine the summation counters and quantity counters associated with $x_j$ via evaluating $\varphi(x_j)$ and subtract $x_j$ from the summation counters and decrement the quantity counters. With this method a complete listing of the inserted elements is possible if the 2-core of the hypergraph is empty. Therefore in the case of uniform hypergraphs we get a space usage of $n/m \cdot r \approx 1.23 \cdot r$ bits per key. As already pointed out by the authors of [10], who highlight parallels to erasure correcting codes (see Section 1.3), a non-uniform version of the IBLT where keys have a different number of associated cells could improve the maximum fraction $c = m/n$ where a complete listing is successful with high probability. Using our example of mixed hypergraphs leads to such an improved space usage of about $1.10 \cdot r$ bits per key.

### 2.2 Retrieval Data Structure

Given a set of key-value pairs $\{(x_j, v_j) \mid x_j \in S, v_j \in \{0,1\}^r, j \in [m]\}$, the *retrieval problem* is the problem of building a function $f : U \to \{0,1\}^r$ such that for all $x_j$ from $S$ it holds $f(x_j) = v_j$; for any $y$ from $U \setminus S$ the value $f(y)$ can be an arbitrary element from $\{0,1\}^r$. Chazelle et al. [4] gave a simple and practical construction of a retrieval data structure, consisting of a vector $\boldsymbol{a}$ and some mapping $\varphi$ that has constant evaluation time, via simply calculating $f(x_j) = \bigoplus_{i \in \varphi(x_j)} a_i$. The construction is based on the following observation, which is stated more explicitly in [3]. Let $\boldsymbol{v} = (v_1, v_2, \ldots, v_m)$ be the vector of the function values and let $\boldsymbol{M}$ be the $m \times n$ incidence matrix of the underlying hypergraph, where the characteristic vector of each hyperedge is a row vector of $\boldsymbol{M}$. If the hypergraph has an empty 2-core, then the linear system $\boldsymbol{M} \cdot \boldsymbol{a} = \boldsymbol{v}$ can be solved in linear time in $(\{0,1\}^r, \oplus)$. For appropriate $c$ this gives expected linear construction time. As before, in the case of uniform hypergraphs the space usage is about $1.23 \cdot r$ bits per key. And in our example of mixed hypergraphs the space usage is about $1.10 \cdot r$ bits per key at the cost of a slight increase of the evaluation time of $f$.

In [8] it is shown how to obtain a retrieval data structure with space usage of $(1 + \varepsilon) \cdot r$ bits per key, for any fixed $\varepsilon > 0$, evaluation time $O(\log(1/\varepsilon))$, and linear expected construction time, while using essentially the same construction as above. The central idea is to transfer the problem of solving one large linear system into the problem of solving many small linear systems, where each system fits into a single memory word and can be solved via precomputed pseudoinverses. As shown in [1] this approach is limited in its practicability but can be adapted to build retrieval data structures with $1.10 \cdot r$ bits per key (and fewer) for realistic key set sizes. But this modified construction could possibly be outperformed by our direct approach of solving one large linear system in expected linear time.

### 2.3 Perfect Hash Function

Given a set of keys $S$, the problem of *perfect hashing* is to build a function $h : U \to \{1, 2, \ldots, n\}$ that is 1-to-1 on $S$. The construction from [3] and [4] gives a data structure consisting of a vector $\boldsymbol{a}$ and some mapping $\varphi$ that has constant evaluation time. Formulated in the context of retrieval, one builds a vector $\boldsymbol{v} = (v_1, v_2, \ldots, v_m)$ such that each key $x_j$ is associated with a value $f(x_j) = v_j$ that is the index $\iota$ of a hash function $g_\iota$ used in the sequence $\varphi(x_j)$. This node $g_\iota(x_j)$ must have the property that if one applies the peeling process to the underlying hypergraph (Algorithm 1) it will be selected and removed because it gets degree 1. If $c$ is below the 2-core threshold then with high probability for each $x_j$ there exists such an index $\iota$, and the linear system $\boldsymbol{M} \cdot \boldsymbol{a} = \boldsymbol{v}$ can be solved in linear time in $\mathbb{Z}_k$. Given the vector $\boldsymbol{a}$ the evaluation of $h$ is done via $h(x_j) = \varphi(x_j)_{\iota+1}$, where $\iota = \sum_{i \in \varphi(x_j)} a_i \bmod k$.

In the case of a 3-uniform hypergraph one gets a space usage of about $1.23 \cdot 2$ bits per key, since there are at most $k = 3$ different entries in $\boldsymbol{a}$. If one applies a simple compression method that stores every 5 consecutive elements from $\boldsymbol{a}$

in one byte, one gets a space usage of about $1.23 \cdot 8/5 \approx 1.97$ bits per key. The range of $h$ is $n = 1.23 \cdot m$.

In contrast to the examples above, improving this data structure by simply using a mixed hypergraph is not completely straightforward, since the increase of the load $c$ is compensated by the increase of the maximum index in the sequence $\varphi(x_j)$, which in our example would lead to a space usage of about $1.10 \cdot 4$ bits per key for uncompressed $\boldsymbol{a}$, since we use up to $k = 16$ functions for $\varphi(x_j)$. However, this can be circumvented by modifying the construction of the vector $\boldsymbol{v}$ as follows. Let $G = (S \cup \{1, 2, \ldots, n\}, E)$ be a bipartite graph with edge set $E = \{\{x, g_\iota(x_j)\} \mid x_j \in S, \iota \in \{0, 1, 2\}\}$. According to the results on 3-ary cuckoo hashing, see e.g. [9,7], it follows that for $c < 0.917$ (as in our case) the graph $G$ has a left-perfect matching with high probability. Given such a matching one stores in $\boldsymbol{v}$ for each key $x_j$ the index $\iota$ of $g_\iota$ that has the property that $\{x_j, g_\iota(x_j)\}$ is a matching edge. Now given the solution of $\boldsymbol{M} \cdot \boldsymbol{a} = \boldsymbol{v}$ in $\mathbb{Z}_3$, the function $h$ is evaluated via $h(x_j) = g_\iota(x_j)$, where $\iota = \sum_{i \in \varphi(x_j)} a_i \bmod 3$. Since $\boldsymbol{a}$ has at most three different entries it follows that the space usage in our mixed hypergraph case is about $1.10 \cdot 2$ bits per key. Using the same compression as before, the space usage can be reduced to about $1.10 \cdot 8/5 = 1.76$ bits per key. Moreover, the range of $h$ is now $n = 1.10 \cdot m$. Solving the linear system can be done in expected linear time. It is conjectured, supported by experimental results, that if $G$ has a matching then it is found by the $(k, 1)$-generalized selfless algorithm from [6, Section 5]; this algorithm can be implemented to work in expected linear time.

A more flexible trade-off between space usage and range yields the CHD algorithm from [2]. This algorithm allows to gain ranges $n = (1 + \varepsilon) \cdot m$ for arbitrary $\varepsilon > 0$ in combination with a adjustable compression rate that depends on some parameter $\lambda$. For example, using a range of about $1.11 \cdot m$, a space usage of 1.65 bits per key is achievable, see [2, Figure 1(b), $\lambda = 5$]. But since the expected construction time of the CHD algorithm is $O(m \cdot (2^\lambda + (1/\varepsilon)^\lambda))$ [2, Theorem 2], our approach could be faster for a comparable space usage and range.

## 3 Maximum Thresholds for the Case $s = 2$

In this section we state our main theorem that gives a solution for the non-linear optimization problem (4) for the case $s = 2$, that is given two edge sizes we show how to compute the optimal (expected) fraction of edges of each size such that the threshold of the appearance of a 2-core of a random hypergraph using this configuration is maximal.

Let $\boldsymbol{k} = (a, b)$ with $a \geq 3$, and $b > a$. Furthermore, let $\boldsymbol{\alpha} = (\alpha, 1 - \alpha)$ and[1] $\alpha \in (0, 1]$, as well as $\lambda \in (0, +\infty)$. Consider the following threshold function as a special case of (3)

$$t(\lambda, a, b, \alpha) = \frac{\lambda}{\alpha \cdot a \cdot (1 - e^{-\lambda})^{a-1} + (1 - \alpha) \cdot b \cdot (1 - e^{-\lambda})^{b-1}} \,. \tag{5}$$

[1] We can exclude the case $\alpha = 0$, since if $3 \leq a < b$, then it holds that $c^*(a) > c^*(b)$.

We transform $t(\lambda, a, b, \alpha)$ in a more manageable function using a monotonic and bijective domain mapping via $z = 1 - e^{-\lambda}$ and $\lambda = -\ln(1-z)$. Hence the transformed threshold function is

$$T(z, a, b, \alpha) = \frac{-\ln(1-z)}{\alpha \cdot a \cdot z^{a-1} + (1-\alpha) \cdot b \cdot z^{b-1}} , \tag{6}$$

where $z \in (0, 1)$. According to (4) and using $T(z, a, b, \alpha)$ instead of $t(\lambda, a, b, \alpha)$ the optimization problem is defined as

$$\max_{\alpha \in (0,1]} \min_{z \in (0,1)} T(z, a, b, \alpha) . \tag{7}$$

For a short formulation of our results we make use of the following three auxiliary functions.

$$f(z) = \frac{-\ln(1-z) \cdot (1-z)}{z} \tag{8}$$

$$g(z, a, b) = f(z) \cdot (b-1) \cdot (a-1) + \frac{1}{1-z} + 2 - b - a \tag{9}$$

$$h(z, a, b) = \frac{a \cdot z^{a-b} - b - f(z) \cdot (a \cdot (a-1) \cdot z^{a-b} - b \cdot (b-1))}{b \cdot ((b-1) \cdot f(z) - 1)} . \tag{10}$$

Furthermore we need to define some "special" points.

$$z' = \left(\frac{a}{b}\right)^{\frac{1}{b-a}} \qquad z_l = f^{-1}\left(\frac{1}{a-1}\right) \qquad z_r = f^{-1}\left(\frac{1}{b-1}\right) \tag{11}$$

$$z_1 = \min\{z \mid g(z) = 0\} \quad z_2 = \max\{z \mid g(z) = 0\} . \tag{12}$$

It can be shown that if $z_1$ and $z_2$ exist, then it holds $z' \neq z_1$ and $z' \neq z_2$. Now we can state our main theorem[2].

**Theorem 3.** *Let $a, b$ be fixed and let $T(z^*, \alpha^*) = \max_{\alpha \in (0,1]} \min_{z \in (0,1)} T(z, \alpha)$. Then the following holds:*

1. *Let $\min_z g(z) \geq 0$.*
   (*i*) *If $h(z') \leq 1$ then the optimal point is $(z^*, \alpha^*) = (z_l, 1)$ and the maximum threshold is given by*

$$T(z^*, \alpha^*) = \frac{-\ln(1-z_l)}{a \cdot z_l^{a-1}} .$$

   (*ii*) *If $h(z') > 1$ then the optimal point is the saddle point*

$$(z^*, \alpha^*) = \left(\left(\tfrac{a}{b}\right)^{\frac{1}{b-a}}, \tfrac{b-1}{b-a} - \tfrac{1}{f(z^*) \cdot (b-a)}\right)$$

   *and the maximum threshold is given by*

$$T(z^*, \alpha^*) = -\ln\left(1 - \left(\frac{a}{b}\right)^{\frac{1}{b-a}}\right) \cdot \left(\frac{b^{a-1}}{a^{b-1}}\right)^{\frac{1}{b-a}} .$$

[2] For any function $\phi = \phi(\cdot, x)$ we will use $\phi(\cdot)$ and $\phi(\cdot, x)$ synonymously, if $x$ is considered to be fixed.

2. *Let* $\min_z g(z) < 0$.
   - (*i*) *If* $h(z') \leq 1$ *then the optimum is the same as in case* 1(*i*).
   - (*ii*) *If* $h(z') \in (1, h(z_2)]$ *then the optimum is the same as in case* 1(*ii*).
   - (*iii*) *If* $h(z') \in (h(z_2), h(z_1))$ *then there are two optimal points* $(z^*, \alpha^*)$ *and* $(z^{**}, \alpha^*)$. *It holds* $1/\alpha^* = h(z^*) = h(z^{**})$ *and* $T(z^*, \alpha^*) = T(z^{**}, \alpha^*)$. *The optimal points can be determined numerically using binary search for the value* $\alpha$ *that gives* $T(\tilde{z}_1, \alpha) = T(\tilde{z}_2, \alpha)$, *where* $\alpha$ *is from the interval* $[1/h(z_{\text{up}}), 1/h(z_{\text{lo}})]$ *and it holds* $h(\tilde{z}_1) = h(\tilde{z}_2) = 1/\alpha$, *with* $\tilde{z}_1$ *from* $(z_l, z_{\text{up}})$, *and* $\tilde{z}_2$ *from* $(z_{\text{lo}}, z_r)$. *The (initial) interval for* $\alpha$ *is:*
     - ▷ $[1/h(z_1), 1/h(z_2)]$, *if* $z_1 < z' < z_2$,
     - ▷ $[1/h(z'), 1/h(z_2)]$, *if* $z' < z_1$,
     - ▷ $[1/h(z_1), 1/h(z')]$, *if* $z' > z_2$.
   - (*iv*) *If* $h(z') \in [h(z_1), \infty)$ *then the optimum is the same as in case* 1(*ii*).

Sketch of Proof. Assume first that $\alpha \in (0, 1]$ is arbitrary but fixed, that is we are looking for a global minimum of (6) in $z$-direction. Since $\lim_{z\to 0} T(z) = \lim_{z\to 1} T(z) = +\infty$ and $T(z)$ is continuous for $z \in (0, 1)$, a global minimum must be a point where the first derivative of $T(z)$ is zero, that is a critical point. Let $\tilde{z}$ be a critical point of $T(z)$ then it must hold $\tilde{z} \in [z_l, z_r)$ and $\alpha = 1/h(\tilde{z})$.

1. Consider the case $\min g(z) > 0$. (The case $\min g(z) = 0$ can be handled analogously). Since $\frac{\partial h(z)}{\partial z} > 0 \Leftrightarrow g(z) > 0$, the function $h(z)$ is monotonically increasing in $\in [z_l, z_r)$. Furthermore it holds, if $g(\tilde{z}) > 0$ then $\tilde{z}$ is a local minimum point of $T(z)$. It follows that for each $\alpha$ there is only one critical point $\tilde{z}$ and according to the monotonicity of $T(z)$ this must be a global minimum point. Now consider the function of critical points $\tilde{T}(z) := T(z, 1/h(z))$ of $T(z, \alpha)$. It holds that $\forall z < z'\colon \frac{\partial \tilde{T}(z)}{\partial z} > 0 \Leftrightarrow g(z) > 0$ and $\forall z > z'\colon \frac{\partial \tilde{T}(z)}{\partial z} < 0 \Leftrightarrow g(z) > 0$ . It follows that the function of critical points has a global maximum at $z' = (a/b)^{\frac{1}{b-a}}$, where $z'$ is at the same time a global minimum of $T(z, \alpha)$ in $z$-direction.
   - (*i*) If $h(z') > 1$ then $\alpha = 1/h(z') \in (0, 1)$ and the optimum point $(z^*, \alpha^*)$ is $(z', 1/h(z'))$, which is the only saddle point of $T(z, \alpha)$.
   - (*ii*) If $h(z') \leq 1$ then because of the monotonicity of $\tilde{T}(z)$ the solution for $\alpha^*$ is 1 (degenerated solution). Since $h(z_l) = 1$ it follows that that $(z^*, \alpha^*) = (z_l, 1)$.

2. Consider the case $\min g(z) < 0$. The function $g(z)$ has exactly two roots, $z_1$ and $z_2$, and for $z \in (z_l, z_r)$ the function $h(z)$ is strictly increasing to a local maximum at $z_1$, is then strictly decreasing to a local minimum at $z_2$, and is strictly increasing afterwards. Now for fixed $\alpha$ there can be more than one critical point and one has to do a case-by-case analysis.

A complete proof of the theorem is given in the full version [19]. □

The distinction between case 1 and case 2 of Theorem 3 can be done via solving $\frac{\partial g(z)}{\partial z} = 0$, for $z \in (0, 1)$, since the function $g(z)$ has only one critical point and this point is a global minimum point. Hence, Theorem 3 can be easily transformed into an algorithm that determines $\alpha^*, z^*$ and $T(z^*, \alpha^*)$ for given $\boldsymbol{k} = (a, b)$. (The pseudocode of such an algorithm is given in [19].) Some results for $c^*(\boldsymbol{k}) = t(\lambda^*, a, b, \alpha^*) = T(z^*, a, b, \alpha^*)$ for selected $\boldsymbol{k} = (a, b)$ are given

in Table 1 and in [19]. They show that the optimal 2-core threshold of mixed hypergraphs can be above the 2-core threshold for 3-uniform hypergraphs.

## 4 Experiments

In this section we consider mixed hypergraphs $\tilde{H}^{\boldsymbol{k}}_{n,m,\boldsymbol{\alpha}}$ as described in Section 1.2. For the parameters $\boldsymbol{k} = (k_1, k_2) \in \{(3,4),(3,8),(3,16),(3,21)\}$ and the corresponding optimal fractions of edge size $\boldsymbol{\alpha}^*$ we experimentally approximated the point $c^*(\boldsymbol{k})$ of the phase transition from empty to non-empty 2-core.

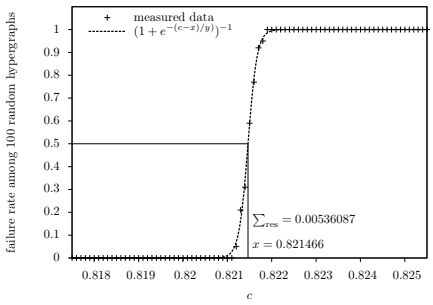



**Fig. 1.** $(k_1, k_2) = (3, 4)$

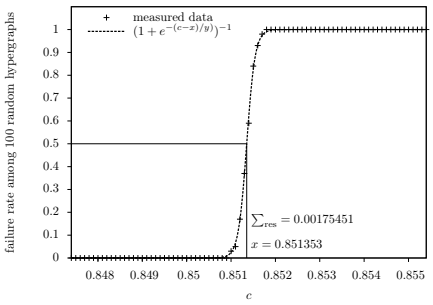



**Fig. 2.** $(k_1, k_2) = (3, 8)$

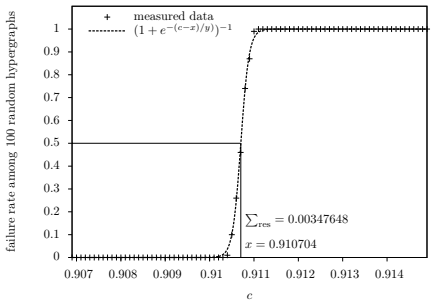



**Fig. 3.** $(k_1, k_2) = (3, 16)$

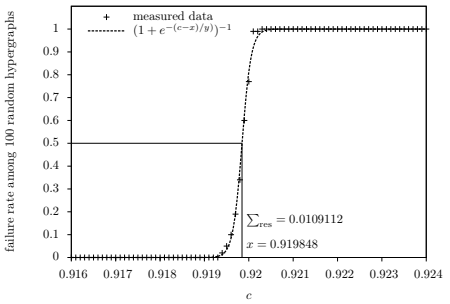



**Fig. 4.** $(k_1, k_2) = (3, 21)$

| $(k_1, k_2)$ | $(3,4)$ | $(3,8)$ | $(3,16)$ | $(3,21)$ |
|---|---|---|---|---|
| $c^*$ | 0.82151 | 0.85138 | 0.91089 | 0.92004 |
| $x$ | 0.82147 | 0.85135 | 0.91070 | 0.91985 |
| $\sum_{\text{res}}$ | 0.00536 | 0.00175 | 0.00348 | 0.01091 |

**Table 2.** Comparison of experimentally approximated and theoretical 2-core thresholds. The values are rounded to the nearest multiple of $10^{-5}$.

For each fixed tuple $(\boldsymbol{k}, \boldsymbol{\alpha}^*)$ we performed the following experiments. We fixed the number of nodes to $n = 10^7$ and considered growing equidistant edge densities $c = m/n$. The densities covered an interval of size 0.008 with the

theoretical 2-core threshold $c^*(\boldsymbol{k})$ in its center. For each quintuple $(k_1, k_2, \alpha^*, n, c)$ we constructed $10^2$ random hypergraphs $\tilde{H}^{\boldsymbol{k}}_{n,m,\boldsymbol{\alpha}}$ with nodes $\{1, 2, \ldots, n\}$ and $\alpha^* \cdot c \cdot n$ edges of size $k_1$ and $(1 - \alpha^*) \cdot c \cdot n$ edges of size $k_2$. For the random choices of each edge we used the pseudo random number generator MT19937 "Mersenne Twister" of the GNU Scientific Library[3]. Given a concrete hypergraph we applied Algorithm 1 to determine if the 2-core is empty.

A non-empty 2-core was considered as failure, an empty 2-core was considered as success. We measured the failure rate and determined an approximation of the 2-core threshold, via fitting the sigmoid function $\sigma(c; x, y) = (1 + \exp(-(c - x)/y))^{-1}$ to the measured failure rate using the "least squares fit" of `gnuplot`[4].The resulting fit parameter $x = x(\boldsymbol{k})$ is our approximation of the theoretical threshold $c^*(\boldsymbol{k})$. Table 2 compares $c^*(\boldsymbol{k})$ and $x(\boldsymbol{k})$. The quality of the approximation is quantified in terms of the sum of squares of residuals $\sum_{\text{res}}$. The results show a difference of theoretical and experimentally estimated threshold of less than $2 \cdot 10^{-4}$. The corresponding plots of the measured failure rates and the fit function are shown in Figures 1, 2, 3 and 4.

## 5 Summary and Future Work

We have shown that the threshold for the appearance of a 2-core in mixed hypergraphs can be larger than the 2-core threshold for $k$-uniform hypergraphs, for each $k \geq 3$. Moreover, for hypergraphs with two given constant edges sizes we showed how to determine the optimal (expected) fraction of edges of each size, that maximizes the 2-core threshold. The maximum threshold found for $3 \leq k_1 \leq 6$ and $k_1 \leq k_2 \leq 50$ is about 0.92 for $\boldsymbol{k} = (3, 21)$. We *conjecture* that this is the best possible for two edge sizes.

Based on the applications of mixed hypergraphs, as for example discussed in Section 2, the following question seems natural to ask. Consider the hypergraph $\tilde{H}^{\boldsymbol{k}}_{n,m,\boldsymbol{\alpha}}$ and some fixed upper bound $\bar{K}$ on the average edge size $\bar{k} = \sum_{i=1}^{s} \alpha_i \cdot k_i$.

*Question.* Which pair of vectors $\boldsymbol{k}$ and $\boldsymbol{\alpha}$ that gives an average edge size below $\bar{K}$ maximizes the threshold for the appearance of a 2-core? That means we are looking for the solution of $\max_{\boldsymbol{k},\boldsymbol{\alpha}} \min_{\lambda > 0} t(\lambda, \boldsymbol{k}, \boldsymbol{\alpha})$ under the constraint that $\bar{k} \leq \bar{K}$.

**Acknowledgment.** The author would like to thank Martin Dietzfelbinger for many helpful suggestions and the time he spent in discussions on the topic. He also would like to thank Udi Wieder for pointing him to applications of mixed hypergraphs in LT codes, Online codes and Raptor codes.

## References

1. Aumüller, M., Dietzfelbinger, M., Rink, M.: Experimental Variations of a Theoretically Good Retrieval Data Structure. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 742–751. Springer, Heidelberg (2009)

[3] GNU Scientific Library, `http://www.gnu.org/software/gsl/`

[4] Gnuplot 4.4: An Interactive Plotting Program, `http://gnuplot.sourceforge.net/`

2. Belazzougui, D., Botelho, F.C., Dietzfelbinger, M.: Hash, Displace, and Compress. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 682–693. Springer, Heidelberg (2009)
3. Botelho, F.C., Pagh, R., Ziviani, N.: Simple and Space-Efficient Minimal Perfect Hash Functions. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 139–150. Springer, Heidelberg (2007)
4. Chazelle, B., Kilian, J., Rubinfeld, R., Tal, A.: The Bloomier Filter: An Efficient Data Structure for Static Support Lookup Tables. In: Proc. 15th SODA, pp. 30–39 (2004)
5. Cooper, C.: The Cores of Random Hypergraphs with a Given Degree Sequence. Random Struct. Algorithms 25(4), 353–375 (2004)
6. Dietzfelbinger, M., Goerdt, A., Mitzenmacher, M., Montanari, A., Pagh, R., Rink, M.: Tight Thresholds for Cuckoo Hashing via XORSAT. CoRR abs/0912.0287 (2009), http://arxiv.org/abs/0912.0287
7. Dietzfelbinger, M., Goerdt, A., Mitzenmacher, M., Montanari, A., Pagh, R., Rink, M.: Tight Thresholds for Cuckoo Hashing via XORSAT. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 213–225. Springer, Heidelberg (2010)
8. Dietzfelbinger, M., Pagh, R.: Succinct Data Structures for Retrieval and Approximate Membership (Extended Abstract). In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 385–396. Springer, Heidelberg (2008)
9. Fountoulakis, N., Panagiotou, K.: Orientability of Random Hypergraphs and the Power of Multiple Choices. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 348–359. Springer, Heidelberg (2010)
10. Goodrich, M.T., Mitzenmacher, M.: Invertible Bloom Lookup Tables. In: Proc. 49th Communication, Control, and Computing (Allerton), pp. 792–799 (2011)
11. Havas, G., Majewski, B.S., Wormald, N.C., Czech, Z.J.: Graphs, Hypergraphs and Hashing. In: van Leeuwen, J. (ed.) WG 1993. LNCS, vol. 790, pp. 153–165. Springer, Heidelberg (1994)
12. Kim, J.H.: Poisson cloning model for random graphs. In: Proc. ICM Madrid 2006, vol. III, pp. 873–898 (2006)
13. Luby, M.: LT Codes. In: Proc. 43rd FOCS, p. 271 (2002)
14. Luby, M., Mitzenmacher, M., Shokrollahi, M.A., Spielman, D.A.: Efficient Erasure Correcting Codes. IEEE Transactions on Information Theory 47(2), 569–584 (2001)
15. Luby, M., Mitzenmacher, M., Shokrollahi, M.A., Spielman, D.A., Stemann, V.: Practical Loss-Resilient Codes. In: Proc. 29th STOC, pp. 150–159 (1997)
16. Majewski, B.S., Wormald, N.C., Havas, G., Czech, Z.J.: A Family of Perfect Hashing Methods. Comput. J. 39(6), 547–554 (1996)
17. Maymounkov, P.: Online codes (Extended Abstract). Tech. Rep. TR2002-833, New York University (2002)
18. Molloy, M.: The pure literal rule threshold and cores in random hypergraphs. In: Proc. 15th SODA, pp. 672–681 (2004)
19. Rink, M.: On Thresholds for the Appearance of 2-cores in Mixed Hypergraphs. CoRR abs/1204.2131 (2012), http://arxiv.org/abs/1204.2131
20. Shokrollahi, A.: Raptor Codes. IEEE Transactions on Information Theory 52(6), 2551–2567 (2006)

# Coalgebraic Bisimulation-Up-To

Jurriaan Rot[1,2,⋆], Marcello Bonsangue[1,2], and Jan Rutten[2,3]

[1] LIACS – Leiden University
{jrot,marcello}@liacs.nl
[2] Centrum Wiskunde en Informatica
[3] Radboud University Nijmegen
janr@cwi.nl

**Abstract.** Bisimulation-up-to enhances the bisimulation proof method for process equivalence. We present its generalization from labelled transition systems to arbitrary coalgebras, and show that for a large class of systems, enhancements such as bisimulation up to bisimilarity, up to equivalence and up to context are sound proof techniques. This allows for simplified bisimulation proofs for many different types of state-based systems.

## 1 Introduction

*Bisimilarity* is a fundamental notion of equivalence between labelled transition systems. Two processes are bisimilar if they are related by a bisimulation, which is a relation between states such that related pairs match each others transitions and their successors (also called derivatives) are again related. *Bisimulation-up-to* refers to a family of techniques for proving bisimilarity based on smaller relations than usual, in many cases reducing the amount of work [13,9,4]. For example, in a *bisimulation up to bisimilarity* the derivatives do not need to be related directly but may be *bisimilar* to states which are [8]; this is a valid proof method for bisimulation on labelled transition systems. *Bisimulation up to context* [13] is another such technique, in which one can use the algebraic structure (syntax) of processes to relate derivatives.

The theory of *coalgebras* provides a mathematical framework for the uniform study of many types of state-based systems, including labelled transition systems but also (non)-deterministic automata, stream systems, various types of probabilistic and weighted automata, etc. The type of a coalgebra is expressed by an endofunctor $F$. One of the main elements of the theory is the coalgebraic notion of bisimulation, which generalizes classical bisimulation for labelled transition systems to arbitrary coalgebras. Another notion of equivalence of coalgebras is *behavioural equivalence*. Intuitively, two states are behaviourally equivalent if they have the same observable behaviour, where the observations depend on the type functor $F$. For many types of systems, including labelled transition

⋆ The research of this author has been funded by the Netherlands Organisation for Scientific Research (NWO), CoRE project, dossier No.: 612.063.920.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 369–381, 2013.

systems, these notions coincide, but for some types, such as certain types of weighted automata, bisimulation is stronger than behavioural equivalence (see, e.g., [3]).

In this paper we introduce a generalization of bisimulation-up-to from labelled transition systems to the theory of coalgebras. In this setting we define bisimulation up to bisimilarity, up-to-union, up-to-context, up-to-equivalence and combinations thereof. As it turns out, the general notion of coalgebraic bisimulation-up-to which we introduce is somewhat problematic: we show that bisimulation up to bisimilarity is *not* sound in general, i.e., that it can not be used as a valid proof principle. So we introduce *behavioural equivalence-up-to*, for which all of the aforementioned instances work very well. Then by the correspondence between behavioural equivalence and bisimulation which holds for many types of systems (for coalgebras for weak pullback preserving functors, to be precise) we obtain the soundness of bisimulation-up-to for such systems.

*Related work.* Sangiorgi [13] introduced the first systematic treatment of more general up-to techniques for labelled transition systems, and discussed the important notion of bisimulation up to context. A good reference for the current state of the art in this line of research is [9]. In [4] bisimulation up to context is applied to obtain a very efficient algorithm for checking equivalence of non-deterministic finite automata. Lenisa [7] developed bisimulation-up-to in the context of a set-theoretic notion of coinduction. Moreover in *loc. cit.* a framework for up-to techniques for coalgebraic bisimulation is studied, but as mentioned in the paper itself already, the important notion of bisimulation up to bisimilarity is problematic in this setting. Bisimulation up to context is studied at a general coalgebraic level in [7].

The up-to-context technique for coalgebraic bisimulation was later derived as a special case of so-called $\lambda$-coinduction: Bartels [2] showed that this technique can be applied in the context of operators defined by certain natural transformations, corresponding for example to the well-known GSOS format in the case of labelled transition systems. A direct corollary of this is the soundness of bisimulation up to context for CCS. However, [2, pages 126, 129] mentions already that it would be ideal to combine the up-to-context technique with other up-to techniques. Indeed, combining up-to-context with up-to-bisimilarity or up-to-equivalence yields powerful proof techniques (see, e.g., [9] and this paper for examples).

The recent paper [14] introduces bisimulation-up-to, where the notion of bisimulation is based on a specification language for polynomial functors (which does not include, for example, labelled transition systems). In contrast, we base ourselves on the standard notion of bisimulation, and only need to restrict to weak pullback preserving functors, to obtain our soundness results.

*Outline.* The following section contains the necessary preliminaries. Then in Section 3 we introduce bisimulation-up-to for coalgebras, together with important instances and examples, and we discuss their soundness in Section 4. In Section 5 we recall behavioural equivalence and its relation with bisimulation,

and in Section 6 we present the main soundness results of bisimulation-up-to via behavioural equivalence-up-to. We conclude in Section 7.

## 2 Preliminaries

By $\mathsf{Set}$ we denote the category of sets and total functions. For a relation $R \subseteq X \times X$ we denote by $\overline{R}$ the smallest equivalence relation containing $R$, i.e., its reflexive, symmetric and transitive closure, or *equivalence closure*. If $R$ is an equivalence relation then we denote by $q_R : X \to X/R$ the quotient map, which sends each element to its equivalence class. For any relation $R \subseteq X \times Y$ we denote by $\pi_1^R : R \to X$ and $\pi_2^R : R \to Y$ the respective projection maps, and we omit $R$ if it is clear from the context. Given two functions $f : X \to Y$ and $g : X \to Z$, the *pairing* of $f$ and $g$ is the unique function $\langle f, g \rangle : X \to Y \times Z$ with the property that $\pi_1 \circ \langle f, g \rangle = f$ and $\pi_2 \circ \langle f, g \rangle = g$. The *kernel* of a function $f : X \to Y$ is defined as $\ker(f) = \{(x, y) \in X \times X \mid f(x) = f(y)\}$.

A *coalgebra* for a functor $F : \mathsf{Set} \to \mathsf{Set}$ is a pair $(X, \alpha)$ consisting of a set $X$ and a function $\alpha : X \to FX$. We call $X$ the *carrier* or the set of *states*, and $\alpha$ the *transition structure* or *dynamics*. A function $X \to Y$ between the respective carriers of two coalgebras $(X, \alpha)$ and $(Y, \beta)$ is a *homomorphism* if $Ff \circ \alpha = \beta \circ f$. For a coalgebra $\alpha : X \to FX$, a relation $R \subseteq X \times X$ is an $(F\text{-})$*bisimulation* if $R$ itself can be equipped with a transition structure $\gamma : R \to FR$ such that the following diagram commutes:

$$\begin{array}{ccccc} X & \xleftarrow{\pi_1} & R & \xrightarrow{\pi_2} & X \\ \downarrow{\scriptstyle \alpha} & & \downarrow{\scriptstyle \gamma} & & \downarrow{\scriptstyle \alpha} \\ FX & \xleftarrow[F\pi_1]{} & FR & \xrightarrow[F\pi_1]{} & FX \end{array}$$

The largest bisimulation on $\alpha$ exists [11] and is denoted by $\sim_\alpha$, or simply by $\sim$ if $\alpha$ is clear from the context. Two states $x, y \in X$ of a coalgebra are called *bisimilar* if $x \sim y$.

*Example 1.* We recall several types of coalgebras and their corresponding notions of bisimulation. In (3) and (4) below we introduce operations on coalgebras; these will become relevant in the examples in Section 3.

1. Finitely branching *labelled transition systems* (lts's) over a set of labels $A$ are coalgebras for the functor $FX = \mathcal{P}_f(A \times X)$. For an lts $\alpha : X \to \mathcal{P}_f(A \times X)$ we write $x \xrightarrow{a} x'$ iff $(a, x') \in \alpha(x)$. So intuitively, for a state $x \in X$, $\alpha(x)$ contains all the outgoing labelled transitions from $x$. Bisimulation instantiates to the classical definition. A relation $R \subseteq X \times X$ is called a *bisimulation* provided that for all $(x, y) \in R$: if $x \xrightarrow{a} x'$ then there exists a state $y'$ such that $y \xrightarrow{a} y'$ and $(x', y') \in R$, and vice versa.

2. One of the simplest interesting types of coalgebras is given by the functor $FX = X + \mathbb{1}$, where $\mathbb{1}$ is the singleton $\{*\}$. We call such coalgebras *deterministic systems with termination*. An $F$-coalgebra $\alpha : X \to X + \mathbb{1}$ can, for a given state $x$, either terminate ($\alpha(x) = *$) or make a transition to another state $x' \in X$ ($\alpha(x) = x'$). If a state $x$ terminates we write $x\downarrow$, otherwise we write $x'$ for $\alpha(x)$, and call $x'$ the *derivative* of $x$. In this case a *bisimulation* is a relation $R \subseteq X \times X$ such that for all $(x, y) \in R$ : either $x\downarrow$ and $y\downarrow$, or $(x', y') \in R$.
3. Coalgebras for the functor $FX = \mathbb{R} \times X$, where $\mathbb{R}$ is the set of real numbers, are called *stream systems* (over the reals). For a stream system $\langle o, t\rangle : X \to \mathbb{R} \times X$ and a state $x \in X$, if $o$ and $t$ are clear from the context we denote $o(x)$ by $x_0$ and $t(x)$ by $x'$. A relation $R \subseteq X \times X$ is a bisimulation if for each $(x, y) \in R$ : $x_0 = y_0$ and $(x', y') \in R$. A special stream system is formed by taking as carrier the set $\mathbb{R}^\omega = \{\sigma \mid \sigma : \mathbb{N} \to \mathbb{R}\}$ of all streams (infinite sequences) of elements of $\mathbb{R}$, and defining the transition structure $\langle o, t\rangle : \mathbb{R}^\omega \to \mathbb{R} \times \mathbb{R}^\omega$ as $o(\sigma) = \sigma(0)$ and $t(\sigma)(n) = \sigma(n + 1)$). This $F$-coalgebra is in fact the *final* one, that is, every stream system has a unique homomorphism into it [11]. We may define operations on streams by means of *behavioural differential equations* [12], in which an operation is defined by specifying its initial value and its derivative. Instead of recalling the general definition we only consider the operations of *addition* ($+$), *shuffle product* ($\otimes$) and *shuffle inverse* ($^{-1}$):

| Differential equation | Initial value | Name |
|---|---|---|
| $(\sigma + \tau)' = \sigma' + \tau'$ | $(\sigma + \tau)_0 = \sigma_0 + \tau_0$ | sum |
| $(\sigma \otimes \tau)' = \sigma' \otimes \tau + \sigma \otimes \tau'$ | $(\sigma \otimes \tau)_0 = \sigma_0 \times \tau_0$ | shuffle product |
| $(\sigma^{-1})' = -\sigma' \otimes (\sigma^{-1} \otimes \sigma^{-1})$ | $(\sigma^{-1})_0 = (\sigma_0)^{-1}$ | shuffle inverse |

The inverse operation is only defined on streams $\sigma$ for which $\sigma_0 \neq 0$. With every real number $r$ we associate a stream $[r] = (r, 0, 0, 0, \ldots)$ (we will abuse notation and denote $[r]$ by $r$), and we abbreviate $(-1) \otimes \sigma$ by $-\sigma$. The set of *terms* $T(\mathbb{R}^\omega)$ is defined by the grammar $t ::= \sigma \mid t_1 + t_2 \mid t_1 \otimes t_2 \mid t_1^{-1}$ where $\sigma$ ranges over $\mathbb{R}^\omega$. We can turn $T(\mathbb{R}^\omega)$ into a coalgebra $\mathcal{S} = (T(\mathbb{R}^\omega), \beta)$ by defining the transition structure by induction according to the final coalgebra (for the base case) and the above specification (for the other terms).
4. Coalgebras for the functor $FX = 2 \times X^A$ correspond to deterministic automata with transitions in $A$. For a coalgebra $\langle o, f\rangle : X \to 2 \times X^A$ and a state $x \in X$, we have $o(x) = 1$ iff $x$ is a *final* or *accepting* state. The function $f(x)$ assigns, to each alphabet letter $a \in A$, the next state or *a-derivative*, denoted $x_a$. A relation $R \subseteq X \times X$ is a bisimulation if for each $(x, y) \in R$ : $o(x) = o(y)$ and for each $a \in A$: $(x_a, y_a) \in R$. The set of formal languages $\mathcal{P}(A^*)$ can be given an $F$-transition structure $\langle o, f\rangle$ as follows: $o(l) = 1$ iff $l$ contains the empty word, and $f(l)(a) = \{\, w \mid aw \in L \,\}$. This is the final $F$-coalgebra.

Using a suitable format of behavioural differential equations we can define the operations of regular expressions as follows [10]:

| Differential equation | Initial value | Name |
|---|---|---|
| $0_a = 0$ | $o(0) = 0$ | zero |
| $1_a = 0$ | $o(1) = 1$ | one |
| $b_a = \begin{cases} 1 & \text{if } b = a \\ 0 & \text{otherwise} \end{cases}$ | $o(b) = 0$ | |
| $(x + y)_a = x_a + y_a$ | $o(x + y) = \max(o(x), o(y))$ | union |
| $(x \cdot y)_a = \begin{cases} x_a \cdot y & \text{if } o(x) = 0 \\ x_a \cdot y + y_a & \text{otherwise} \end{cases}$ | $o(x \cdot y) = \min(o(x), o(y))$ | composition |
| $(x^*)_a = x_a \cdot x^*$ | $o(x^*) = 1$ | Kleene star |

In a similar way as we have done for the operations on streams, we can construct the set of regular expressions over languages $T(\mathcal{P}(A^*))$ and turn it into a coalgebra $\mathcal{R} = (T(\mathcal{P}(A^*)), \beta)$, defining the transition structure $\beta$ by induction according to the final coalgebra and the above specification of the operators.

*Algebras and distributive laws.* In this paragraph we shortly recall some concepts needed for our discussion of bisimulation up to context. These are quite technical, and we do not have the space to cover them in detail; however, a large part of the remainder of this paper can be understood without these preliminaries. An (Eilenberg-Moore) *algebra* for a monad $T$ on $\mathsf{Set}$ is a tuple $(X, \alpha)$ consisting of a set $X$ and a map $TX \to X$, satisfying some additional laws (see, e.g., [6]). For an example, recall that in the above Example 1(3), we formed the set of terms $T(\mathbb{R}^\omega)$ over streams of real numbers. In general, given operation and constant symbols (with associated arities), the functor $T$ which constructs the corresponding set of terms over sets of variables $X$, is (part of) a monad, which we call the *term monad* (for this signature). The construction of regular expressions in Example 1(4) is another such example. Moreover for a set $X$, a monad gives us an algebra $\alpha : TTX \to TX$. In the case of a term monad this $\alpha$ is called a *term algebra*; it turns a term over terms into a single term.

An $(F, T)$-*bialgebra* is a triple $(X, \alpha, \beta)$ consisting of a $T$-algebra $\alpha$ and an $F$-coalgebra $\beta$. We can extend the coalgebra $\mathcal{S} = \langle T(\mathbb{R}^\omega), \beta \rangle$ from Example 1(3) above to a bialgebra $\langle T(\mathbb{R}^\omega), \alpha, \beta \rangle$ where $\alpha$ is the term algebra. Similarly $\mathcal{R}$ forms a bialgebra together with its associated term algebra. Given a $T$-algebra $\alpha$, the set of *contexts* over a relation $R \subseteq X \times X$ is defined as $C_\alpha(R) \subseteq X \times X = \langle \alpha \circ T\pi_1, \alpha \circ T\pi_2 \rangle(TR)$. If $T$ is a term monad, then $C_\alpha(R)$ can be characterized concretely as follows: for two terms $t_1, t_2 \in TX$ we have $(t_1, t_2) \in C_\alpha(R)$ iff we can obtain $t_2$ by substituting the variables of $t_1$ for variables related by $R$.

The interplay between syntax and semantics can be captured by the categorical notion of a *distributive law* of a monad $T$ over a functor $F \times Id$, which is a natural transformation $\lambda : T(F \times Id) \Rightarrow FT \times T$ satisfying some laws (see, e.g., [2,6] for a definition). Intuitively $T$ models syntax and $F$ models behaviour,

and distributive laws of this type can in fact be seen as abstract operational semantics. Indeed, for particular functors $F$ they correspond to concrete formats such as the well-known GSOS format for processes [2,6]. For instance, the definition of the operations in Example 1 above give rise to distributive laws. For a distributive law $\lambda$, a bialgebra is called a $\lambda$-*bialgebra* if $\alpha$ and $\beta$ decompose via $\lambda$; for lack of space we again refer to [2,6] for a precise definition. For now we note that the coalgebras $\mathcal{S}$ and $\mathcal{R}$ of Example 1, together with their respective term algebras, form $\lambda$-bialgebras.

## 3 Bisimulation-Up-To

We introduce the following definition, which generalizes the notions of *progression* and *bisimulation-up-to* from labelled transition systems [9] to coalgebras.

**Definition 1.** *Let $\alpha : X \to FX$ be an $F$-coalgebra. Given relations $R, S \subseteq X \times X$, we say that $R$* progresses to *$S$ if there exists a transition structure $\gamma : R \to FS$ making the following diagram commute:*

$$\begin{array}{ccccc} X & \xleftarrow{\pi_1^R} & R & \xrightarrow{\pi_2^R} & X \\ \downarrow{\scriptstyle \alpha} & & \downarrow{\scriptstyle \gamma} & & \downarrow{\scriptstyle \alpha} \\ FX & \xleftarrow[F\pi_1^S]{} & FS & \xrightarrow[F\pi_2^S]{} & FX \end{array}$$

*If $R$ progresses to $f(R)$ for a function $f : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ then we say $R$ is a* bisimulation up to *$f$.*

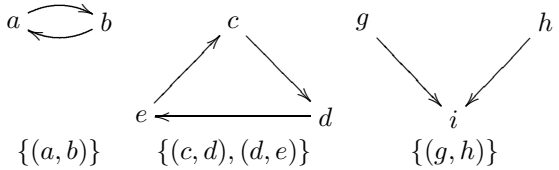
Notice that we have not put any restrictions on the function $f$ in the definition of bisimulation-up-to, and so in general a bisimulation up to $f$ will *not* be a bisimulation, neither will it contain only bisimilar pairs. We continue to introduce several concrete instances of bisimulation-up-to, inspired by the corresponding techniques for labelled transition systems [9]. For now, these instances of bisimulations-up-to do not yet guarantee any pairs to be bisimilar; a discussion of when they do follows afterwards, when we define and study their *soundness*.

*Up-to-identity.* For a trivial instance of bisimulation-up-to, consider the identity function *id* on relations. A bisimulation up to *id* is simply a bisimulation.

*Up-to-equivalence.* Consider the function $f(R) = \overline{R}$ which takes a relation $R$ to its equivalence closure. We call a bisimulation up to $f$ a bisimulation *up to equivalence*.

*Example 2.* Let $\alpha : X \to X + \mathbb{1}$ be a deterministic system with termination, and $R \subseteq X \times X$ a relation. Spelling out the definition, we find that $R$ is a bisimulation up to equivalence if for all $(x, y) \in R$: either $x \downarrow$ and $y \downarrow$, or $x' \ \overline{R} \ y'$. Thus instead of requiring the respective derivatives of each of the pairs

to be related in $R$ again, for a bisimulation up to equivalence, they need only to be related by the reflexive, symmetric and transitive closure of $R$. Consider the following three deterministic systems and relations (where $x \to y$ iff $x' = y$):




All three of these relations are bisimulations up to equivalence, whereas none of them are actual bisimulations. Consider for instance the relation $\{(a, b)\}$; when we compute the respective derivatives of $a$ and $b$ we obtain $a' = b$ and $b' = a$, but $(b, a) \notin \{(a, b)\}$. However, the pair $(b, a)$ *is* in the least equivalence relation containing $\{(a, b)\}$. □

*Up-to-union.* For $S \subseteq X \times X$ a relation, consider the function $f(R) = R \cup S$. We call a bisimulation up to $f$ a bisimulation *up to union with* $S$. In order for a relation $R$ to be a bisimulation up to union with $S$, the derivatives of $R$ may be related either by $R$ again or by $S$.

*Example 3.* For a deterministic automaton $\langle o, f \rangle : X \to 2 \times X^A$, a relation $R \subseteq X \times X$ is a bisimulation up to union with $S$ if for all $(x, y) \in R$: $o(x) = o(y)$ and for any alphabet letter $a \in A$: either $x_a R y_a$ or $x_a S y_a$.

Consider for instance the relation $R = \{(1^*, 1)\}$ on the automaton $\mathcal{R}$ of regular expressions over languages, introduced in Example 1(4). We first note that $o(1^*) = 1 = o(1)$; next let $a \in A$ be an alphabet letter. Then $(1^*)_a = 1_a \cdot 1^* = 0 \cdot 1^*$ and $1_a = 0$. Note that $(0 \cdot 1^*, 0) \notin R$, so $R$ is not a bisimulation. However suppose $S$ is a relation containing some of the basic equivalences between regular expressions, such as $(0 \cdot r, 0) \in S$ for any regular expression $r$. Then our relation $R$ is a bisimulation up to union with $S$, since the derivatives of the single pair in $R$ are related by $S$. Notice that $R$ is also a bisimulation up to union with $\sim$, given that $0 \cdot a$ is bisimilar to 0. □

*Up-to-context.* The notion of bisimulation up to *context* applies to coalgebras where the state space consists of the elements of an algebra. So let $(X, \alpha, \beta)$ be an $(F, T)$-bialgebra (see the last part of Section 2). If a relation $R \subseteq X \times X$ progresses to the set of contexts $C_\alpha(R)$, then we call $R$ a *bisimulation up to context.*

If $T$ is a term monad, such as in the case of the introduced operations on streams or regular expressions, then in practice this technique only becomes interesting when combined with other techniques such as up-to-equivalence or up-to-bisimilarity. A notable exception is when one considers bisimulation up to context on a *final* coalgebra, which is the approach taken in the examples in [2] (e.g., page 126). In that case, combination with up-to-bisimilarity comes for free, since bisimilarity implies equality on final coalgebras [11].

*Up-to-union-and-equivalence.* Let $f(R) = \overline{R \cup S}$ for a fixed relation $S$. If $R$ progresses to $f(R)$ then we call $R$ a *bisimulation up to $S$-union and equivalence*. By taking the equivalence closure of $R \cup S$, derivatives may be related by arbitrary compositions of $R$ and $S$. If we consider the special case $S = \sim$ we see that this is in fact a generalization of *bisimulation up to bisimilarity*, meaning that the derivatives may be *bisimilar* to elements which *are* related by $R$. More formally, a bisimulation up to bisimilarity is based on the function $g(R) = \sim \circ R \circ \sim$. Note that for any relation $R$, we have $g(R) \subseteq f(R)$. Indeed for a bisimulation up to $\sim$-union and equivalence we take arbitrary compositions of $\sim$ and $R$, including the specific one $\sim \circ R \circ \sim$.

*Example 4.* For a deterministic automaton $\langle o, f \rangle : X \to 2 \times X^A$, a relation $R \subseteq X \times X$ is a bisimulation up to $S$-union and equivalence if for all $(x, y) \in R$: $o(x) = o(y)$ and for any alphabet letter $a \in A$: $(x, y) \in \overline{R \cup S}$. Recall the automaton $\mathcal{R}$ of regular expressions from Example 1(4). Consider the implication $l \sim al + b \Rightarrow l \sim a^*b$ for a language $l \in \mathcal{P}(A^*)$ over alphabet symbols $A = \{a, b\}$, intuitively expressing that $a^*b$ is the unique solution of the "equation" $l \sim al + b$. Let $R = \{(l, a^*b) \mid l \sim al + b\}$, and let $l$ be a language such that $l \sim al + b$. First we check that the outputs are equal: $o(l) = o(al + b) = 0 = o(a^*b)$. Next we compute the $a$- and $b$-derivatives of $l$: $l_a \sim (al + b)_a \sim l$ and $l_b \sim (al + b)_b \sim 1$. Now $l_a \sim lR(a^*b)_a$ and $l_b \sim 1 \sim (a^*b)_b$. Thus $R$ is a bisimulation up to $\sim$-union and equivalence. It is not a bisimulation, since $(l_b, (a^*b)_b) \notin R$. □

*Up-to-union-context-and-equivalence.* If a relation $R \subseteq X \times X$ on the carrier of an $(F, T)$-bialgebra $(X, \alpha, \beta)$ progresses to $\overline{C_\alpha(R \cup S)}$, then $R$ is called a *bisimulation up to $S$-union, context and equivalence*. This is an important extension of bisimulation up to context because the equivalence closure allows us to relate derivatives of $R$ by equational reasoning, using $C_\alpha(R \cup S)$ in *multiple* steps.

Consider again the bialgebra $\langle T(\mathbb{R}^\omega), \alpha, \beta \rangle$ consisting of the stream system $\mathcal{S}$ of Example 1(3) and the term algebra $\alpha$. A bisimulation up to $S$-union, context and equivalence is a relation $R \subseteq T(\mathbb{R}^\omega) \times T(\mathbb{R}^\omega)$ such that for all $(t_1, t_2) \in R$: $o(t_1) = o(t_2)$ and $(t_1', t_2') \in \overline{C_\alpha(R \cup S)}$. The following is an example on $\mathcal{S}$.

*Example 5.* Suppose we are given that $\otimes$ is associative and commutative (so $\sigma \otimes \tau \sim \tau \otimes \sigma$, etc.) and that $\sigma + (-\sigma) \sim 0$ (notice that these assumptions actually hold in general [12]). Let $R = \{(\sigma \otimes \sigma^{-1}, 1) \mid \sigma \in T(\mathbb{R}^\omega), \sigma_0 \neq 0\}$. We can now establish that $R$ is a bisimulation up to $\sim$-union, context and equivalence on the coalgebra $\mathcal{S}$. First consider the initial values: $(\sigma \otimes \sigma^{-1})_0 = \sigma_0 \times \sigma_0^{-1} = 0 = \sigma_0 \times (\sigma_0)^{-1} = 1 = 1_0$. Next we relate the derivatives by $\overline{C_\alpha(R \cup \sim)}$:

$$
\begin{aligned}
(\sigma \otimes \sigma^{-1})' = \sigma' \otimes \sigma^{-1} + \sigma \otimes (\sigma^{-1})' &= \sigma' \otimes \sigma^{-1} + \sigma \otimes (-\sigma' \otimes (\sigma^{-1} \otimes \sigma^{-1})) \\
C_\alpha(\sim)^*\ (\sigma' \otimes \sigma^{-1}) + (-(\sigma' \otimes \sigma^{-1}) &\otimes (\sigma \otimes \sigma^{-1})) \\
C_\alpha(R \cup \sim)\ (\sigma' \otimes \sigma^{-1}) + (-(\sigma' \otimes \sigma^{-1}) &\otimes 1)\ C_\alpha(\sim)^*\ 0 = 1'
\end{aligned}
$$

where $C_\alpha(\sim)^*$ denotes the transitive closure of $C_\alpha(\sim)$; in the above we apply multiple substitutions of terms for bisimilar terms. Notice that $R$ is not a

bisimulation; here, establishing a bisimulation-up-to is much easier than finding a bisimulation which contains $R$. □

For the special case $S = \emptyset$, bisimulation up to $f$ is called *bisimulation up to context and equivalence*, or *bisimulation up to congruence*. Finally notice that bisimulation up to $\sim$-union, context and equivalence generalizes the notion of *bisimulation up to context and bisimilarity* (see, e.g., [9]), in a similar way as up-to-union-and-equivalence generalizes up-to-bisimilarity.

## 4 Soundness

In the above examples, intuitively, establishing that a relation $R$ is a bisimulation-up-to should imply that $R$ indeed contains only bisimilar pairs, i.e., $R \subseteq \sim$. However this depends on the instance of bisimulation-up-to under consideration. For example, one function $f : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ which, in general, does not have this property is $f(R) = X \times X$, where *any* derivatives are related. But more subtly, for up-to-context, this depends on the bialgebraic structure.

**Definition 2.** *Let $\alpha : X \to FX$ be a coalgebra, and $f : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$ a function. If $R \subseteq \sim$ for any relation $R \subseteq X \times X$ which is a bisimulation up to $f$, then we say bisimulation up to $f$ is* sound for $\alpha$. *If this holds for any coalgebra, then we say bisimulation up to $f$ is* sound.

Bisimulation up to identity is sound, of course. If $S$ is a relation which contains only bisimilar pairs, then bisimulation up to union with $S$ is sound.

**Proposition 1.** *For any coalgebra $\alpha : X \to FX$, and for any relation $S \subseteq X \times X$: if $S \subseteq \sim$ then bisimulation up to union with $S$ is sound.*

*Proof.* If $R$ is a bisimulation up to union with $S$ then it is also a bisimulation up to union with $\sim$. For any such $R$, $R \cup \sim$ is a bisimulation. □

As a consequence of the above proposition, Example 3 contains a full proof that $1^*$ and $1$ are bisimilar, given the knowledge that $0 \cdot 1^*$ is bisimilar to $0$. From [2] we obtain a sufficient condition for the soundness of bisimulation up to context:

**Proposition 2 ([2], Corollary 4.3.8).** *If $(X, \alpha, \beta)$ is a $\lambda$-bialgebra for a distributive law $\lambda$ of a monad $T$ over a functor $F \times Id$, then bisimulation up to context is sound for $(X, \beta)$.*

Bisimulation up to bisimilarity is *not* sound in general. This is illustrated by the following example, which is based on an example from [1] (introduced there to show that bisimulation and behavioural equivalence (see Section 5) do not coincide, in general).

*Example 6.* Define the functor $F : \mathsf{Set} \to \mathsf{Set}$ as $FX = \{(x_1, x_2, x_3) \in X^3 \mid |\{x_1, x_2, x_3\}| \leq 2\}$ and $F(f)(x_1, x_2, x_3) = (f(x_1), f(x_2), f(x_3))$. Consider the $F$-coalgebra with states $X = \{0, 1, 2, \tilde{0}, \tilde{1}\}$ and transition structure $\{0 \mapsto (0, 1, 0), 1 \mapsto (0, 0, 1), \tilde{0} \mapsto (0, 0, 0), \tilde{1} \mapsto (1, 1, 1), 2 \mapsto (2, 2, 2)\}$. Then $0 \not\sim 1$. To see

this, note that in order for the pair $(0,1)$ to be contained in a bisimulation $R$, there must be a transition structure on this relation which maps $(0,1)$ to $((0,0),(1,0),(0,1))$. But this triple can not be in $FR$, because it contains three different elements. However, it is easy to show that $0 \sim 2$ and $1 \sim 2$: the relation $\{(0,2),(1,2)\}$ is a bisimulation. Now consider the relation $S = \{(\tilde{0},\tilde{1}),(2,2)\}$. We can define a transition structure $\gamma : S \to FS$ as follows:

$$(\tilde{0},\tilde{1}) \mapsto ((0,1),(0,1),(0,1)) \qquad (2,2) \mapsto ((2,2),(2,2),(2,2))$$

But $0 \sim 2\ S\ 2 \sim 1$ (and $2 \sim S \sim 2$) so $S$ is a bisimulation up to bisimilarity. Thus if up-to-bisimilarity is sound, then $S \subseteq \sim$ so $0 \sim 1$, which is a contradiction. □

Bisimulation up to equivalence is not sound either, which can be shown by a similar argument. While the above counterexample is based on a somewhat "unrealistic" functor, in [3, Figure 1] there is an example of a weighted automaton, which is easily turned into another counterexample, showing that bisimulation up to bisimilarity (or up to equivalence) is not sound in the case of weighted automata. Fortunately, for a large class of systems (even including certain types of weighted automata) bisimulation up to equivalence *is* sound, namely for those modeled by functors which preserve weak pullbacks; we develop this result in Section 6.

## 5 Behavioural Equivalence

In this section we recall a notion of equivalence which is in general weaker than bisimulation, but in many cases (depending on the type of system) the two notions coincide. Suppose $\alpha : X \to FX$ is a coalgebra. Two states $s,t$ are called ($F$-)*behaviourally equivalent* if there exists a homomorphism $f : X \to Y$ into some other coalgebra, such that $f(s) = f(t)$. This notion is pointwise extended to relations $R \subseteq X \times X$, i.e., $R$ is an ($F$-)*behavioural equivalence* if $R \subseteq \ker(f)$. In the sequel we will base ourselves on a more concrete characterization of behavioural equivalence, which can be seen to be a slight variation of a characterization from [5] (which considers only equivalence relations in this context, but the generalization to arbitrary relations is easy).

**Proposition 3 ([5], Lemma 4.12).** *Let $\alpha : X \to FX$ be an $F$-coalgebra. A relation $R \subseteq X \times X$ is a behavioural equivalence iff the following diagram commutes:*

$$R \underset{\pi_2}{\overset{\pi_1}{\rightrightarrows}} X \xrightarrow{\alpha} FX \xrightarrow{Fq} F(X/\overline{R})$$

*where $q$ is the quotient map of $\overline{R}$.*

*Example 7.* Let $\alpha : X \to X + \mathbb{1}$ be a deterministic system with termination, and $R \subseteq X \times X$ a relation with quotient map $q$; then $Fq = q + id$. According to the above proposition, $R$ is a behavioural equivalence if for all $(x,y) \in R$, either $x \downarrow$ and $y \downarrow$, or $q(x') = q(y')$. The latter case is equivalent to $x'\ \overline{R}\ y'$. □

In [5], behavioural equivalences which are equivalence relations are called *congruences*, after [1], which introduced this coalgebraic notion of congruence. We chose not to use the term "congruence" to avoid confusion with the well-known concept from universal algebra. Behavioural equivalence coincides with the notion of *pre-congruence* from [1]. We proceed to recall from that paper the precise relation with bisimulation.

**Proposition 4 ([1]).** *Any $F$-bisimulation is an $F$-behavioural equivalence. If $F$ preserves weak pullbacks, then the equivalence closure of any $F$-behavioural equivalence is an $F$-bisimulation.*

For a fixed coalgebra $\alpha$ the greatest behavioural equivalence exists [1], and it is denoted by $\approx_\alpha$ or simply $\approx$ if $\alpha$ is clear from the context. From the above proposition we immediately obtain that for weak pullback preserving functors, the greatest behavioural equivalence and the greatest bisimulation coincide.

Many interesting functors used to model systems coalgebraically actually do preserve weak pullbacks, including all functors introduced in Example 1. One example of a relevant functor that does *not* preserve weak pullbacks is that of weighted automata over $\mathbb{R}$. Figure 1 of [3] is an example of a weighted automaton containing states which are behaviourally equivalent, but not bisimilar. In fact, the notion of equivalence of weighted automata chosen in [3] is indeed behavioural equivalence, which coincides with so-called *weighted bisimilarity*.

Example 7 on the one hand illustrates how to obtain a proof method for behavioural equivalence for a given functor $F$, using Proposition 3. On the other hand, it suggests that in this proof method, we can reason up to equivalence. Indeed, while bisimulation up to equivalence is problematic (not sound) in general, we will see in the next section that for behavioural equivalence this comes quite naturally.

## 6 Soundness via Behavioural Equivalence-Up-To

Given a function $f : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$, we define behavioural equivalence up to $f$ as a generalization of the characterization given in Proposition 3. The quotient map of $\overline{f(R)}$ is now used to relate derivatives, instead of the quotient map of $\overline{R}$.

**Definition 3.** *Let $\alpha : X \to FX$ be an $F$-coalgebra and $R \subseteq X \times X$. Let $f : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$. $R$ is a* behavioural equivalence up to $f$ *if the following diagram commutes:*

$$R \underset{\pi_2}{\overset{\pi_1}{\rightrightarrows}} X \xrightarrow{\alpha} FX \xrightarrow{Fq} F(X/\overline{f(R)})$$

*where $q$ is the quotient map of $\overline{f(R)}$.*

For example, for a deterministic system $\alpha : X \to X + \mathbb{1}$, a relation $R \subseteq X \times X$ is a behavioural equivalence up to $f$ whenever for all $(x, y) \in R$ either $x \downarrow$ and $y \downarrow$ or $x' \ \overline{f(R)} \ y'$. In order to proceed we define soundness of behavioural equivalence-up-to.

**Definition 4.** *Let $\alpha : X \to FX$ be a coalgebra, and $f : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$. If $R \subseteq \approx$ for any relation $R \subseteq X \times X$ which is a behavioural equivalence up to $f$, then we say behavioural equivalence up to $f$ is* sound for $\alpha$. *If this holds for any coalgebra, then we say behavioural equivalence up to $f$ is* sound.

Behavioural equivalence up to union, equivalence, context etc. are defined in the same way as for bisimulation-up-to.

**Lemma 1.** *If $(X, \alpha, \beta)$ is a $\lambda$-bialgebra for a distributive law $\lambda$ of a finitary*[1] *monad $T$ over a functor $F \times Id$, then behavioural equivalence up to context is sound for $(X, \beta)$.*

We obtain our main result:

**Theorem 1.** *Suppose $S \subseteq \approx$. Then ($F$-)behavioural equivalence up to ($S$-union and) equivalence is sound. Moreover, if $(X, \alpha, \beta)$ is a $\lambda$-bialgebra for a distributive law $\lambda$ of a finitary monad $T$ over a functor $F \times Id$, then ($F$-)behavioural equivalence up to ($S$-union,) context (and equivalence) is sound for $(X, \beta)$.*

*Proof.* Let $f(R) = \overline{C_\alpha(R \cup S)}$. If $R$ is a behavioural equivalence up to $f$, then it is also simply a bisimulation up to $f'$, where $f'(R) = C_\alpha(R \cup S)$. If $S \subseteq \approx$, then $R$ is a behavioural equivalence up to $f''(R) = C_\alpha(R \cup \approx)$ as well. Then $R \cup \approx$ is behavioural equivalence up to context, so $R \subseteq \approx$ by Lemma 1. □

Behavioural equivalence-up-to is related to bisimulation-up-to as follows:

**Lemma 2.** *Let $f : \mathcal{P}(X \times X) \to \mathcal{P}(X \times X)$. Then (1) any bisimulation up to $f$ is also a behavioural equivalence up to $f$, and (2) if $F$ preserves weak pullbacks, then soundness of behavioural equivalence up to $f$ implies soundness of bisimulation up to $f$.*

From the above lemma and Theorem 1 we immediately obtain the following:

**Corollary 1.** *If $F$ preserves weak pullbacks, then Theorem 1 holds for bisimulation-up-to as well.*

Consequently, all the examples of bisimulations-up-to in Section 3 contain actual proofs of bisimilarity, based on smaller relations than any other relations needed to establish bisimilarity without these techniques.

## 7 Conclusions and Future Work

We generalized bisimulation-up-to to the theory of coalgebras. By extending the theory to behavioural equivalence, we established the soundness of bisimulation up to union and equivalence for systems modeled by functors which preserve weak pullbacks. For any coalgebra with an algebraically structured state space

[1] A monad is finitary if its underlying functor $T$ preserves filtered colimits. If $T$ is a term monad, this means there may be infinitely many operations, but each of them must have finite arity.

which forms a $\lambda$-bialgebra for a distributive law $\lambda$, we have shown that bisimulation up to union, context and equivalence is sound. Future work includes a generalization to other categories, investigation of instances of bisimulation-up-to for concrete types of systems (e.g., [4]), and the integration with the systematic treatment of enhancements of [9].

## References


1. Aczel, P., Mendler, N.: A Final Coalgebra Theorem. In: Dybjer, P., Pitts, A.M., Pitt, D.H., Poigné, A., Rydeheard, D.E. (eds.) Category Theory and Computer Science. LNCS, vol. 389, pp. 357–365. Springer, Heidelberg (1989)
2. Bartels, F.: On generalised coinduction and probabilistic specification formats. PhD thesis, CWI, Amsterdam (2004)
3. Bonchi, F., Bonsangue, M., Boreale, M., Rutten, J., Silva, A.: A coalgebraic perspective on linear weighted automata. Inf. Comput. 211, 77–105 (2012)
4. Bonchi, F., Pous, D.: Checking NFA equivalence with bisimulations up to congruence. In: POPL (to appear, 2013)
5. Gumm, H.P.: Elements of the general theory of coalgebras. In: LUATCS 1999 Rand Afrikaans University, South Africa (1999)
6. Klin, B.: Bialgebras for structural operational semantics: An introduction. TCS 412(38), 5043–5069 (2011)
7. Lenisa, M.: From set-theoretic coinduction to coalgebraic coinduction: some results, some problems. ENTCS 19, 2–22 (1999)
8. Milner, R.: Calculi for synchrony and asynchrony. TCS 25, 267–310 (1983)
9. Pous, D., Sangiorgi, D.: Enhancements of the bisimulation proof method. In: Advanced Topics in Bisimulation and Coinduction, pp. 233–289. Cambridge University Press (2012)
10. Rutten, J.: Automata and Coinduction (An Exercise in Coalgebra). In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 194–218. Springer, Heidelberg (1998)
11. Rutten, J.: Universal coalgebra: a theory of systems. TCS 249(1), 3–80 (2000)
12. Rutten, J.: Behavioural differential equations: a coinductive calculus of streams, automata, and power series. TCS 308(1-3), 1–53 (2003)
13. Sangiorgi, D.: On the bisimulation proof method. Math. Struct. Comp. Sci. 8(5), 447–479 (1998)
14. Zhou, X., Li, Y., Li, W., Qiao, H., Shu, Z.: Bisimulation Proof Methods in a Path-Based Specification Language for Polynomial Coalgebras. In: Ueda, K. (ed.) APLAS 2010. LNCS, vol. 6461, pp. 239–254. Springer, Heidelberg (2010)

# A Model Transformation Language Based on Logic Programming*

Jesús M. Almendros-Jiménez and Luis Iribarne

Dpto. de Lenguajes y Computación
Universidad de Almería
04120-Almería, Spain
{jalmen,luis.iribarne}@ual.es

**Abstract.** In this paper we present a model transformation language based on logic programming. The language, called PTL (*Prolog based Transformation Language*), can be considered as a hybrid language in which ATL-style rules are combined with logic rules for defining transformations. The proposal has been implemented so that a Prolog program is automatically obtained from a PTL program. We have equipped our language with debugging and tracing capabilities which help developers to detect programming errors in PTL rules.

## 1 Introduction

Several transformation languages and tools have been proposed in the literature (see [3] for a survey). The most relevant is the language *ATL (Atlas Transformation Language)* [7], a *domain-specific language* for specifying model-to-model transformations. ATL is a hybrid language, and provides a mixture of declarative and imperative constructs. The declarative part of ATL is based on rules. Such rules define a source pattern matched over source models and a target pattern that creates target models for each match.

In this work we present a language, called PTL (*Prolog based Transformation Language*), that can be considered as a hybrid language in which ATL-style rules are combined with logic rules for defining transformations. ATL-style rules are used to define mappings from source models to target models while logic rules are used as helpers. From a theoretical point of view, we have studied a declarative semantics for PTL. The declarative semantics interprets meta-models and identifies models that conform to meta-models while PTL rules interpretation provides semantics to the main ATL constructors. From a practical point of view, our proposal can be intended as an application of logic programming to a context in which rule-based systems are required. Our language provides the

* This work has been supported by the Spanish Ministry MICINN and Ingenieros Alborada IDI under grant TRA2009-0309. This work has been also supported by the EU (FEDER) and the Spanish Ministry MICINN under grants TIN2010-15588, TIN2008-06622-C03-03, and the JUNTA ANDALUCIA (proyecto de excelencia) ref. TIC-6114.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 382–394, 2013.

elements involved in model transformation: meta-models handling and mapping rules.

The adoption of ATL style syntax facilitates the definition of mappings: basically, objects are mapped to objects of given models. Nevertheless, we retain logic programming as basis, for instance, by defining helpers with Prolog code instead of OCL code, adopted by ATL. It makes that our framework combines model transformation and logic languages. One of the main ends of helpers in ATL is to serve as query language against the source models. In a logic programming based approach, logic (i.e., Prolog style) rules serve as query language. MOF (Meta Object Facility) is considered in our approach as meta-meta-model, and source and target meta-models have to be defined in terms of the MOF meta-model. With this aim, our language can be also used for defining transformations from (and to) the MOF meta-model to (from) the source (target) meta-models.

The proposal has been implemented so that a Prolog program is automatically obtained from a PTL program. Hence, PTL makes use of Prolog as transformation engine. The encoding of PTL programs with Prolog is based on a Prolog library for handling meta-models. In order to execute a transformation, source models have to be stored in XMI format. Source models are loaded from XMI files, and target models are obtained in XMI format. So, our approach can be integrated with XMI-compliant tools. We have equipped our language with debugging and tracing capabilities which help developers to detect programming errors in PTL rules. Debugging detects PTL rules that cannot be applied to source models, while tracing shows rules and source model elements used to obtain a given target model element. The Prolog-based compiler, interpreter, debugger and tracer of our PTL language have been developed under *SWI-Prolog*. The source code of PTL can be downloaded from `http://indalog.ual.es/mdd`.

The structure of the paper is as follows. Section 2 will introduce PTL. Section 3 will describe the encoding of PTL with Prolog. Section 4 will give some details about the PTL interpreter, debugger and tracer. Section 5 will review related work. Finally, Section 6 will conclude and present future work.

## 2 Prolog-Based Transformation Language (PTL)

A PTL program consists of (a) **meta-model definitions** (source and target meta-models), (b) **mapping rules** and (c) **helpers**. Meta-model definitions define meta-model elements: class and roles together with typed attributes for classes, and the cardinality of roles. Mapping rules have the form **rule** *rule_ name* **from** *pointers* **where** *boolean_ condition* **to** *objects*. Helpers are defined by Prolog rules. The syntax of elements (a) and (b) is shown in Figure 1, where *mm_ name*, *class_ name*, *attribute_ name*, *role_ name*, *type_ name*, *rule_ name*, *pointer_ name* and *helper* are user defined names and *value* can be any Boolean, integer, string, etc. A mapping rule maps a set of objects of the source model into a set of objects of the target model. The rule can be conditioned by a Boolean condition, including equality (i.e. ==) and inequality (i.e. =\=), and the *and* operator. The rule condition is marked with *where*. Objects of target models

```
metamodel := metamodel '(' mm_name '[' definitions ']' ')'
definitions := definition | definition definitions
definition := class '(' class_name '[' attributes ']' ')' |
              role '(' role_name ',' class_name ',' class_name ',' lower ',' upper ')'
attributes := attribute | attribute ',' attributes
attribute := atttribute_name '(' type_name ')'
rule := rule rule_name from pointers [where condition ] to objects
pointers := pointer | '(' pointer,...,pointer ')'
condition := bcondition | condition and condition
bcondition := access == access | access =\= access
pointer := pointer_name : metamodel ! class
objects := object,...,object
object := pointer '(' binding,...,binding ')'
access := value | pointer_name | pointer_name@attribute |
          pointer_name@role@attribute | helper '(' access,...,access ')' |
          resolveTemp'(' '(' access,...,access ')', pointer_name')' |
          sequence'('[']'access,...,access'['']')'
binding := attribute <- access | role <- access
```

**Fig. 1.** PTL syntax

are defined assigning values to attributes and roles, and they can make use in their definition of attribute and role values of the source models, together with *resolveTemp*, helpers and the *sequence* construction. The *resolveTemp* function permits to assign objects created by another rule. With this aim, *resolveTemp* has as parameters the objects to which the rule is applied, and the name of the object created by the rule.

### 2.1 Declarative Semantics of PTL

PTL has a declarative semantics whose basis is the encoding of PTL with logic programming. The declarative semantics is based on the interpretation of meta-models and PTL rules.

Assuming a set $\mathcal{D}$ of domain values partitioned into $d$ domains $d_1, \ldots, d_d$, then a meta-model $\mathcal{MM}$ is a quadruple $\mathcal{MM} = (\mathcal{C}, \mathcal{A}, \mathcal{R}, \mathcal{H})$ where $\mathcal{C}$ is a set of class names $c_1, \ldots, c_n$, $\mathcal{A}$ is a set attribute names, $\mathcal{R}$ is a set of role names $r_1, \ldots, r_m$ and $\mathcal{H}$ is a set of helper names $h_1, \ldots, h_s$; where each class name $c_i$ has an associated set of attribute names $att^i_1, \ldots, att^i_{l_i} \in \mathcal{A}$, where $l_i$ is the number of attributes of the class $c_i$. In addition, role names $r_p$ have a defined domain (respectively, range), denoted by $domain(r_p)$, (respectively, $range(r_p)$), which is a class name of $\mathcal{C}$. Attributes and helpers have also a domain and range, where $att^i_j \in \mathcal{A}$ has as domain $c_i$ and range some of element of $\mathcal{D}$; and $h_w : c_{w_1} \ldots c_{w_t} \rightarrow c_{w_{(t+1)}}$ where $w_i \in \{1, \ldots, n\}$. Finally, roles $r_p$ have an associated lower and upper cardinality, $min(r_p) \in I\!N$ and $max(r_p) \in I\!N \cup \infty$.

Now, a model $\mathcal{M}$ is a quintuple $\mathcal{M} = (\mathcal{C}^{\mathcal{M}}, \mathcal{A}^{\mathcal{M}}, \mathcal{R}^{\mathcal{M}}, \mathcal{H}^{\mathcal{M}}, \mathcal{O}^{\mathcal{M}})$, which instantiates a meta-model, that is, it is an interpretation $\mathcal{C}^{\mathcal{M}}$ of the elements of $\mathcal{C}$, an interpretation $\mathcal{A}^{\mathcal{M}}$ of the elements of $\mathcal{A}$, an interpretation $\mathcal{R}^{\mathcal{M}}$ of the elements of $\mathcal{R}$, and an interpretation $\mathcal{H}^{\mathcal{M}}$ of the elements of $\mathcal{H}$, together with a domain $\mathcal{O}^{\mathcal{M}}$, which is a set of object names. $\mathcal{C}^{\mathcal{M}}$ is an interpretation of the class names $c^{\mathcal{M}}_1, \ldots, c^{\mathcal{M}}_n$, which are disjoint subsets of $\mathcal{O}^{\mathcal{M}}$. Roles are interpreted as a set of (partial) functions $r^{\mathcal{M}}_1, \ldots, r^{\mathcal{M}}_m$ from $\mathcal{O}^{\mathcal{M}}$ to subsets of $\mathcal{O}^{\mathcal{M}}$, and attributes are interpreted as a set of (partial) functions $att^{<\mathcal{M},i>}_j$ from $\mathcal{O}^{\mathcal{M}}$ to elements of $\mathcal{D}$.

(1) $[\![$ **rule** rn **from** $ps$ **where** $bc$ **to** $os$ $]\!]^{\mathcal{M}} = \cup_{\{\overline{v}\in\overline{C},\overline{(p,C)}\in<<ps>>^{\mathcal{M}},[\![bc]\!]^{\mathcal{M}}_{(p,v)}\ is\ true\}}[\![os]\!]^{\mathcal{M}}_{(p,v)}$

(2) $<< pn : MM!c >>^{\mathcal{M}} = (pn, c^{\mathcal{M}})$ whenever $c \in \mathcal{C}$ of $\mathcal{MM}$

(3) $<< (p_1, ..., p_n) >>^{\mathcal{M}} = << p >>^{\mathcal{M}}$

(4) $[\![bc_1$ **and** $bc_2]\!]^{\mathcal{M}}_{(p,v)}$ if $[\![bc_1]\!]^{\mathcal{M}}_{(p,v)}$ and $[\![bc_2]\!]^{\mathcal{M}}_{(p,v)}$ then true else false

(5) $[\![ac_1 == ac_2]\!]^{\mathcal{M}}_{(p,v)}$ if $[\![ac_1]\!]^{\mathcal{M}}_{(p,v)} = [\![ac_2]\!]^{\mathcal{M}}_{(p,v)}$ then true else false

(6) $[\![ac_1 = \backslash = ac_2]\!]^{\mathcal{M}}_{(p,v)}$ if $[\![ac_1]\!]^{\mathcal{M}}_{(p,v)} = [\![ac_2]\!]^{\mathcal{M}}_{(p,v)}$ then false else true

(7) $[\![pn : MM!c(b_1, ..., b_n)]\!]^{\mathcal{M}}_{(p,v)} = \mathcal{M}' \cup_{1\le i\le n} [\![b_i]\!]^{<\mathcal{M},o>}_{(p,v)}$ where $\mathcal{M}' = (\{c^{\mathcal{M}'}\}, \{\}, \{\}, \{\}, \{o\})$, $c^{\mathcal{M}'} = \{o\}$ and $o = gen\_id(\overline{v}, pn)$, whenever $c \in \mathcal{C}$ of $\mathcal{MM}$

(8) $[\![o_1, ..., o_n]\!]^{\mathcal{M}}_{(p,v)} = \cup_{i=1,...,n}[\![o_i]\!]^{\mathcal{M}}_{(p,v)}$

(9) $[\![v]\!]^{\mathcal{M}}_{(p,v)} = v$

(10) $[\![p_i]\!]^{\mathcal{M}}_{(p,v)} = v_i$

(11) $[\![p_i@att]\!]^{\mathcal{M}}_{(p,v)} = att^{\mathcal{M}}(v_i)$

(12) $[\![p_i@r@att]\!]^{\mathcal{M}}_{(p,v)} = att^{\mathcal{M}}(r^{\mathcal{M}}(v_i))$

(13) $[\![h(ac_1, ..., ac_n)]\!]^{\mathcal{M}}_{(p,v)} = h^{\mathcal{M}}([\![ac_1]\!]^{\mathcal{M}}_{(p,v)}, \ldots, [\![ac_n]\!]^{\mathcal{M}}_{(p,v)})$

(14) $[\![$**resolveTemp**$((ac_1, ..., ac_n), p)]\!]^{\mathcal{M}}_{(p,v)} = o$ where $([\![ac_1]\!]^{\mathcal{M}}_{(p,v)}, \ldots, [\![ac_n]\!]^{\mathcal{M}}_{(p,v)}) \rightarrow^{\mathcal{P}}_{p} o$

(15) $[\![$**sequence**$([ac_1, ..., ac_n])]\!]^{\mathcal{M}}_{(p,v)} = \{[\![ac_1]\!]^{\mathcal{M}}_{(p,v)}, \ldots, [\![ac_n]\!]^{\mathcal{M}}_{(p,v)}\}$

(16) $[\![att <- acc]\!]^{<\mathcal{M},o>}_{(p,v)} = \mathcal{M}'$, where $\mathcal{M}' = (\{\}, \{att^{\mathcal{M}'}\}, \{\}, \{\}, \{\})$, $att^{\mathcal{M}'}(o) = [\![acc]\!]^{\mathcal{M}}_{(p,v)}$

(17) $[\![r <- acc]\!]^{<\mathcal{M},o>}_{(p,v)} = \mathcal{M}'$, where $\mathcal{M}' = (\{\}, \{\}, \{r^{\mathcal{M}'}\}, \{\}, \{\})$, $r^{\mathcal{M}'}(o) = [\![acc]\!]^{\mathcal{M}}_{(p,v)}$

**Fig. 2.** Declarative Semantics of PTL

Finally helpers are interpreted as (partial) functions $h^{\mathcal{M}}_{w} : c^{\mathcal{M}}_{w_1} \ldots c^{\mathcal{M}}_{w_t} \rightarrow c^{\mathcal{M}}_{w_{(t+1)}}$ $w_i \in \{1, \ldots, n\}$.

Now, a model $\mathcal{M}$ conforms to $\mathcal{MM}$ whenever $r^{\mathcal{M}}_{p}(o_i) \in c^{\mathcal{M}}_{j}$ and $min(r_p) \le \#r^{\mathcal{M}}_{p}(o_i) \le max(r_p)$, for each $r_p \in \mathcal{R}$ and $o_i \in c^{\mathcal{M}}_{i}$, where $domain(r_p) = c_i$ and $range(r_p) = c_j$; and, in addition, $att^{<\mathcal{M},i>}_{j}(o_i) \in d_k$ whenever $o_i \in c^{\mathcal{M}}_{i}$ where $c_i = domain(att^{i}_{j})$ and $range(att^{i}_{j}) = d_k$.

Given two models $\mathcal{M}_1$ and $\mathcal{M}_2$ of a meta-model $\mathcal{MM}$, we can build the union of models as follows $c^{\mathcal{M}_1 \cup \mathcal{M}_2} = c^{\mathcal{M}_1} \cup c^{\mathcal{M}_2}$ for each $c \in \mathcal{C}$ of $\mathcal{MM}$, $\mathcal{A}^{\mathcal{M}_1 \cup \mathcal{M}_2}$, $\mathcal{R}^{\mathcal{M}_1 \cup \mathcal{M}_2}$ and $\mathcal{H}^{\mathcal{M}_1 \cup \mathcal{M}_2}$ are the union of the graphs of each function, and $\mathcal{O}^{\mathcal{M}_1 \cup \mathcal{M}_2} = \mathcal{O}^{\mathcal{M}_1} \cup \mathcal{O}^{\mathcal{M}_2}$. Let us remark that the union of two models that conform to a meta-model does not necessarily conform to the meta-model due to cardinality restrictions. Model unions will be used in the definition of the declarative semantics of PTL.

Now, we can provide declarative semantics to PTL as follows. Given a *source model* $\mathcal{M}$, a PTL program $\mathcal{P}$ with rules $r_1, \ldots, r_n$, *defines the target model* $\mathcal{M}' = \cup_{1\le i\le n}[\![r_i]\!]^{\mathcal{M}}$, where $[\![r]\!]^{\mathcal{M}}$ denotes the model obtained by $r$ from $\mathcal{M}$, which is defined in Figure 2.

In Figure 2, the main cases are the following. Case (1) defines the semantics of a PTL rule. The semantics is defined as the union of the target objects obtained from bindings to source objects that satisfy the Boolean condition. Cases (2) and (3) bind pointers to object names of the source model. Finally, case (7) creates the target object and attribute and role bindings are obtained in cases (16) and (17). The expression $[\![e]\!]^{\mathcal{M}}_{(p,v)}$ where $\overline{(p,v)} \equiv (p_1, v_1), \ldots, (p_n, v_n)$ denotes the value of $e$ in the model $\mathcal{M}$, with respect to the assignments $p_1 \rightarrow v_1, \ldots, p_n \rightarrow vn$ of pointers to object names of the model $\mathcal{M}$; the expression $[\![acc]\!]^{<\mathcal{M},o>}_{(p,v)}$ denotes the

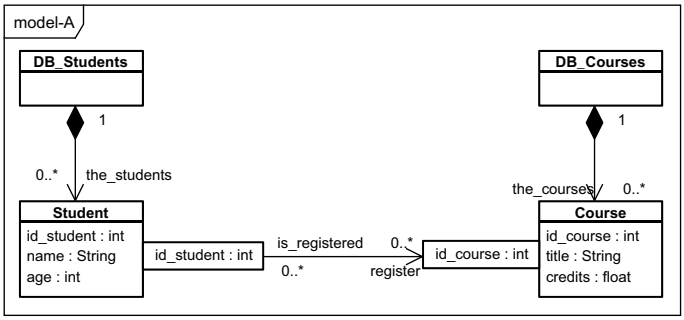



**Fig. 3.** Entity-relationship modeling of the Case Study

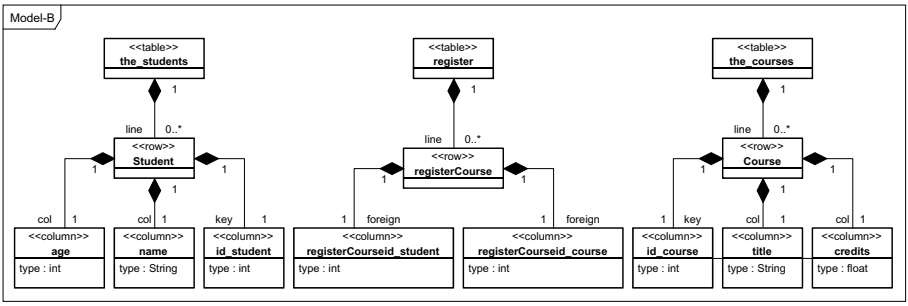



**Fig. 4.** Relational modeling of the Case Study

value of *acc* in the object $o$ with respect to the assignments $p_1 \rightarrow v_1, \ldots, p_n \rightarrow vn$ in $\mathcal{M}$; the expression $(o_1, \ldots, o_k) \rightarrow_p^{\mathcal{P}} o$, used in the *resolveTemp* definition (case (14)), denotes that a sequence of object names $(o_1, \ldots, o_k)$, $o_i \in \mathcal{O}^{\mathcal{M}}$, $1 \leq i \leq k$ that is transformed into the object $o$ of pointer $p$ with regard to $\mathcal{P}$, that is, there exists $(rule\ rn\ from\ ps\ to\ obs) \in \mathcal{P}$, $ps \equiv p_1, \ldots, p_k$ such that $o \in O^{\mathcal{M}'}$, $\mathcal{M}' = [\![obs]\!]^{\mathcal{M}}_{(p,o)}$ and $gen\_id(\bar{o}, p) = o$; finally, $gen\_id$ is a bijective function that takes $n$ object names of $\mathcal{M}$ and a pointer $p$ and returns a object name of $\mathcal{M}'$.

## 2.2 Example of Transformation

Now, we would like to show an example of transformation. The model of Figure 3 represents the modeling of a database. We will call this kind of modeling as *"entity-relationship"* modeling of a database in contrast to the model of Figure 4 which will be called *"relational"* modeling of a database. The entity-relationship modeling of Figure 3 can be summarized as follows. Data are represented by classes (i.e., *Student* and *Course*), including attributes; stores are defined for each data (i.e., *DB_Students* and *DB_Courses*); relations are represented by associations and relation names are association names. Besides, roles are defined (i.e., *the_students*, *the_courses*, *is_registered* and *register*). Relationships can

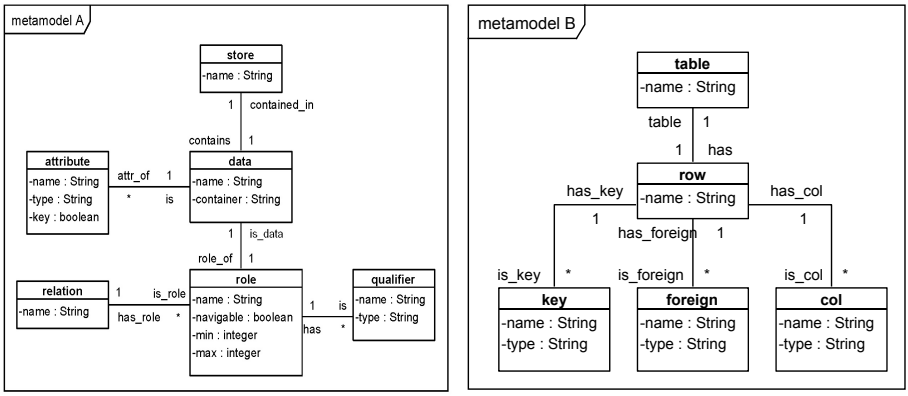



**Fig. 5.** Meta-model of the Source/Target Models

be adorned with qualifiers and navigability. Qualifiers are used to specify the key attributes of each entity being foreign keys of the corresponding association. Figure 4 shows the relational modeling of the same database. Tables are composed of rows, and rows are composed of columns with three roles: cols, keys and foreign keys. Figure 5 represents the meta-models of both types of modeling. In the first case, *DB_ Students* and *DB_ Courses* are instances of the class *store*, while *Student* and *Course* are instances of the class *data*, and the attributes of *Student* class and *Course* class are instances of the class *attribute*. In the second case, tables and rows of the target model are instances of the corresponding classes, and the same can be said about *key*, *col* and *foreign* classes.

Now, the problem of model transformation is how to transform a class diagram of the type A (like Figure 3) into a class diagram of type B (like Figure 4). In other words, the PIM conceptual model is transformed into the PSM relational model. The transformation generates two tables called *the_ students* and *the_ courses* each including three columns that are grouped into rows. The table *the_ students* includes for each student the attributes of *Student* of Figure 3. The same can be said for the table *the_ courses*. Given that the association between *Student* and *Course* is navigable from *Student* to *Course*, a table of pairs is generated to represent the assignments of students to courses, using the *registerCourse* as name of the cited table. The columns *registerCourseid_ student* and *registerCourseid_ course* taken from qualifiers, play the role of foreign keys which are represented by role *foreign* in the associations of Figure 4.

### 2.3 Example in PTL

In PTL, **meta-model definitions** are given in order to define the transformation (see Figure 6 in the case of the source model). In addition **mapping rules** are defined in PTL (see Figure 7). The rule *table2_ er2rl* defines the tables and rows obtained from navigable roles (in the case study, *register* and *registerCourse*). The name of the table is the name of the role, and the name of the row is built from

```
metamodel(er,
[
   class(data, [name,container]),
         class(store, [name]),
         class(attribute, [name,type,key]),
         class(relation, [name]),
         class(role, [name,navigable,min,max]),
         class(qualifier, [name,type]),
         role(contains,store,data,"1","1"),
         role(contained_in,data,store,"1","1"),
         role(attr_of,data,attribute,"0","*"),
         role(is,attribute,data,"1","1"),
         role(has_role,role,relation,"1","1"),
         role(is_role,relation,role,"1","*"),
         role(has,qualifier,role,"1","1"),
         role(is,role,qualifier,"0","*"),
         role(is_data,role,data,"1","1"),
         role(role_of,data,role,"1","1")
]
).
```

**Fig. 6.** Source Metamodel

```
rule table2_er2rl from
p:er!role where (p@navigable==true and p@max=="*") to
 (t:rl!table(
     name <- p@name,
     has <- r
           ),
  r:rl!row(
     name <- concat(p@name,p@is_data@name),
     table <-t,
     is_foreign <- sequence([resolveTemp((p@is,p),f1),inverse_qualifier(p)])
         )
  ).
rule foreign2_er2rl from
(p:er!qualifier,q:er!role) where (p@has==q and q@navigable==false) to
        (f2:rl!foreign(
             name <-concat(concat(q@name,q@is_data@name),p@name),
             type <- p@type,
             has_foreign <- inverse_row(p)
                      )
         ).
```

**Fig. 7.** Examples of PTL Mapping Rules

```
inverse_row(A, E) :-
        associationEnds(er, has, A, B),
        associationEnds(er, has_role, B, C),
        associationEnds(er, is_role, C, D),
        role_navigable(er, D, true),
        resolveTemp(D, r, E).
```

**Fig. 8.** Helpers

the concatenation of the name of the role, and the name of the role end. Moreover, we have to set the role ends from (to) tables to (from) rows (i.e., *has* and *table*) together with the role *is_foreign*. The role *is_foreign* is a sequence of two

```
role_id(er, A) :-role(er, A, [name(_), navigable(_), min(_), max(_)]).
data_container(er, A, B) :-data(er, A, [name(_), container(B)]).
attribute_type(er, A, B) :-attribute(er, A, [name(_), type(B), key(_)]).
qualifier_has(er, A, B) :-associationEnds(er, has, A, B).
relation_is_role(er, A, B) :-associationEnds(er, is_role, A, B).
```

**Fig. 9.** Prolog library

elements (*registerCourseid_ student* and *registerCourseid_ course*). For this reason the *sequence* constructor is chosen. Besides, *resolveTemp* retrieves *registerCourseid_ course*, and a helper called *inverse_ qualifier* is used for the retrieval of *registerCourseid_ student*. The rule *foreign2_ er2rl* computes the foreign class *registerCourseid_ student*, which is computed from roles and qualifiers which are not navigable. The rule sets the role *has_ foreign* with a helper called *inverse_ row*, which computes the row *registerCourse*.

**Helpers** can be defined with logic rules. For instance, in Figure 8 we can see the definition of the helper *inverse_ row*. Helpers can make use of the metamodel Prolog library (see next Section) and the *resolveTemp* construction. It is worth observing that helpers are defined with the following convention: we can define helpers with several arguments, but the last one has to be the result of the helper. In other words, predicates associated to helpers work as functions. The previous convention requires that helpers in PTL mapping rules have $n$ arguments while code of helpers has $n + 1$ arguments.

## 3 Encoding with Prolog

Now, we would like to show how PTL mapping rules are encoded with Prolog rules. The schema of the encoding is as follows. Given a PTL mapping rule of the form: **rule** *rn* **from** *ps* **where** *bc* **to** *obs* where $obs \equiv object_1, \ldots, object_n$, $object_i \equiv pointer_i(binding_1, \ldots, binding_k)$ and $ps \equiv p_1, \ldots, p_m$, $p_j \equiv q_j : mm_j!class_j$ then the encoding is as follows:

(1) $object(mm_j, class_j, Var, \overline{Vars}, enc[atts], q_j) : -rn(\overline{Vars}), enc[bind\_att]$.
(2) $associationObjects(mm_j, role_p, Var_1, Var_2) : -rn(\overline{Vars}), enc[role_p{<}{-}access]$.
(3) $rn(\overline{Vars}) : -enc[bc]$.

where *bind_att* is the subset of $binding_1, \ldots, binding_k$ of bindings of the form $attribute <\!\!- access$; expressions $role_p <\!\!- access$ are each one of the remaining bindings ($p \in \{1, ..., k\}$); and finally, the element $enc[atts]$ is the encoding of attributes, and the elements $enc[bind\_att]$, $enc[role_p <\!\!- access]$ and $enc[bc]$ are the encoding of such expressions using the Prolog library for meta-models and Prolog helpers.

The predicate *object* encodes the creation of objects of the target model. The predicate *associationObjects* encodes the creation of links between objects of the target model. There are rules *object* and *associationObjects* for each object and each link created by one rule. Hence, one PTL mapping rule is encoded by a set of Prolog rules, one rule for each object that is created, and one rule for each role set by the rule.

```
(1) object(rl, foreign, L, (A, B),[name(K), type(C)], f2) :-
        foreign2_er2rl((A, B)),
        qualifier_type(er, A, C),
        qualifier_name(er, A, I),
        role_is_data(er, B, E),
        data_name(er, E, G),
        role_name(er, B, F),
        concat(F, G, H),
        concat(H, I, K),
        gen_id((A, B), f2, L).

(2) associationObjects(rl, has_foreign, C, B) :-
        foreign2_er2rl((A, D)),
        qualifier_id(er, A),
        inverse_row(A, B),
        object(rl, foreign, C, (A, D), _, f2).

(3) foreign2_er2rl((A, B)) :-
        qualifier_id(er, A),
        role_id(er, B),
        qualifier_has(er, A, B),
        role_navigable(er, B, false).
```

**Fig. 10.** Encoded Rule

With respect to the Prolog library for handling meta-models, the Figure 9 contains (some of) the predicates generated to handle the *er* metamodel of Figure 6. We have three kinds of predicates: (a) those for *accessing class attributes*, for instance, *data_container* and *attribute_type*, (b) those for *accessing roles*, for instance, *qualifier_has* and *relation_is_role*, and (c) a special kind of predicates that retrieve the *identifier* of a certain object (for instance, *role_id*). The first and third kind of predicates call predicates representing class objects, which are called as the class name, and they have as arguments: the name of the meta-model, the object identifier and a Prolog list with the attributes: each attribute is represented by a Prolog term of the form: *attribute_name(value)*. The second kind of predicates calls to a predicate called *associationEnds*, representing role end objects. The *associationEnds* element has as arguments: the name of the meta-model, the name of the role, and the identifiers of the role ends. In other words, models are stored with Prolog facts.

The Prolog library for meta-models makes possible to encode PTL rules. For instance, we can see in Figure 10 the encoding of the PTL rule *foreign2_er2rl* of Figure 7. Finally, *resolveTemp* construction of PTL can be easily defined with our encoding: `resolveTemp(B, C, A) :- object(_, _, A, B, _, C).`

The encoding has taken into account the declarative semantics defined for PTL. The following theorem establishes the soundness and completeness of the encoding (due to the lack of space we only provide an sketch of the proof). Given a PTL program $\mathcal{P}$ we denote by $\mathcal{I}^{\mathcal{P}}$ (resp. $\mathcal{O}^{\mathcal{P}}$) the input (resp. output) meta-models of $\mathcal{P}$, by enc($\mathcal{R}^{\mathcal{P}}$) the encoding of the mapping rules of $\mathcal{P}$, by $enc(\mathcal{M})$ the representation of the elements of $\mathcal{M}$ by Prolog facts, and by $\mathcal{H}^{\mathcal{P}}$ the rules for helpers. Now, a model $\mathcal{M}$ *satisfies* $\mathcal{H}^{\mathcal{P}}$ if the interpretation $\mathcal{H}^{\mathcal{M}}$ coincides with the logic consequences of $\mathcal{H}^{\mathcal{P}} \cup enc(\mathcal{M})$.

**Theorem 1.** *Given a PTL program $\mathcal{P}$, a model $\mathcal{M}$ that satisfies $\mathcal{H}^{\mathcal{P}}$, and $\mathcal{M}' = \cup_{1 \leq i \leq n} [\![r_i]\!]^{\mathcal{M}}$, where $r_1, \ldots, r_n \equiv \mathcal{R}^{\mathcal{P}}$ then $o \in \mathcal{O}^{\mathcal{M}'}$ iff $o$ is a logic consequence of the program $enc(\mathcal{R}^{\mathcal{P}}) \cup \mathcal{H}^{\mathcal{P}} \cup enc(\mathcal{M})$.*
*Sketch of Proof:*
*The idea is to prove that (a) $rn(\overline{v})$, (b) $object(mm, c_j, o, \overline{v}, [att_1(va_1), \ldots, att_{l_j}(va_{l_j})], q)$ and (c) $associationObjects(mm, r, o, o')$ are logic consequences of the program $enc(\mathcal{R}^{\mathcal{P}}) \cup enc(\mathcal{M}) \cup \mathcal{H}^{\mathcal{P}}$ iff (a) $\overline{v} \in \overline{C}, \overline{(p, C)}$ $\in << ps >>^{\mathcal{M}}$ and $[\![bc]\!]^{\mathcal{M}}_{(p,v)}$ is true; (b) $o \in \mathcal{O}^{\mathcal{M}'}$, $o \in c_j^{\mathcal{M}'}$, $att_i^{<\mathcal{M}',j>}(o) = va_i$, $1 \leq i \leq l_j$, $\overline{v} \rightarrow_q^{\mathcal{P}} o$; and (c) $o' \in r^{\mathcal{M}'}(o)$, respectively, where (rule rn from ps where bc to obs) $\in \mathcal{P}$.*

The conformance of $\mathcal{M}$ (resp. $\mathcal{M}'$) to $\mathcal{I}^{\mathcal{P}}$ (resp. $\mathcal{O}^{\mathcal{P}}$) is not required for this result. In particular, in order to ensure the conformance of $\mathcal{M}'$ we have to require semantic conditions over the rules of $\mathcal{R}^{\mathcal{P}}$ and $\mathcal{H}^{\mathcal{P}}$, and the conformance of $\mathcal{M}$. This result is out of the scope of this paper.

## 4 PTL Interpreter, Debugger and Tracer

Now, we would like to give some details about the PTL interpreter, debugger and tracer (more details can be found in `http://indalog.ual.es/mdd`). Firstly, we would like to show how a PTL program is executed from Prolog. A predicate `ptl` is called with the file name in which the PTL code is included, that is: `?- ptl('er2rl.ptl')`. The `ptl` predicate automatically generates the Prolog library of the meta-models defined in the PTL program, it encodes PTL rules with Prolog rules, and it generates the target models from the source models. In order to execute a particular transformation, we have to specify how source and target meta-models are defined in terms of the MOF meta-meta-model. Hence a transformation is executed as follows: `?- transform(['mm2er.ptl','er2rl.ptl','rl2mm.ptl']).` The full code of the transformations `mm2er`, `er2rl` and `rl2mm` can be downloaded from `http://indalog.ual.es/mdd`.

Debugging is able to find rules that cannot be applied, and provides the location in which the error is found. PTL mapping rules are not applied due to Boolean conditions that are not satisfied and objects that cannot be created. A Boolean condition is not satisfied whenever a certain equality or inequality is false.

Let us suppose that the PTL rule *table2_ er2rl* of Figure 7 includes *p@name==\"*\"* instead of *p@max==\"*\"*. This is a typical programming error and it cannot be detected by the compiler (i.e, the PTL program is well-typed and syntactically correct). Now, we find that the target model is wrong. In such a case we can query the debugger, obtaining:

```
?- debugging(['mm2er.ptl','er2rl.ptl','rl2mm.ptl']).
Debugger: Rule Condition of: table2_er2rl cannot be satisfied.
Found error in: role_name
```

The debugger shows the name of the PTL rule (i.e. *table2_ er2rl*) that cannot be applied and the error found (i.e. *role_ name*). But the debugger can also detect that target objects cannot be created, for instance, let us suppose that we write *p@navigable==false*, instead of *p@navigable==true* then the debugger answers:

```
?- debugging(['mm2er.ptl','er2rl.ptl','rl2mm.ptl']).
Debugger: Objects of: table2_er2rl cannot be created.
Found error in: resolveTemp
```

In this case the objects of the rule *table2_ er2rl* cannot be created, and the programming error comes from *resolveTemp*, because the call does not succeed.

However, we can find a programming error due to the opposite case: a certain target model element is created but it is wrong. In such a case, we can trace from the wrong target element to find the reason (i.e., applied rules and source model elements) of the creation of such element. A target element can be created from a missing Boolean condition.

Let us suppose that in rule *foreign2_ er2rl* of Figure 7 we omit the Boolean condition *q@navigable==false*. We review the target model and find a wrong element: *is_ registeredStudentid_ student*. Now, we can trace the execution from the XMI identifier of the wrong element, obtaining the applied rules are the XMI ids of the source model elements as follows:

```
?- tracing(['mm2er.ptl','er2rl.ptl','rl2mm.ptl'],'275284307q275325284r2f2c').
Tracing the element: 275284307q275325284r2f2c

Rule: foreign1_rl2mm
Element: foreign
Metamodel: rl
....
Element: property
Metamodel: mm
xmi:id is CrZGO_iGCkzsbhYY
```

## 5 Related Work

Logic programming based languages have already been explored in the context of model engineering in some works. The *Tefkat* language [9,8] is a declarative language whose syntax resembles a logic language with some differences (for instance, it incorporates a *forall* construct for traversing models). *VIATRA2* [1] is a well-known language which has some features which make the proposal close to our approach. Although the specification of model transformation is supported by graph transformations, Prolog is adopted as a transformation engine. Thus XMI models and rules are translated into a Prolog graph notation. Prolog has been also used in the *Model Manipulation Tool (MoMaT)* [11] for representing

and verifying models. In [5], they present a declarative approach for modeling requirements (designs and patterns) which are encoded as Prolog predicates. A search routine based on Prolog returns program fragments of the model implementation. Traceability and code generation are based on logic programming. They use *JTransformer*, which is a logic-based query and transformation engine for Java code, based on the Eclipse IDE. Logic programming based model querying is studied in [4], in which logic-based facts represent meta-models. In [10] they study a transformation mechanism for the EMF Ecore platform using Prolog as rule-based mechanism. Prolog terms are used and predicates are used for deconstructing and reconstructing a term of a model. In [2] the authors have compared OCL and Prolog for querying UML models. They have found that Prolog is faster when the execution time of queries is linear. *Abductive logic programming* is used in [6] for reversible model transformations, in which changes of the source model are computed from a given change of the target model.

Most of the quoted works make use of Prolog for representing meta-models and model elements. The representation varies from one to another, but in essence Prolog facts are used for representing model instances while rules are used for representing transformations and constraints on models. In our case Prolog facts are also used for representing model instances; however they are not directly handled by the programmer given that they are automatically generated from XMI files. Prolog rules are used as helpers in PTL but the main component of PTL are mapping rules which are rules inspired by ATL.

## 6 Conclusions and Future Work

In this paper we have presented a model transformation language based on logic programming. As future work, we could also extend debugging and tracing capabilities. For instance, we have considered tracing from the target model to the source model; however, tracing from source model to target model would also be possible and interesting. We would like also to improve the performance of our system. The execution times we have obtained for small examples are satisfactory. However, we would like to optimize Prolog code and Prolog representation, for instance, storing Prolog facts in secondary memory for large models.

## References

1. Balogh, A., Varró, D.: The Model Transformation Language of the VIATRA2 Framework. Science of Programming 68(3), 187–207 (2007)
2. Chimia-Opoka, J., Felderer, M., Lenz, C., Lange, C.: Querying UML models using OCL and Prolog: A performance study. In: IEEE International Conference on Software Testing Verification and Validation Workshop, ICSTW 2008, pp. 81–88 (2008)
3. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal 45(3), 621–645 (2006)

4. Dohrmann, P., Herold, S.: Designing and Applying a Framework for Logic-Based Model Querying. In: 2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 164–171 (2010)
5. Goldberg, M., Wiener, G.: A Declarative Approach for Software Modeling. In: Russo, C., Zhou, N.-F. (eds.) PADL 2012. LNCS, vol. 7149, pp. 18–32. Springer, Heidelberg (2012)
6. Hettel, T., Lawley, M., Raymond, K.: Towards Model Round-Trip Engineering: An Abductive Approach. In: Paige, R.F. (ed.) ICMT 2009. LNCS, vol. 5563, pp. 100–115. Springer, Heidelberg (2009)
7. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: A model transformation tool. Science of Computer Programming 72(1-2), 31–39 (2008)
8. Lawley, M., Steel, J.: Practical Declarative Model Transformation with Tefkat. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 139–150. Springer, Heidelberg (2006)
9. Lawley, M., Raymond, K.: Implementing a practical declarative logic-based model transformation engine. In: Proceedings of the 2007 ACM Symposium on Applied Computing, SAC 2007, pp. 971–977. ACM, New York (2007)
10. Schätz, B.: Formalization and Rule-Based Transformation of EMF Ecore-Based Models. In: Gašević, D., Lämmel, R., Van Wyk, E. (eds.) SLE 2008. LNCS, vol. 5452, pp. 227–244. Springer, Heidelberg (2009)
11. Storrle, H.: A Prolog-based Approach to Representing and Querying UML Models. In: Intl. Ws. Visual Languages and Logic (VLL 2007), vol. 274, pp. 71–84 (2007), http://ceur-ws.org/Vol-274/

# Hypermodelling Reporting: Towards Cockpits for Code Structure

Tim Frey and Matthias Gräf

Otto-von-Guericke University, Magdeburg, Germany
tim.frey@tim-frey.com, matthias.graef@gmail.com

**Abstract.** Do we have a vendor lock in? How many classes of a framework do we extend in our code? These questions may be asked by software development managers. In order to reveal such facts, a lot of effort is needed. We present the Hypermodelling approach to build software cockpits and use it to reduce the effort. We show a cockpit for the variance of software that is reflecting facts about the inheritance of types. We reveal a schematic cockpit view and evaluate the effort to implement it. Project managers can now use a cockpit to investigate software variances more easily. This also enables easy investigations of dependencies on frameworks. Important indicators about variance can now be investigated at a central spot. This avoids costly, time-consuming and deep investigations in the first place. Further research can reveal additional cockpits for other roles to cover the whole development cycle. Furthermore, the reasonable effort to create such cockpits enables the possibility to create different kinds of cockpits and evaluate or compare the usage of those.

**Keywords:** hypermodelling, software dependency, project controlling.

## 1 Introduction

The complexity of software development is undoubted. Project managers face the problem to control the development process. In order to overcome this complexity, project teams use issue tracking systems. Issue tracking systems enable the specification of desired functionality or needed bug fixes [1]. Researchers come up with solutions, leveraging Data Warehouse (DW) [2] technology to control the data of issue tracking systems [3]. This enables managers to gain an overview by dashboards. This project control dashboards, also called cockpits, contain key performance indicators and charts about the project process.

Besides the issue tracking systems and their data, many challenges are faced within the development of a system. Often parts of systems or whole systems themselves are created by offshore development teams without having senior developers checking the code quality. In case, the foreign code base is merged into the own repository, managers want to know which frameworks the foreign code is depending on. Currently, programs need to be parsed to extract information about framework usage. That consumes time and effort and makes information mining inflexible and static. Furthermore, if you want to see the dependencies in a clarified way (e.g. dependency counters, statistics), this is hardly possible. Therefore, the code is currently often merged into a company's repository without the needed investigations because the effort is too high. If the code needs

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 395–407, 2013.

to be altered at a later time, the box of Pandora is opened: Developers do not have the necessary knowledge about the used frameworks. That missing knowledge is the main problem to react accordingly. If a manager knows about such issues, he can take actions to be ready for it. For instance, such actions can be to validate/check the contracted features of the desired system for training purposes of the developers for the used frameworks. Therefore, an easy way to control a minimum standard for taking over code developed by an outsourcing company is needed. Currently, the software cockpit cannot address this issue since it is just based on issue data that is not available in this case. Furthermore, cockpits are not targeting at source code structure. Hence, software engineering faces the challenge to provide efficient overviews of code structure.

In prior work, we presented Hypermodelling that utilizes DW technology for source code inspections and shows that queries on code are is possible [4–6]. Our current contribution is is: We evaluate the feasibility if Hypermodelling can be used to create a cockpit as replacement for a code study that was done before manually and revealed several indicators about inheritance [6]. We also meter the effort to build such a cockpit and show how we derived this cockpit systematically. Additionally, we present the cockpit that we built in this paper online in a video.[1] With our cockpit it is possible to investigate the dependency on frameworks on various abstraction levels just with a few clicks. This helps project managers to investigate foreign and own code more easily. Furthermore, out of the whole effort to built this specific cockpit we can conclude that Hypermodelling allows to build cockpits for code structure in a reasonable timeframe.

First, we describe the needed background about the Hypermodelling approach and how we use it as basis to create a cockpit. Succeeding, we discuss a concrete cockpit and evaluate its realization. We compare our work to related research, draw conclusions and give an outlook to future work.

## 2 Background

Hypermodelling uses DW-technology for program analysis. A more detailed description is available in [4–6]. DW systems are an integrative component in business computing [2]. They are used to assemble data of different sources together. The integrated data is arranged into multi-dimensional data structures, i.e. so called cubes, which serve as base for queries [2]. These queries allow aggregating different relations and hierarchies that occur in the data. For instance, revenues for a specific salesman in a specific area can be computed for a given time period. Thereby, this query aggregates the region, the salesman and the time in relation to revenue indicators. Likewise, hierarchies can be abstracted. For instance, the region of the salesman can be split into continents, countries and counties and the aggregates are associated with the distinct revenues for those. Likewise, this can be done for other hierarchies, like customer group, year, or department. Generally, the idea is that different aggregations enable detailed investigations. With Hypermodelling we introduce the idea that programming elements, like annotations or classes, are similar to the data that is used within a DW. For instance, classes are defined within a package hierarchy. Annotations are associated with classes

[1] Http://hypermodelling.comVideo:http://www.youtube.com/watch?v=6SSqpYRz_0w

and their members. They are also defined in their own package. This is like the association of a salesman to a region, time period, and revenues. The hierarchies in code are similar to hierarchies of region or time. All together, we load source code into a DW and realize the internal semantic associations as a DW cube. This enables us to built queries on top of this cube and it creates the possibility to compute different aggregations for code. For this paper, we combine cubes that are presented in our former work [4–6] and built queries on it. The results of queries on this cube are the reports that we present in this paper. Figure 1 gives an overview over the general structure. We extract program structure and issues on code style into a relational database. From this the cubes are filled. Next, queries on these cubes can be executed. Such queries are used to fill the contents of the code cockpit that we present in this paper.

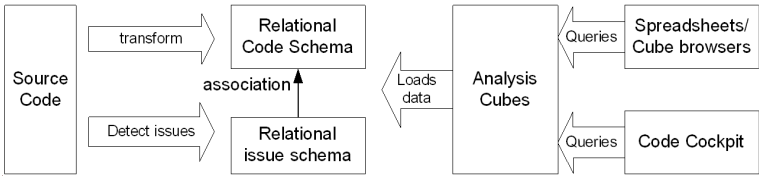



**Fig. 1.** From code to the cockpit

## 3 Software Variance Cockpit

First, we explain the variance of software with an UML class diagram. Then we go into detail and discuss main indicators to investigate dependencies based on variance indicators. Succeeding, we present a schematic view how a cockpit may look like.

### 3.1 Software Variance

Different aspects of software variance and viewpoints about the dependency are discussed in former work [6]. Therein, a detailed study about indicators that we present and use in the following is done. Here, we briefly present relevant facts that reflect properties of a software system and which we use in the cockpit. We describe the variance and indicators that are raised later, again. In order to reveal the diverse variance viewpoints, we present demo called interfaces and methods to explain software variance in the context of inheritance.

The class diagram in Figure 2 shows two types (*NamedEntity* class and the *Clinic* interface) that get inherited by classes and are defined in the same package. We call the types that are implemented or extended just supertypes and their children inheritors. The inheritors occur in different packages for the interface and in the same package for the class. Methods get overridden and additional methods are contributed in their child types. Through overriding and addition, the defined standard by the supertypes

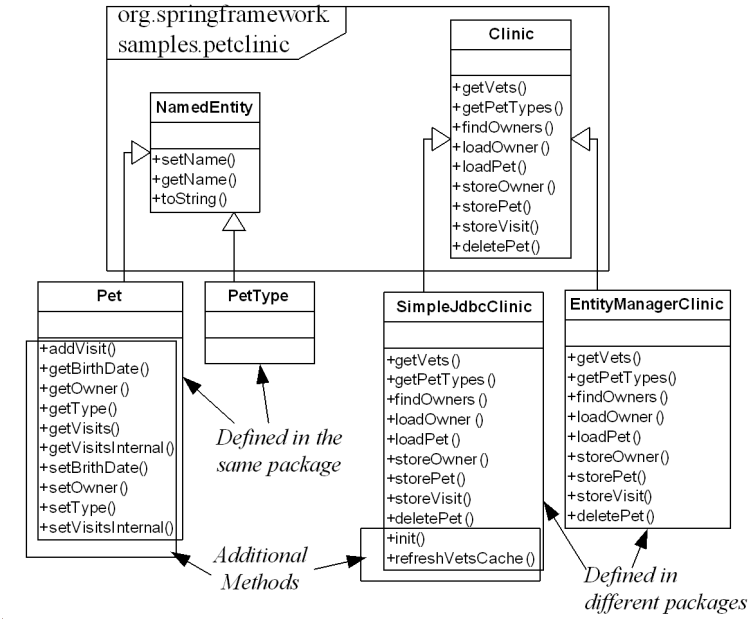



**Fig. 2.** Variance explained with an UML class diagram

get varied in the children by different means. In the following, we describe diverse indicators that can be computed out of this diagram.

### Total Inheritors

Investigations have shown that the dependency on parent classes is problematic in cases of software updates [6]. Therefore, one important indicator of software variance and the dependency on libraries is the total amount of inherited types. The more types are inherited the more is the software coupled to the super classes or interfaces. Typically, the types of libraries that are inherited are structured in a package hierarchy. This makes it possible to investigate the dependency on a library just by investigating the dependency on a package hierarchy. Therefore, the total amount of inherited types of a package depicts one degree of dependency on the package. If this package belongs to a library the package dependency is a library dependency. When more types get inherited of a package the dependency is higher. An example that can be depicted from Figure 2: The total amount of inheritors of the *NamedEntity* class is two; and the same goes for the clinic class. In case we refer to the total variance of the *org.springframework.samples.petclinic* package, the total amount is four. If we use the child package as discriminator, we can compute the amount for that perspective, too. For instance, *Pet* and *PetType* is defined in the same package and the total amount of inheritors that form the *org.springframework.samples.petclinic* package is two. Due to the fact that both other types (*SimpleJdbcClinic* and *EntityManagerClinic*) are defined in different packages the amount would

be one for each package. This shows us that the amount of total inheritors and the variance differs from the viewpoint of the package wherein the children are defined. This is important, because the packages that are depending on types of another package can be depicted by numbers, reflecting the viewpoint. Hence, project managers can investigate the amount/degree of dependency from a package to the supertypes of another package.

**Distinct Inheritors**

Similarly to measuring total inheritance, but from a different perspective, is the amount of distinct inherited types. On the contrary to the total inherited types this number reflects the diversity of the inherited types. Thus, it answers the question: How many different types are inherited. If plenty of different types serve as parents the number grows. So, taken Figure 2 as example, the distinct inheritance of the *org.springframework.samples.petclinic* package is two, since only two different types are inherited. For the *Pet* and *PetType* package it is one what is likewise the same for the two other types in their own package. There, one type gets varied for each package.

**Implementation Ratio**

However, when we want to have an indicator that reflects the "intensity" of used types we need another perspective. Thus, we introduce the indicator implementation ratio. With it, we enable a more concrete comparison of distinct supertypes to the amount of subclasses: The indicator enables to compare the amount of distinct types with the total inherited types in one number. Since classes or interfaces that implement or extend other types add commonly new functionality to the existing ones, also the original types get varied. Therefore, we see figures of implementation ratio as one indicator to measure the variance and dependency of supertypes. For all the shown types in Figure 2 the implementation indicator of the *org. springframework.samples.petclinic* package for all packages is computed as follows: The amount of inheritors of the package is four. The distinct implemented supertypes are two. So we can compute 4/2=2 and see that the indicator is two. Clearly, this can be, like all the other indicators before, computed for the diverse packages.

The measure of the implementation ratio is useful to depict a standard variation. A high or low value is a first indicator how intense supertypes are varied. When the indicator has at a high value, we can conclude that the defined standard of the supertype gets aligned a lot. This means that the same types are often implemented or extended. Every time a type is used, it gets adapted to the specific application needs. This way, developers have a starting point for further investigations to determine the types, responsible for high variance. This is especially useful, if varied supertypes are updated. If supertypes with a high ratio get updated, many children are dependent on them. Thus, it is recommendable to investigate how the children adjust the supertype to keep its future version compatible to the inheritors. Furthermore, framework manufacturers can use the ratio indicator to determine which types are mostly adapted. With that information, they can investigate how developers vary the types. It is possible to depict if there is a common use or functionality in the extending types. If so, this functionality can be encapsulated into a new version of the supertype.

**Drill Down to Top Inherited Types**

When a project is investigated the desire to drill down to concrete types raises regularly. Investigators ask which specific types are inherited and how often. Therefore, this kind of drilldown is interesting for investigations. First an investigator focuses an abstract level of a package and then investigates the concrete inherited types. Therefore, a drill down of inheritance variance is to look into the specific types that get inherited. For example a parent package, like the *org.springframework. samples.petclinic* is correlated with an implementing package, like the one containing the *Pet* and *PetType,* we can compute which types get most inherited. Thus, it is just the total amount of inheritors (Section 2) for a specific package. The most inherited types are then the top inherited types in that package.

**Drill Down to Top Overwritten Methods**

It is often interesting to determine the most common method names in inheritors. This knowledge is important to see first, if there are similarities and second to have the possibility to compare it with the original methods in the supertype. Through the inspection of the amount of used method names in children, conclusions can be drawn how often a method gets overridden. This way, estimations can be done about how much changes in the supertype method may affect the children.

The more types inherit a type the more it is important in the implementation. For instance, we can see that, logically, all methods defined in the *Clinic* superinterface get implemented in clients. For the *Pet* supertype we can see that the original defined methods do not get overridden in children and thus do not vary. However, we can depict which other methods are added in children. This gets quite handy, when looking at a larger number of children, to answer the question: Do many children override the same methods or add methods with the same name?

### 3.2 A Schematic Cockpit Proposal

Out of the former described indicators, we formulate the arrangement of different elements for analysts. All the different variance stats are presented in Figure 3. The views with the grey diamonds are recommended to be graphical to provide an easier overview that is more eyes catching than pure tables. In general, we use principles of information dashboard design [9] as far as they can be applied for our special case.

**The Overview Part**

The overview part shows an abstract overview and graphical visualization of the currently investigated package or project. The following numbers refer to the circled numbers in Figure 3 and Figure 4 that is following in Section 4. We refer here already to Figure 4, because Figure 4 shows an actual implementation of Figure 3. This way, the number's description corresponds also to the numbers in Figure 4.

1. The package selection enables the user to select packages that he wants to explore. He can select packages from the hierarchy to have a known navigation.

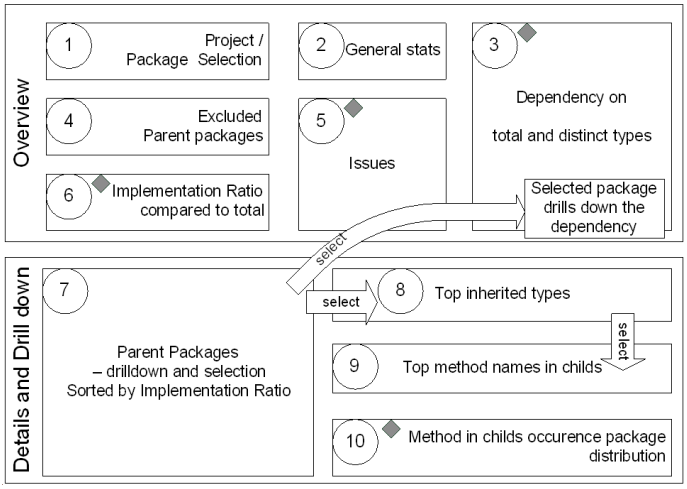



**Fig. 3.** A Schematic Variance Exploration Cockpit

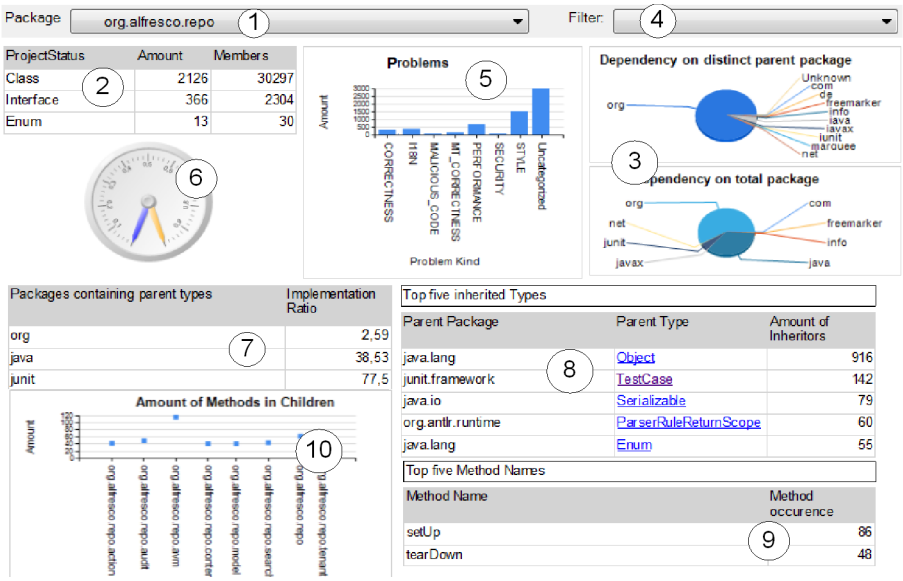



**Fig. 4.** The cockpit in action

This selection serves the purpose to define the base that is investigated for variance. All other views show facts about the package that is selected here. This selection is the start of the exploration of a projects package hierarchy that slices/discriminates all the other elements of the report.

2. The general stats show statistic data about the current selected package. How many classes are in there, how many interfaces and how many method calls are occurring in a package. The main reason to show these stats is to give the investigator an easy overview to estimate the ”size” of the investigated package.
3. Here we see a graphical representation about the dependency on different and total types of the current investigated package. There, the abstract level of packages is shown and how many types of a package are inherited. We propose to use a pie or spider-net chart to visualize the total and distinct dependency graphically.
4. Here the user can specify excluded packages. The reason that we see a necessity to offer this view lies in our previous investigations [6]. There, we recognized that standard packages are used often and the use density of one package covers the usage of others. Therefore, users can just exclude dominant packages that contain parents to gain a clearer view. All the different elements in the cockpit are filtered to exclude those packages.
5. In order to show also other indicators, we recommend showing them additionally to the variance issues that are related to the currently selected package. Such issues can be successful or unsuccessful unit tests of the classes in the package or indicators like code smells. Through the showing of the issues at the same time with the dependency on parent packages, managers can see correlations between issues and used parent packages at the same time. For example, when a certain package is used a certain style issue seems to be occurring with a high amount. When this thesis is formulated it can be the beginning of an in-detail analysis.
6. The implementation ratio of the current selected package (view 1) is shown with a graphical visualization. Through this, a direct excelling rate can be determined easily. So when the rate is low or high, is can be instantly perceived to decide if another package should be investigated.

**The Drill Down Part**

This part shows the details about the inheritance and presents specific types, packages and methods to the user. It is the part where the user can get into details.

1. We show the packages from where the supertypes origin. The user can drill down into the package hierarchy of the supertypes. When this drill down is done the depending shown measures, are updated. For instance the dependency on the distinct types (view 3) is now computed for the selected packages.
2. When a package is clicked (in view 7), also view 8 is adjusted. In this view we show the top parent types that are inherited from this package. This way, users can select a project and then explore which packages are the most original and then drill down into the parent packages hierarchy and into the concrete parent types.
3. Like in the case of packages and the drill down to types, this is also a drill down. In view 8 a type can be clicked and then the top method names in children get listed in this view. Thereby, the total occurrence of this method name is shown. This way, it is easily to see which method names are very common in the selected package (1).

4. However, in view 9 users have the possibility to see the method occurrence in the selected package. The visualization in 10 gives the users the opportunity to see the amount of methods of the currently selected supertype (selected in 8) in other packages. Users can easily see which other packages could be interesting for investigations about the same type. Most likely if other packages have a high amount of methods for the children of that type, the other packages share a huge variance, too.

## 4 Application Evaluation

In order to evaluate our approach to provide a dashboard for the inspection of software variance we evaluate the feasibility through a demo application. We answer the question how much effort is needed to realize a cockpit based on the Hypermodelling approach. Furthermore, we overcome the necessity to create queries and provide a flexible real world solution and not to stick to a schematic ivory tower. Our cockpit can serve additionally for further investigations of software variance. In the following, we first line out the necessary effort to build the cockpit based on the Hypermodelling approach. Then, we give a brief overview about the implementation that reflects the schematic overview in Figure 3.

### 4.1 Realization Evaluation

For the realization, we used the following approach: First, a description, similar to the one in section 3 was made. We used a schematic picture similar to Figure 3 and a detailed explanation what different elements should represent to provide a holistic goal. Then, we gave this description, plus a DW that was already filled with source code and the associated queries for the diverse elements to a reporting specialist with the task to create the cockpit. The specialist was experienced with the used reporting technology[2] and worked in that field for five years. Within 10 hours the specialist realized a first version of the cockpit that contained fixed queries. Fixed means that the queries could not be altered and the view was not dynamic and showed a report without selectable elements. Out of this first version, we adjusted and specified in detail requirements. Those requirements how the cockpit should look and behave were precise and well formulated and given to the specialist. This iteration was done 4 times and every time new requirements about behavior were added. For instance, first we defined which elements should be parameterizable and change dynamically. Then, we defined different selectable elements and so on. In the last iteration, the loading time of the cockpit came in focus. It took five minutes with automatically generated queries on an Windows 7 pc with an Intel code 2 duo with 4 GB RAM. In order to fix this performance issue the expert rewrote two queries of the cockpit manually, what reduced the loading of the cockpit down to acceptable 11 seconds. In order to achieve this performance increase only a small optimization to exclude empty values was applied. Therefore, even further tuning is possible. This way, we recognized that performance tunings are no real issues with a query based approach.

[2] We used Microsoft Analysis Services and the SQL Server 2008 R2.

All together, including the first fixed version the specialist needed about 80 hours to create the demo cockpit. All over, it needs to be mentioned that the specialist is normally focused on more simple and business related reports and needed to look up a lot of information for the complexity of the cockpit and its possibilities within that time. Additionally, the cockpit was built after his normal working hours. Thus, we can estimate that with a main working task and the complete knowledge about the used reporting technology the cockpit would have been built much faster. Therefore, we see indications that cockpits for the internal code structure can be created within reasonable time and effort.

### 4.2 Cockpit Usage

We show the cockpit in Figure 4. The numbering of the various elements corresponds to the schematic view in Figure 2. All the various parameters that are defined in the report can also be selected through URL parameters, when accessing the report. This has the advantage that when something of interest is discovered, the current looking report can be just linked and send to another person. Then, the other person has completely the same view and can go into further investigations. Also when an interesting issue is revealed various export formats give the possibility to use the export for presentations or archival reasons.

We see that the user can select a package for which the various stats are displayed (1). This is done via a parameter that can, as described before, also be specified via the URL. When the parameter is altered the whole report changes and shows the new specified package. The project stats (2) refer to the current package. Currently, it shows the amount of types (classes and interfaces) and their members (methods and fields). The different dependency types (total and distinct) are visualized via pie charts to give a quick graphical insight (3). In order to support estimations if there may be correlations of style issues and the dependency of packages, the problem bars (5) show the amount of detected issues. The speed indicator (6) allows gaining a quick overview of the variance of the selected package to the whole investigated project. All together, the whole charts can be sliced by excluding packages that contain parent types (4) to avoid a dominance of certain packages.

In (7) we offer a drill down into the parent packages. Depending on this selection the top parent types get shown in 8. Also, the dependency within 3 is adjusted. So, if the user starts to explore a specific package contain parent types he can explore this kind of hierarchy.

When a specific top inherited type is selected the most common method names are presented in 9. Additionally, the distribution chart in 10 shows the occurrence of methods in children in all the packages. Through this chart the user can quickly gain an insight if the current package that is investigated alters the selected type more than others.

## 5 Related Work

Soft-Pits focus on software cockpits for issue tracking systems [3]. More detailed industrial applications and empirical evaluations indicate further benefits of a software

cockpit [7]. Our cockpit approach differs since it is not only based on data out of issue tracking systems. Furthermore, our cockpit enables a direct viewpoint on code structure. Through the empirical studies we are certain that our cockpit would also be considered useful. We see our cockpit as an implementation for a specific role: the role of a code analyst. Thus, generally, we do not see other research as competitive, rather we see it as supplemental and propose to do further investigations about synergies between the different approaches and results.

Storing source code in databases [8] describes code queries with a logic programming language. The source code is stored in a relational database and queries in the logic language are translated into SQL queries. This can be seen as related because relational databases can be queried, too. Hypermodelling uses DW technology to do the query and the relational model is just used to be the source for the cubes. Therefore, it was very efficiently possible to create a report for the software dependency, because a whole DW toolbox could be used. With normal databases the report would have had to be programmed by hand.

Semmle code[3] is a source analysis and investigation tool. It offers a query interface to investigate source code and its associated elements within the development environment. Thereby result presentations like heat maps as addition to traditional pie charts are possible. Our cockpit differs, because it is based on the DW technology and focuses on the efficient investigation of certain use cases. But generally, the visualizations offered by Semmle should be evaluated if they can be a useful contribution for a software cockpit.

The IBM Rational Team Concert (RTC) tools[4] provide dashboards for projects with a set of predefined widgets, allowing monitoring of the 'health' of a project. The main difference to our proposed cockpits is that we focus on queries for the source code structure and RTC cockpits show mostly project statistics out of issue tracking. Therefore, we see possible synergies for further research in extending the RTC dashboards with Hypermodelling based ones.

We investigate different factors about inheritance already in prior work [6]. There, we inspect inheritance with manual queries and derive the indicators that we use within the cockpit of the current paper. Therefore, this paper advances prior research by creating a cockpit on top or the prior derived queries and indicators.

Finally, our presented cockpit origins "traditional" DW applications. In the area of DW cockpits are used to give a management overview about business indicators and offer analysts and managers easy possibilities to gain quick insights about the company performance [2]. Therefore, deeper comparisons about best practices in this area can reveal how to create better cockpits for software.

## 6 Conclusions and Further Work

We described the business case about the difficulties to investigate source code for dependencies on frameworks. A brief description of the Hypermodelling approach was

[3] http://semmle.com

[4] https://jazz.net/products/rational-team-concert/

given. The diverse indicators of software variance that express inheritance dependencies were described. We presented a schematic overview integrating these indicators and their relation together. We evaluated the feasibility through an implementation. Thereby, we figured out that this kind of cockpits can be built with reasonable effort and within a reasonable time range. Related research showed indications that code cockpits are valuable in real world projects and supported our idea. Now, audits about the dependency and standard variation of software can be done easily. Our contribution enables further research about different cockpits for various roles about the internal structure of code.

However, in this paper we presented a cockpit that describes the dependency mainly on inheritance mechanisms of object oriented programming. Also other mechanisms are available that are used to couple artifacts in a program together Therefore, we see the necessity to investigate which kind of other dependency mechanisms exist and integrate them in a future version of the cockpit.

Furthermore, our cockpit shows that it is generally possible to create a cockpit, visualizing relations in the code structure through ”traditional” management charts. Further research should focus on studies about the application of cockpits in real world scenarios. First, we see the necessity to investigate how and by whom exactly the cockpit is used. This can lead to further application business cases for our cockpit. Additionally, we are working towards real world evaluations within development projects. The goal is hereby to come up with case studies and adjust the cockpit to specific developer needs.

Another trail of research is to create further cockpits about the code structure to enable advanced project management support. This way, we see further research to identify further cockpits within diverse areas of development. For instance, developer or management centric dashboards can also be developed. As a first step, we plan to work together with enterprises to reveal the most desired cockpits and the information that should be shown in them. This way, we are certain that code cockpits can enable advanced software project management support in the future.

**Acknowledgement.** Thanks go to Gunter Saake, Veit Köppen, Manuela Fath and the anonymous reviewers for their comments and help on earlier drafts of this paper.

## References

1. Larndorfer, S., Ramler, R., Buchwiser, C.: Experiences and Results from Establishing a Software Cockpit at BMD Systemhaus. In: 35th Euromicro Conference on Software Engineering and Advanced Applications, pp. 188–194 (2009)
2. Inmon, W.H.: Building the Data Warehouse, 4th edn. J. Wiley & Sons, New York (2005)
3. Münch, J., Heidrich, J.: Software project control centers: concepts and approaches. Journal of Systems and Software 70, 3–19 (2004)
4. Frey, T., Köppen, V., Saake, G.: Hypermodelling - Introducing Multi-dimensional Concern Reverse Engineering. In: 2nd International ACM/GI Workshop on Digital Engineering (IWDE), Germany (2011)
5. Frey, T.: Hypermodelling for Drag and Drop Concern Queries. In: Proceedings of Software Engineering 2010, Gesellschaft für Informatik (GI). Springer, Germany (2012)

6. Frey, T., Köppen, V.: Exploring Software Variance with Hypermodelling - An exemplary approach. In: 5. Arbeitstagung Programmiersprachen im Rahmen der Software Engineering 2012, Gesellschaft für Informatik. Springer, Germany (2012)
7. Ciolkowski, M., Heidrich, J., Simon, F., Radicke, M.: Empirical results from using custom-made software project control centers in industrial environments. In: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 243–252. ACM (2008)
8. Hajiyev, E., Verbaere, M., de Moor, O.: *codeQuest:* Scalable Source Code Queries with Datalog. In: Thomas, D. (ed.) ECOOP 2006. LNCS, vol. 4067, pp. 2–27. Springer, Heidelberg (2006)
9. Few, S.: Information Dashboard Design: The Effective Visual communication of Data. O'Reilly Media (2006)

# Search in Source Code Based on Identifying Popular Fragments

Eduard Kuric and Mária Bieliková

Institute of Informatics and Software Engineering,
Faculty of Informatics and Information Technologies,
Slovak University of Technology in Bratislava
Ilkovičova 3, 842 16 Bratislava, Slovakia
{kuric,bielik}@fiit.stuba.sk

**Abstract.** When programmers write new code, they are often interested in finding definitions of functions, existing, working fragments with the same or similar functionality, and reusing as much of that code as possible. Short fragments that are often returned by search engines as results to user queries do not give enough information to help programmers determine how to reuse them. Understanding code and determining how to use it, is a manual and time-consuming process. In general, programmers want to find initial points such as relevant functions. They want to easily understand how the functions are used and see the sequence of function invocations in order to understand how concepts are implemented. Our main goal is to enable programmers to find relevant functions to query terms and their usages. In our approach, identifying popular fragments is inspired by PageRank algorithm, where the "popularity" of a function is determined by how many functions call it. We designed a model based on the vector space model by which we are able to establish relevance among facts which content contains terms that match programmer's queries. The result is an ordered list of relevant functions that reflects the associations between concepts in the functions and a programmer's query.

**Keywords:** search, source code, reuse, pagerank, ranking, functional dependencies.

## 1 Introduction

Software development is one of the most creative things a human can do. Every day, a programmer needs to answer several questions for the purpose of finding solutions and making decisions. It requires the integration of different kinds of project (software system) information, as well as, it depends on the programmer's knowledge, experience, skills and inference. The information retrieval is a key area to obtain success on reuse initiatives. "*To reuse a software component, you first have to find it*" [14]. There were several studies conducted to find out how programmers comprehend software systems and what information they need to know about source code [11]. The studies deal with identifying and analysing motivations, strategies, and goals, which developers have when they search in source code. In [12] authors summarize a list of motivations (search

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 408–419, 2013.

forms) for code search. Even though several search forms are used in code search engines, there is still room for innovations.

The structure of code is not conducive to being read in a sequential style. In general, programmers read code selectively. They identify parts of the source relevant to the target task. For example, programmers are looking for the usage of an existing piece of code, the implementation of some functionality or code with some properties (patterns). Empirical studies indicate that 40 % to 60 % of source code is reusable within one application and only 15 % is unique to a specific application [10]. However in software engineering there is necessity to adopt systematic reuse code. It requires instruments which facilitate reuse such as source code search tools. The good source code search tool encourages access to the existing code instead of creating new one. The developer's motivation and goals for searches indicate certain functionality, which is required in such search tools.

In general, the process of identifying the parts of the source code that correspond to a specific functionality is called *concept/feature location*, i.e., the aim of the process of *concept location* is to find the source code that implements these concepts. The input to the concept location process is a description of a problem (a change task) expressed in natural language and the output is a set of software components (elements) that implement or address the concept. However, there is a difficulty that the input and the output are in different levels of abstraction, i.e., the input is formulated in nutural language (domain level) and the output is the source code (implementation level). Therefore, considerable knowledge is required to translate from one level to another.

The complexity and significance of the concept location process increases with the size of the software system. The aim of the methods for concept location is to reduce the search space which the programmer needs to investigate (explore). One common way of various approaches is a decomposition of the source code into units, such as classes and functions (methods) which are enriched with additional information (e.g. relationships between elements of the source code).

Understanding code and determining how to use it is a manual and time-consuming process. In general, programmers try to find initial points such as relevant functions (methods in object oriented parlance). They want to easily understand how the functions are used and see the sequence of function invocations in order to understand how some concepts are implemented [11]. When programmers learn about a program (source code), the control flow (execution of function calls) needs to be followed. It means successive jumping from one function to another.

A code query helps to identify locations of interest in the source code. There exist many developer tools and environments that facilitate a developers' work. Short code fragments, which are returned to a programmer's query in current programming environments, do not provide enough "background" to help them how to reuse the fragments, and programmers usually have to invest considerable effort to understand how to reuse the fragments. When programmers write new code, they are often interested in finding definitions of functions, existing, working fragments, with the same or similar functionality and reusing as much of that code as possible. It is important that search engines support programmers in finding answers to similar questions and issues.

The most commonly used tools for code queries such as grep are based on purely text-based pattern matching. Even the present development environments, such as Eclipse and Visual Studio, require a great deal of learning effort. When solving a task, programmers usually have particular questions in their mind, such as "*Who implements this interface or these abstract methods?*" [11]. Such question often cannot be answered directly using existing functionality offered by development environments. Even if a particular conceptual query is directly supported, beginning programmers are not often familiar with their development environment, i.e., they are not yet aware of the integrated support features. For example, even though there is a feature "*Find references...*" in a context menu of Eclipse and we can easily answer to the query such as "*Where is this method called*", novice programmers still need to be aware that the feature - hidden in the context menu - is what they are looking for.

Keyword-based code search tools face the problem of low precision on their results due to the fact that a single word of the programmer's query may not match the desired functionality. It is because no source code content is analyzed or the programmer's needs are not clearly represented in the query.

Our main goal is to enable programmers to find relevant functions to query terms and their usages. In our approach, identifying popular fragments is inspired by PageRank algorithm, where the "popularity" of a function is determined by how many functions call it. We designed a model based on the vector space model, by which we are able to establish relevance among facts which content contains terms that match programmer's queries directly. The result is an ordered list of relevant functions that reflects the associations between concepts in the functions and the programmer's query.

This paper is structured as follows. The second section provides an overview of related work. The third section presents processing of the source code repository. In the Section 4, searching for relevant functions is presented. Finally, an evaluation is outlined in the Section 5.

## 2 Related Work

Current source code search engines are based on information retrieval approaches. Currently, text-based information retrieval systems are successfully used to locate relevant documents. Extraction of keywords from comments, names of functions and variables were often sufficient for finding reusable routines [5]. However, these source code search engines process code as plain text and extracted keywords have unknown semantics. In other words, the search engine compares query keywords to the names of the objects and retrieves matches. It is the most simplistic approach, which does not take into account additional information such as dependencies among objects.

Programs contain functional abstractions, which provide an essential level for code reusing. In other words, programmers define functions once and call them from different places in code. Approaches using functional abstractions to improve code search was proposed in [6,9]. However, these code search engines do not analyse how functions are used in the context of other functions, despite the fact that understanding the sequence of function invocations is one of the important questions that programmers ask [11].

Some approaches are based on programmer's query refinement. In [13] authors present approach, where queries and restrictions can be formulated in natural language,

for example, "*Give me details about the bidding process*". The presented approach is based on the use of domain models containing the objectives, processes, actions, actors and an ontology of domain terms, their definitions, and relationships with other domain-specific terms.

In [15] is presented an approach (CodeBroker) based on tracking what the programmer was doing. Instead of waiting for programmers to explore the reuse source code repository by using explicit queries, information delivery autonomously locates and presents components (code fragments) by using the programmers' partially written programs as implicit queries. Presented tool does the similarity analysis between components based on a concept similarity or constraint compatibility. The concept similarity is identified based on comments in the source code. A constraint similarity is identified based on the function (method) signatures. It further refines the query with inputs from the programmer.

An approach presented in [4] is based on using learning techniques. Authors focus on the task of searching for software in large, complex, and continuously growing libraries. They introduce the concept of active browsing, where an active agent tries to infer programmers' intentions and advise them.

CodeFinder presented in [7] is a tool which uses a query browser to help the programmer construct queries that can be sent to the repository. This helps the programmer make more effective queries. The tool is able to craft the query in a way that can be best used by the source code repository. CodeFinder is based on Spreading Activation algorithm to search sample source code. The advantage of CodeFinder is that it helps the programmer refine and reformulate the query. However, as the repository of sample code increases, Spreading Activation may provide some unrelated results.

Sourcerer [1] is a search engine for open source code that extracts fine-grained structural information from the code as a search basis. This information is used to enable search forms that go beyond conventional keyword-based searches.

Codifier [2] is a programmer-centric search interface (tool), which enable programmers to ask specific questions related to programming languages. It is based on indexing source code using modified compilers (C, C++, C#, VBScript) to extract lexical and syntactic metadata.

Although the mentioned tools are promising, they do not seem to leverage the various complex relationships which are presented in the source code and therefore have limited features. Developers have to spend considerable time by tracking relationships and the tools do not help them effectively organize the presentation of the relationships.

Our proposed method consists of two phases, namely processing of the source code repository and searching for relevant functions given a programmer's query.

## 3 Processing of the Source Code Repository

There are two main elements in the phase of the processing of the source code repository, namely an index creator and a function graph creator (see Figure 1). The index creator (A1) creates a document index and a term index (A2) from the source code repository. The purpose of the index creator is to enable to retrieve relevant functions based on matches between terms in programmer's queries and terms in the source code

files. The function graph creator (B1) creates a directed graph of functional dependencies (B2). The PageRank process (C) is run on the directed graph of functional dependencies, and it calculates a rank vector, in which every element is a score for each function in the graph.

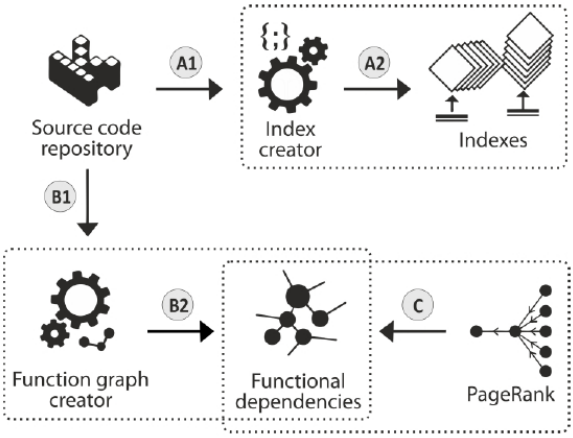



**Fig. 1.** Processing of the source code repository

### 3.1 Index Creator

The index creator creates indexes (document and term indexes) from all source code files (documents $D$) of projects in the repository. The creator uses the vector space model which is used by search engines to rank matching documents according to their relevance to a search query. By using our parser, from each document $d \in D$, terms $t$ are extracted from comments, names of functions and identifiers. For each term $t$, NLP (natural language processing) techniques, such as stemming and identifier splitting, are applied. Each document $d$ is modeled as a vector of terms which occur in that document.

For storing the indexes, we have adopted the distributed database management system Apache Cassandra[1]. It is a highly scalable, distributed and structured key-value store with efficient disk access. It is a hybrid between column-oriented DBMS and row-oriented store. Cassandra was especially designed to handle very large amounts of data. The created document index is structured as follows:

| Document | Document ID | Term | Term | ... |
|---|---|---|---|---|
| $d_1$ | $d_{1_{ID}}$ | $t_{ID}\|TF_{t_{ID},d_1}$ | $t_{ID}\|TF_{t_{ID},d_1}$ | ... |
| ... | ... | ... | ... | ... |
| $d_m$ | $d_{m_{ID}}$ | $t_{ID}\|TF_{t_{ID},d_m}$ | $t_{ID}\|TF_{t_{ID},d_m}$ | ... |

where for each $j = 1,\ldots,m$ ($m = |D|$ is the total number of documents), each row $d_j$ contains a unique document identifier $d_{j_{ID}}$; a list of pairs such that $t_{ID}$ is a unique

[1] Apache Cassandra: http://cassandra.apache.org/

identifier of a term which occurs in $d_j$ and $TF_{t_i}, d_{ID}$ is a calculated term frequency value for $t_{ID}$ and corresponding $d_j$.

The created term index is structured as follows:

| Term | Term ID | IDF | Count | Document | Document | ... |
|---|---|---|---|---|---|---|
| $t_1$ | $t_{1_{ID}}$ | $IDF_{t_1}$ | $\lvert d_{t_1}\rvert$ | $d_{ID}$ | $d_{ID}$ | ... |
| ... | ... | ... | ... | ... | ... | ... |
| $t_n$ | $t_{n_{ID}}$ | $IDF_{t_n}$ | $\lvert d_{t_n}\rvert$ | $d_{ID}$ | $d_{ID}$ | ... |

where for each $i = 1, \ldots n$, ($n$ is the total number of extracted terms), each row $t_i$ contains a unique term identifier $t_{i_{ID}}$; calculated inverse document frequency value $IDF_{t_i}$ (for calculation see below); the number of documents where the term $t_i$ occurs $|d_{t_i}|$; and a list of document identifiers $d_{ID}$ in which $t_i$ occurs.

Term frequency $(TF_{i,j})$ for term $t_i$ and document $d_j$ is calculated as follows:

$$TF_{i.j} = \frac{n_{i.j}}{\sum_k n_{k,j}}, \tag{1}$$

where $n_{i,j}$ is the number of occurrences of the term $t_i$ in the document $d_j$ and $\sum_k n_{k,j}$ is the sum of all occurrences of terms $n_k$ in the document $d_j$.
Inverse document frequency $(IDF_i)$ for term $t_i$ is calculated as follows:

$$IDF_i = log\frac{|D|+1}{|\{d|t_i \in d\}|}, \tag{2}$$

where $|D|$ the total number of documents in the corpus and $|\{d|t_i \in d\}|$ is the number of documents where the term $t_i$ occurs. The corpus represents the set of all source code files (documents) of all projects in the repository.

The $TF/IDF_{i,j}$ for term $t_i$ and document $d_j$ is calculated as follows:

$$TF/IDF_{i.j} = TF_{i,j} \times IDF_i. \tag{3}$$

### 3.2 Function Graph Creator

The purpose of the function graph creator is to construct a directed graph of functional dependencies. Nodes represent functions names (full signature of functions). A directed edge from the function $F$ to the function $G$is created if the function $G$ is invoked in the function $F$. In object-oriented programming languages, such as Java, C# or C++, there are problems, for example, with polymorphism, inheritance and method overloading. For solving this problem we use intermediate representation of source code, i.e., an abstract syntax tree (AST) and control flow graph (CFG) obtained from AST. The AST structures provide such information as nodes representing class definitions, member declarations, function (method) definitions, variable declarations, initialization and assignment statements, and method invocation statements. This allows us clearly to identify method invocations.

The algorithm of creating the graph of functional dependencies by using our parser and AST is as follows:

1. From a document $d$ (source code file), obtain full signature of a defined function $f$.
2. From the full signature of the function, create unique identifier $F_n$ using function invocation statement obtained from AST.
3. If there do not exist a node called $F_n$, create a new node called $F_n$.
4. For each function $g$ which is invoked in the function $f$:
   (a) Execute step 2 and then 3.
   (b) Create a directed edge from the node representing the function $f$ to the node representing the function $g$.
5. If there is an unprocessed function in the document $d$,execute step 1.
6. If there is an unprocessed document in the repository, select this document and execute step 1.

### 3.3 Ranking Functional Dependencies

For ranking of the functional dependencies, we use the PageRank algorithm. Using the PageRank, we are able to determine the "popularity" of a function. The PageRank of a function is defined recursively and depends on how many functions call (invoke) it. The rank value indicates importance of a particular function. On the other hand, following functional dependencies help programmers to understand how to use found functions, i.e. they can see and trace the sequence of function invocations.

The formula for PageRank of a function $f_i$, denoted $r(f_i)$, is the sum of the PageRanks of all functions that invoke $f_i$:

$$r(f_i) = \sum_{f_j \in T_{f_i}} \frac{r(f_j)}{|f_j|}, \tag{4}$$

where $T_{f_i}$ is the set of functions that invoke $f_i$ and $|f_j|$ is the number of functions that the function $f_j$ invokes. It is applied iteratively starting with $r_0(f_i) = 1/n$, where $n$ is the number of functions. The algorithm is repeated until the PageRank score converges to some stable values or it is terminated after some number of iterations. Functions called from many other functions, have a significantly higher PageRank score than those that are used infrequently.

## 4 Searching for Relevant Functions

The search phase (illustrated in Figure 2) enables programmers to find relevant functions to query terms and subsequently to trace their usages. Searching consists of three main steps. First, (top) relevant documents are retrieved based on a similarity $sim(d_j, q)$ between documents (source code files) and programmer's query $q$. Second, each document $d_j$ is divided into subdocuments, where each one contains only one definition of a function $F_n$ contained in the "parent" document $d_j$. For each subdocument $d_{jk}$, a similarity $sim(d_{jk}, q)$ to the query $q$ is calculated. Finally, an ordered list of relevant functions is obtained so that, for each function $F_n (F_n \in d_j)$, a final score $sc(F_n, q)$ is calculated as the sum of the similarities and a PageRank score $pr(F_n)$.

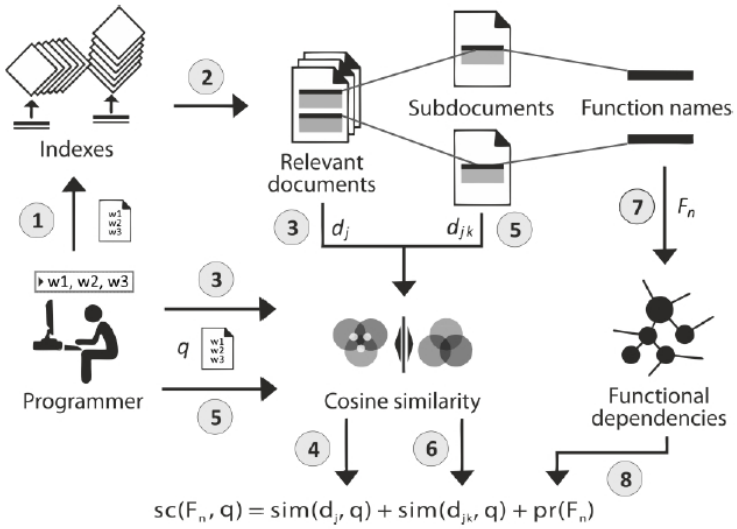



**Fig. 2.** Searching for relevant functions

### 4.1 Retrieving Relevant Documents

When a programmer enters a query (1), for each query term $w_i$, the NLP techniques are applied and subsequently a list of (top) relevant documents (source code files) is retrieved (2). The list contains documents, where at least one query term occurs in each document. A similarity (3) between two documents (query $q$ and a relevant document $d_j$) is calculated (4) using the cosine similarity (distance) as follows:

$$sim(d_a, d_b) = \frac{d_a . d_b}{\|d_a\| \|d_b\|}, \tag{5}$$

where $d_a, d_b$ are document vectors. Elements of the vectors are pre-calculated TF/IDF weights.

### 4.2 Subdocument Processing

1. Each retrieved relevant document is divided into subdocuments, where each one contains only one definition of a function with surrounded comments (if any).
2. From each subdocument, terms are extracted from comments, function name and identifiers.
3. For the extracted terms, TF/IDF weights are calculated (a subdocument vector).
4. For each subdocument $d_{jk}$, the cosine similarity (5) to the programmer's query $q$(6) is calculated.

### 4.3 Ranking of the Relevant Functions

1. From the relevant (sub)documents, unique identifiers $F_n$ are created using function invocation statements obtained from AST.

2. For each function $F_n$, a final score $sc(F_n, q)$ is calculated as a sum of:
   (a) a similarity $sim(d_j, q)$ between the document $d_j$ in which the function $F_n$ is defined ($F_n \in d_j$) and the programmer's query $q$ (4);
   (b) a similarity $sim(d_{jk}, q)$ between the subdocument $d_{jk}$ in which the function $F_n$ is defined ($F_n \in d_{jk}$) and the programmer's query $q$ (6);
   (c) a PageRank score $pr(F_n)$ for the function $F_n$ (7)(8) which represents "global popularity" of the function.

Based on our initial experiments we identified a problem with included support libraries (in the project). Consider the following situation when a programmer wants to find a function for compressing texture in the target project and she enters, for example, a query "compress, texture". However, "similar" function can be defined in a source code of an included support library. Moreover, this function can be called from many other functions defined in the library, too. In the case that a function for compressing texture is directly defined in the project we want to prefer this function. We solve this problem using weighting of functions of the supported libraries, i.e. we multiply their calculated score by so-called *damping factor* $\delta = 0.5$.

## 5 Evaluation and Conclusions

Typically, search engines are evaluated by experts whose task is to determine relevance of the results for a given query. We implemented a plugin into Microsoft Visual Studio which allows programmers to formulate a query and the result is an ordered list of relevant code fragments retrieved by our method. To determine how effective our method is, we conducted an experiment with 6 participants and 2 software projects, namely, ANNOR and The Green Game. These projects are written in C# programming language. ANNOR is an application for automatic image annotation and The Green Game (TGG) is a strategic computer game. Our goal was to evaluate how well these participants could find code fragments that matched given tasks. We divided the participants into 2 groups of 3 members (group#1, group#2) and we performed two experiments.

In the first experiment, we created a set of 6 change tasks, i.e., 3 tasks for each project. These participants reformulated the target tasks into a sequence of words that described concepts they needed to find. During performing the target tasks, the participants were asked to use only the search engines, i.e., supporting tools, such as the solution explorer and jumping, were prohibited.

The set of 6 change tasks, which we used in our experiment, is as follows:

1. Find the method for calculating a co-occurrence rank of obtained annotation and change the damping factor to the value 0.3.
2. Find the method for loading an input image and add a log event if loading the image fails.
3. Find the code fragment for extracting local features from training images and change the sigma parameter to the value 0.45.
4. Find the code fragment for changing color of a selected object (building) and change the current color to yellow.

5. Find the method which starts playing background music automatically in main menu and disable this feature.
6. Find the method for rendering a radial cursor and change the radius parameter to the value 10.

The tasks 1-3 were specified for the ANNOR project ($ANNOR_{tasks}$) and the tasks 4-6 were specified for the TGG project ($TGG_{tasks}$). The goal of the participants of the group#1 was to perform $ANNOR_{tasks}$ using our search engine. The goal of the participants of group#2 was to perform these change tasks using the built-in search engine. Subsequently, on the contrary, the participants of group#1 were tasked to perform $TGG_{tasks}$ using the built-in search engine and the participants of group#2 were tasked to perform these change tasks using our search engine.

After performing these tasks, we compared the number of participants' queries created using our search engine and the built-in search engine. The average number of participants' queries is shown in Table 1. We can see that by using our search engine, the participants had to make less effort for locating target code fragments (methods). It is confirmed by the average number of created queries for the target change tasks.

**Table 1.** Comparing average number of participants' queries

| Search engine | AVG number of queries |
|---|---|
| built-in search engine | 5 |
| our search engine | 2 |

In the second experiment, we specified 2 implementation tasks. These tasks were focused on reusing code fragments (methods) and they were formulated as follows:

1. ANNOR: Implement a tool for selecting rectangular area in the target image. Reuse the tool for selecting irregular polygon area as much as possible.
2. The Green Game: Implement a module for rendering an elliptic cursor. Reuse functionality for rendering the point cursor as much as possible.

The participants of group#1 were tasked to implement the first task using the built-in search engine whereas the participants of group#2 were tasked to implement the first task using our search engine. For implementing the second task, the participants of group#1 used our search engine and the participants of group#2 used the built-in search engine.

For each query, each participant evaluated relevance of the results. In other words, once the participants obtained lists of code fragments which were ranked in descending order, they examined these code fragments to determine if they matched the tasks. For a query and for each obtained result, a level of confidence, such as completely/mostly irrelevant, mostly/highly relevant, was assigned by the participant. Retrieved fragments were evaluated as relevant only if they are ranked with the confidence levels mostly or highly relevant, i.e., a retrieved code fragment is relevant to a task and the participant can understand how to reuse it to solve the task or it can be reused directly.

In our experiment, we used the precision metrics which reflects the accuracy of the programmer's search. The precision is the fraction of the top 5 ranked code fragments which are relevant to the query. The precision $P$ is calculated as follows:

$$P = \frac{the\ number\ of\ retrieved\ relevant\ code\ fragments}{the\ total\ number\ of\ retrieved\ code\ fragments}. \quad (6)$$

Since we limited the number of retrieved code fragments to top 5, the recall was not evaluated in this experiment.

The calculated average precision is shown in Table 2. It illustrates that search results, obtained using our search engine, were more relevant during performing the tasks compared with the use of the built-in search engine.

**Table 2.** Comparing average precision of search results

| Search engine | AVG precision |
|---|---|
| built-in search engine | 0.37 |
| our search engine | 0.64 |

Programmers often want to locate code fragments which are notable in a software project. Such fragments may represent, for example, internal patterns which should be used or reused during implementing certain functionality, and therefore they do not want to implement their own solutions from beginning. Our approach is able to locate such fragments, because in addition to the relevance of code fragments to a given query, our method establishes "popularity" of code fragments, i.e., it prefers such code fragments which support and motivate programmers in the process of reuse of existing solutions.

Our proposed method could be used in collaborative programming [3,8], too. For example, programmers can annotate code fragments based on identifying their "popularity" (among several projects). By adding new annotations to the source code such as *pattern/exemplar*, *good example*, there could be improved orientation of the programmers in the code through disclosure of the current state. The programmers would be able to see directly which parts of the code are interesting, stable or (often) reused.

**Acknowledgement.** This work was partially supported by the grants VG1/ 0675/1/ 2011-2014, 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. Bajracharya, S., et al.: Sourcerer: a search engine for open source code supporting structure-based search. In: Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications, NY, pp. 681–682 (2006)
2. Begel, A.: Codifier: A programmer-centric search user interface. In: Proc. of the Workshop on Human-Computer Interaction and Information Retrieval, pp. 23–24 (2007)

3. Bieliková, M., et al.: Webification of Software Development: General Outline and the Case of Enterprise Application Development. In: Procedia Technology, 3rd World Conf. on Inf. Tech., vol. 4 (2012)
4. Drummond, C.G., et al.: A Learning Agent that Assists the Browsing of Software Libraries. IEEE Trans. Softw. Eng., 1179–1196 (2000)
5. Frakes, W.B., Pole, T.P.: An Empirical Study of Representation Methods for Reusable Software Components. IEEE Trans. Softw. Eng. 20, 617–630 (1994)
6. Grechanik, M., et al.: A search engine for finding highly relevant applications. In: Proc. of the 32nd ACM/IEEE Int. Conf. on Softw. Eng., ICSE 2010, pp. 475–484. ACM, NY (2010)
7. Henninger, S.: Retrieving software objects in an example-based programming environment. In: Proc. of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1991, pp. 251–260. ACM, New York (1991)
8. Kramár, T., Barla, M., Bieliková, M.: Personalizing Search Using Metadata Based, Socially Enhanced Interest Model Built from the Stream of User's Activity. Journal of Web Engineering 12(1) (2013)
9. Ossher, J., et al.: SourcererDB: An aggregated repository of statically analyzed and cross-linked open source Java projects. In: Proc. of the 2009 6th IEEE Int. Working Conf. on Mining Softw. Repositories, MSR 2009, pp. 183–186. IEEE CS, Washington (2009)
10. Morisio, M., et al.: Practical Software Reuse. Springer, London (2002)
11. Sillito, J., et al.: Asking and Answering Questions during a Programming Change Task. IEEE Trans. Softw. Eng. 34(4), 434–451 (2008)
12. Sim, S.E., et al.: Archetypal Source Code Searches: A Survey of Software Developers and Maintainers. In: Proc. of the 6th International Workshop on Program Comprehension, IWPC 1998, pp. 180–187. IEEE Computer Society, Washington (1998)
13. Sugumaran, V., Storey, V.C.: A semantic-based approach to component retrieval. SIGMIS Database 34, 8–24 (2003)
14. Prieto-Diaz, R., Freeman, P.: Classifying Software for Reusability, pp. 6–16. IEEE (1987)
15. Ye, Y., Fischer, G.: Supporting reuse by delivering task-relevant and personalized information. In: Proc. of the 24th International Conference on Software Engineering, ICSE 2002, pp. 513–523. ACM, New York (2002)

# SimCo – Hybrid Simulator for Testing of Component Based Applications

Richard Lipka, Tomáš Potužák, Premek Brada, and Pavel Herout

Department of Computer Science and Engineering,
University of West Bohemia Plzeň, Czech Republic
{lipka,tpotuzak,brada,herout}@kiv.zcu.cz

**Abstract.** Testing of component-based applications is important in order to ensure that third-party components do not compromise the functionality or properties of the whole system. However, thorough testing of functionality, behaviour and extra-functional properties is a tedious and time consuming process. In this paper we present an approach to discrete event simulation testing of components and component sets. Its unique feature is the ability to execute a mixture of real, implemented components and simulated mock-ups of the remaining parts of the application. Together, this approach allows faster testing on a wide scale of different inputs for tested components. At the same time, the use of actual components increases the confidence in the simulation test results. The approach has been implemented using the OSGi platform in the form of the SimCo framework and toolset, for which the key architectural considerations are discussed together with a short case study illustrating its usage.

**Keywords:** software component, testing, simulation, performance, OSGi.

## 1 Introduction

In computer science and software engineering, component based development is becoming more and more widespread due to maximized reusability of software and also development simplified by using third party components. This tendency is more visible with the expansion of mobile devices. In a mobile device environment, an application obtained from a third party through an application repository like Apple's AppStore can be described as a component which is using the components provided by the device's OS and other applications.

Testing of such applications is very important in order to ensure that the third-party application will not compromise the functionality or properties of the whole system. Such testing often has to be performed by the maintainer of the application repository and not only by the author of the application – the maintainer need not be responsible for the proper functionality of the applications but should ensure that they are safe and without performance problems for his clients. Therefore black box testing is one of the important techniques in this field. Also, for this type of cases, extra-functional properties are more important than the functional ones. However, preparing and executing sufficiently thorough tests for large number of applications or components is a challenging task from technical, process and time perspectives.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 420–431, 2013.

In this paper, we are presenting a tool for testing real software components in a simulated environment. Because performance of an application is not a simple product of the performance of each component, we have designed our tool to be able to execute tests not only on one component but also on an arbitrary set of components or on the whole application. Using discrete event-based simulation and simulation components, the tests on real components provide useful information about their real performance and, at the same time, might be executed in a shorter time.

The rest of this paper is structured as follows. In Section 2 we review the foundational ideas and research work related to component simulations. Section 3 provides detailed description of the simulation tool and its internal components. Use of the resulting approach is illustrated by two short case studies in Section 4, after which a conclusion closes the paper.

## 2 Background on Component Testing and Simulation

In the component approach, the whole application can be created from a set of individual software parts called components. They are considered to be black box entities, with clear definition of its interface and functionality (sometimes called behaviour) but without observable inner state [1] and knowledge of their inner working. Because all communication within component application has to be performed only through defined interfaces, no hidden dependencies should exist and thus it is easier to substitute one component with another. Component model specifies how software components look, behave and interact, while a component framework is an implementation of a specific component model, providing the required infrastructural functionality.

The designer of a component application usually works with components stored in a repository [11]. Often, the correctness of the functionality of stored components is considered to be implied as they should be tested in a standard way by developers before they are deployed into the repository. For the designer of an application, information about both component's interface features and performance or other extra-functional properties might be useful. The latter information is often obtained by executing simulation tests of the components, as e.g. in the Palladio approach [2–4].

Discrete event simulation is an often used method for simulation based testing or system analysis. Each event contains a time stamp when it should occur in simulation time and an action which shall be executed [8]. All events are handled by a calendar. When the simulation is started, the calendar chooses the event with the earliest time (the smallest value of the time stamp), sets simulation time to the time of the event and performs the action of the event. Then, next event according to its time stamp is selected from the calendar and executed and so on. Each event can cause the creation of one or more new events, which are being added to the calendar [8].

In discrete event simulation, time between two events can be arbitrary long or short, thus it allows using a fine division of time. However, it is advantageous mainly when events are not abundant due to the overhead connected to event execution.

In the area of simulation testing in component oriented development, tools and frameworks based on component simulation are commonly used. Their development began before the year 2000 [9–11] but the research in this field continues till today [12–15].

The frameworks are utilized either for general simulations (such as discrete event simulation [12]) or are focused more specifically (such as simulation of computer networks [14]).

Generally, the research of component-based simulations is focused on the relationship between the simulated components and their composability while using a specific framework [12, 13]. The effort to use components in a distributed computing environment is also very common [16–18]. Besides the focus on specific frameworks, there are also attempts to create formalism for component-based simulations [17]. The variability of such simulations in terms of application reconfiguration is often mentioned as the most advantageous feature of component-based simulations [14].

Simulation testing of components is rarely focused on the functionality of components, since their correct functionality is considered to be implied. Often, the main goal of simulation tests is verification of quality of services (QoS) and extra-functional properties, as is described in [15, 20]. All tests of components are black-box tests with exception of self-testing (see [19] for example). Because distributed use of components is often expected, there is a need to test them for this usage [21].

However, it is necessary to consider that the component's functionality may not be tested properly by its developer, or that black-box component of the third party might be a security risk [21]. We therefore believe it is important to focus on testing of component functionality as well as their suitability for different hardware configurations, as they may be deployed on different devices.

Further, it should be noted that real components are rarely tested in simulation tools. Instead, only their models are in the vast majority of approaches used for the simulation purposes as e.g. in Palladio [2]. These models are often created using a static description, such as resource consumption, behaviour description, and so on [15, 20]. Specific descriptive languages have been created for these purposes [4, 15]. Use of models instead of real components brings the issue that the model may not accurately reflect the component's properties and that the results based on its usage cannot be reliably used for reasoning about the components.

## 3 The SimCo Simulation Tool

One idea in using of components is to have them stored in repository as stand-alone applications. This approach is used even in specific domains, e.g. the Openmatics portal [22] where applications form repository are installed into hardware inside vehicles and provide different kinds of functionality, such as position reporting, telemetry and so on. Such components can be created by third party and it is important to test their functionality, behaviour and extra-functional properties (like duration of computations or amount of data flowing through a network interface) before they are offered to customers through the repository, in order to ensure that they will not cause any undesirable effects. Black box test with using of simulation tool may provide convenient way of testing.

Therefore, we designed SimCo simulation framework and toolset for testing of real, implemented components in simulated environment. Its aim is to enable the use of obtained results in comparing different components or applications, as well as for

calibrating other simulation models such as Palladio. The remainder of this section describes the overall approach, technical and architectural design of the simulation framework, and its usage.

### 3.1 General Approach

Components in real applications are expected to perform actions (such is an invocation of another's component method) driven by internal component logic, user inputs or external asynchronous events like sensor interrupts. In other words, the time between these communication events is unpredictable, variable and can span long intervals in real time (seconds or even minutes) which makes standard testing of such applications complicated and time demanding or, alternatively, requires changes in tested components or the environment to cause the events to occur more often

Also, we may assume that there will be relatively long periods of time when an application component waits for the result of another component's service. This time slows down the testing process but the results of these services may be important for the application functionality and control flow, especially when handling of extreme or invalid values is being tested.

For both of these reasons we are using discrete event simulation which enables us to (1) speed up the process of testing of components by means of quickly providing pre-processed values through the simulation environment and performing the tests in simulation time, and (2) have a way of creating complex simulation scenarios with a wide range of inputs for components.

The main feature of our approach is however the use of *hybrid simulation* where real components and simulated components are mixed during the testing of the whole application. When the tests are focused only on a part of application or a single component, the rest of the application can be replaced by mock-ups (simulated components) which require a considerably lower amount of time for their execution. The tests will nevertheless be performed on real components themselves without any changes to their implementation. This is consistent with the black box understanding of components and important since the results can be relied on (unlike the model-based approaches).

The simulation tool is designed to test especially extra-functional properties of components, such as duration of computation or amount of data flowing through a network interface. However, it can as well be used to verify standard functionality and state-based behaviour.

### 3.2 Component Technologies Used

One of our aims is that the simulation framework itself be component-based (mainly for extensibility reasons) and that the proposed methods be relevant to the current state of the art component models. There are several industrial component models that were consequently considered as good candidates for the framework implementation, and the following frameworks were chosen since they provide a good balance between the adherence to the principles of component-based programming and industrial relevance.

**The OSGi Framework and Its Implementation.** The OSGi framework [5] describes a dynamic component model and offers service platform for Java programming language where component-based applications and components alone can be remotely installed, started, stopped, updated and uninstalled without requiring application restart [6]. The OSGi framework is commonly used in many different industry areas such as automotive industry, cell phones, portable devices, software development and so on [6]. For SimCo implementation, we use the Equinox OSGi implementation. OSGi components (called bundles) communicate through services which are implementations of specific interfaces. When a bundle is installed into the framework, its exported interfaces (services that are provided by the bundle) are registered, so other bundles may ask framework for bundle providing these services. The registration of a service is done directly in the code of the bundle, or by the Component Service Runtime for services declaratively specified in the manifest.

**Spring.** The Spring framework [6] offers a variety of features to support the development of enterprise-grade applications. The most characteristic features of the Spring framework are inversion of control, aspect-oriented programming, data access, transaction management and remote access. Spring also offers easy configuration of class-based elements called Spring beans as well as their dependencies through an XML configuration file.

**Spring Dynamic Modules.** The Spring Dynamic Modules extension for OSGi service platform enables the development of OSGi components using Spring framework. Moreover, SpringDM improves manageability of the OSGi services. The main advantage of using SpringDM is a transfer of bundle service dependencies from the code of Java classes into XML configuration files. The code of the bundle is then easier to develop and some changes in registered and used services can be performed only in configuration files, without the need to change the source code of the bundle.

### 3.3 Structure of the Simulation Application in SimCo

As was stated above, in the simulation tool, real and simulated components are used together. In order to provide the needed facilities for component testing and measurements, the whole simulation is composed from four types of components: framework, real, simulated, and intermediate ones.

The first type are *framework components* which constitute the SimCo simulation core and provide supporting features. These components ensure the functionality of the framework and also provide basic services necessary for the simulation. Among these components, a component controlling the calendar is the most important one. It contains all events which occur during the simulation run. Consequently, it controls the progress of the simulation based on the loaded scenario (the scenarios will be described later). Additional components ensuring other functionality, such as logging or measuring performance, belong to this group as well.

Second type is represented by the *real components* of the component-based application under test. There can be one or more real components in one simulation experiment.

They can interact with each other and with the simulation environment. As mentioned above, SimCo requires no changes in the implementation of these components, so they are the same as those which will be deployed in the real application.

If testing of real component's provided or used services is not the goal of the simulation experiment, they can be replaced by their simulation equivalents. Therefore, *simulated components* export the same interface as their real counterparts but the actual computation is replaced by a model according to decision of the designer of the experiment, e.g. a random number generator or a list or pre-processed results, provided as an answer for a service call from the real component.

In order to measure real components' performance and keep the simulation consistent, we need to intercept all events in the simulation even when they are passed directly between a pair of real components. Hence the fourth type of components has to be used, called *intermediate components*, which serve as proxies for the real components. All calls of the component hidden behind this proxy are noted and then passed to the real component, and likewise the answers (returned values) are returned through this proxy. The intermediate component also allows us to model deployment of real components in the distributed environment, as it may be set to cause delays of calls or even to induce errors into the communication.

One of the biggest advantages of using SpringDM is that it is possible to change the components used in the final application only by modifying XML documents that describes composition of the application. This allows us to easily replace real components for their simulated counterparts or to place intermediate components between two real ones.

### 3.4 Considerations Related to the Hybrid Simulation

Depending on the position of the simulated component, it may be required to provide a more complex behaviour than described above. An example is passing events to other components. The settings of each simulated component are stored in its configuration file which however does not contain the description of its behaviour. The behaviour has to be implemented inside the component.

Depending on the topology of the tested application, there are several possible configurations of the inter-component connections, described in Figure 1 below. First, the real components may be connected only to simulated components or to other real components (Figure 1-c). If there is a chain of simulated components attached to a real one (see Figure 1-a), it might be considered to replace the whole chain by implementation of simulation of component A, as it is the only one which interacts with the real component. However, if the simulation models of all components in the chain have been already created, it saves time of the experimenter to use them instead of creating the new implementation.

More complex situation occurs when simulation component is surrounded by two real components (see on Figure 1-b). In this case, it is necessary to consider the possibility that, in reaction to an action invoked by the real component B, simulated component invokes after some computation an action upon the real component A (and even that return value from the component A is passed as a return value for original invocation

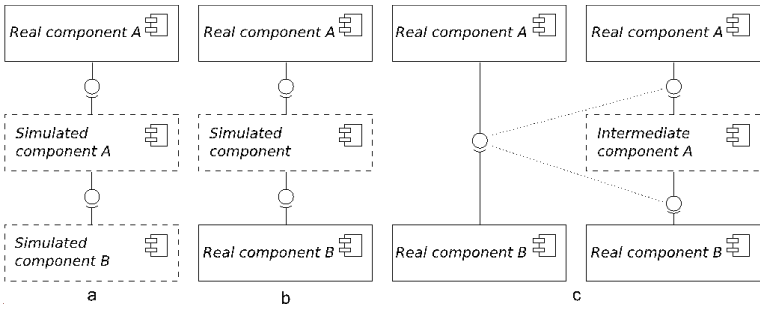



**Fig. 1.** Deployment of simulated and intermediate components

from the component B). In such case, behaviour of simulated component must respect this and it cannot be modelled only by using pre-calculated values.

Lastly but probably most importantly, real components within the simulation should not be aware that they are not the part of a real application but they are deployed in a simulated environment. Hence, from their point of view, simulation environment must act as the environment for which they were originally developed. On the other hand, both the simulation environment and the real components must be under complete control of the simulation tool. Therefore, all interactions among the real as well as simulated components must be performed by using the calendar and events so that we can log all interactions and obtain the desired measurements [23]. For this purpose, all services provided by the real components are wrapped by intermediate components.

### 3.5 Scenarios and Events

To support complex testing of components, our tool uses scenarios to drive the simulation tests. They allow us to describe the environment and activities of simulated components in a great detail.

The testing scenario is divided into three parts. In the first part, the settings of the entire simulation are described. An example of such setting is the value of simulation time at which the simulation should finish.

In the second part, all the components of the simulation are described. The settings of all components are described separately (one settings XML file per component). For the simulated components, these additional files also contain the description of the components' settings (e.g. seeds and parameters of random number generators). An example can be the return values of the services provided by the simulated components in dependence of their input values.

The third part of the scenario is the description of events which can occur during the simulation. Each event has a source and a target. The source may lie within the simulated system (i.e. simulated and real components) or in outer environment, in order to simulate both behaviour of components and also the actions of users and other outer inputs. The target is always within the simulated system. The event can have parameters relevant to its purpose. For example, an event representing processing of some graphical data would have the path to the data file as its parameter.

Events can be divided in accordance with their occurrence into three classes – regular, casual, and rare. The *regular events* are created periodically during the whole simulation run or during a selected/certain period of time. They can be events based on timers (e.g. periodical checking of changes of a component).

The *casual events* can occur often, but not periodically. However, the time of their occurrence may be described by probability distribution. They could be for example used to represent requests of users.

The *rare events* occurs so infrequently that their probability distribution cannot be identified easily. They even may not occur at all during the simulation run. An example of such event may be an accident, failure, damage of network hardware, and so on. The timestamp of such events can be set differently in different scenarios for one set of tested components. So, it is possible to observe the behaviour of the tested components when the events happen in various points of the simulation time.

Use of scenarios is an important benefit of our simulation tool. It allows controlling the size and length of tests without the necessity to manipulate with tested components. The scenario allows the tester to induce a large number of rare situations in a short time if it is necessary and thus speeding up testing. Scenarios are stored in separate files, so a set of scenarios (and tests) can be stored with the application, to be used when some component of application is changed.

## 4 Examples of SimCo Based Experiments

In this section we illustrate the operation and usage of SimCo on an example component-based application. The experiments were created to represent testing of a typical component-based application without dependencies on any specific hardware.

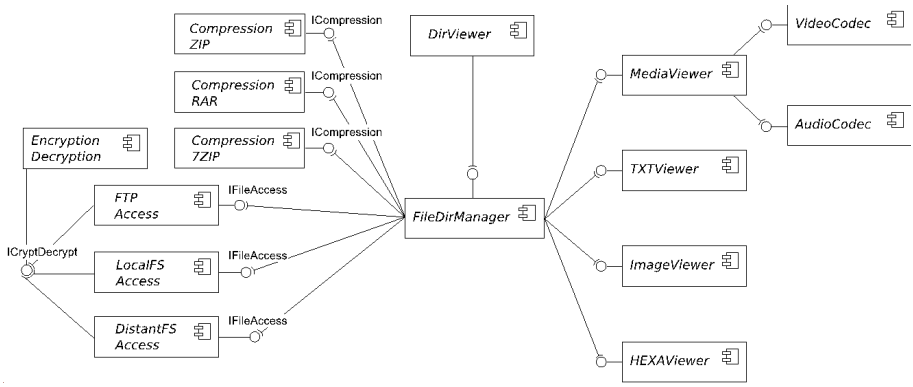



**Fig. 2.** Tested application – File manager

For these purposes we are using a simple File Manager application with several functions. Its architecture is outlined in Figure 2. Due to its purpose, the file manager requires a wide variety of different services, which can be implemented with different quality of services and consequently tested and arbitrarily replaced. It allows us to test performance of several different algorithms or to test properties of network communication.

### 4.1 Performance Experiment

This experiment is designed to demonstrate the use of our simulation tool. The test is focused on the speed of HEXAViewer component. Two different implementations of HEXAViewer were implemented. One preloads the whole file before displaying it. The other one displays only part of the file which will fit to the screen and loads additional data only when required. (This means that time of opening of the first one has a linear dependency on the size of opened file and the second one has a constant time). Three real components are used, HEXAViewer, FileDirManager and LocalFS Access. The DirViewer component is simulated (see Figure 3). As DirViewer component is a GUI of the application, its simulation allows us to simulate the behaviour of the user without the necessity to work directly with the GUI.

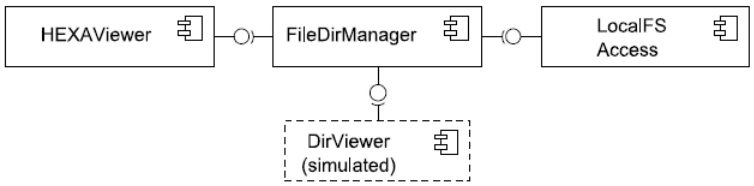



**Fig. 3.** Deployment of components for performance testing of HEXAViewer

The behaviour of the DirViewer is set to open ten times each input file with randomly generated content. We used 14 input files with different lengths and measured time between request for opening of the file and moment when the window is created. The

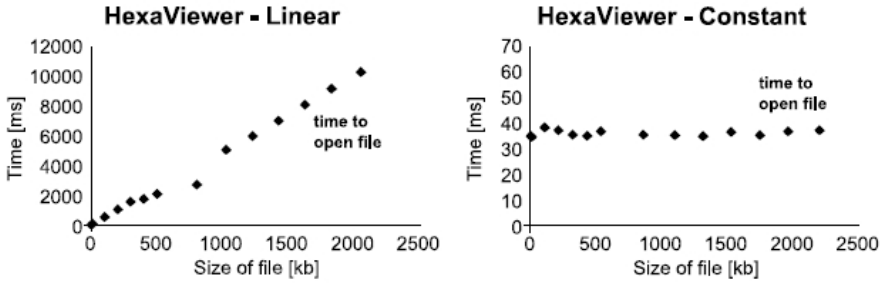



**Fig. 4.** Measured results

simulation tool captures the time of invocation of the file opening operation between DirVewer and FileDirManager and the time of the event created by HEXAViewer informing the application that the window with file is displayed.

The results at Figure 4 show the measured time requirements. An average duration (in milliseconds) of opening of each file is shown for both implementations of HEXAViewer. The results cannot be interpreted only as time required by HEXAViewer to open the file, as duration of other activities in the application influences them as well (e.g. obtaining the file from LocalFS Access), but shows relative performance of two different variations of the application and thus might be used to decide which one is more suitable.

### 4.2 Communication Experiment

This experiment shows the ability of SimCo to measure the amount of communication of a tested component with outside environment. This is particularly important for testing of component based applications for mobile devices, where communication through the cell phone network can lead to cost increase for their users.

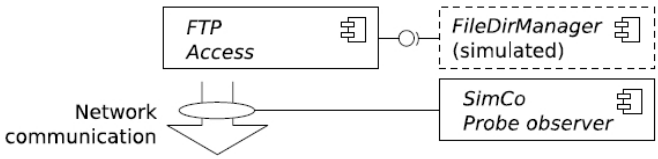



**Fig. 5.** Deployment of components for communication measurement

In the experiment, FTP Access component is used, along with simulated FileDirManager (see Figure 5). FileDirManager provides input for FTP Access and initiates download of ten different files. A probe based on the *libpcap* library is used to measure the amount of transferred data. The probe is capable of analyzing packets of transport and network layer of TCP/IP stack. If analysis of the application protocol is required it has to be added as a plug-in to the probe. We have tested that all required files were transferred and no data were lost. More important, we have tested that we are able to measure amount of such communication with all its overhead.

## 5 Conclusion

In this paper, we have described the design of SimCo, a component-based hybrid simulation tool for testing real components within a simulated environment. In comparison with other simulation testing tools and strategies, we enable the possibility to choose arbitrary part of the tested application and replace the rest by simulation. This enables to make tests of a specific part of the application faster than when the whole application would be tested. At the same time, the important tested components are real, not only their models, so the results describe their real properties. Use of test scenarios which

drive the simulation also allows us to store a wide range of tests for each application without the necessity to modify the application itself or to have the application prepared for testing in the way used by unit tests tools like JUnit.

The core of the simulation tool – which is itself component based – is complete now; however there is still remaining work on the automation of repeated tasks and especially on the usability of the tool. So far, the GUI serves only as visualization; we would like to enhance it to allow at least visual editing of the properties of the simulated components and on-line display of the measured results. We are also preparing a second case study to demonstrate the possibilities of the framework.

The remaining issue is handling of API calls problematic from the simulation point of view (time handling, network communication, etc.), so they will be managed completely by the simulation tool. The possibilities of such handling and the hazards of uncontrolled calls are topic of our currently ongoing research.

**Acknowledgements.** This work was supported by the Czech Science Foundation under the grant number 103/11/1489 "Methods of development and verification of component-based applications using natural language specifications".

## References

1. Szyperski, C., Gruntz, D., Murer, S.: Component Software – Beyond Object-Oriented Programming. ACM Press, New York (2000)
2. Becker, S., Koziolek, H., Reussner, R.: The Palladio component model for model-driven performance prediction. Journal of Systems and Software 82(1), 3–22 (2009)
3. Heam, P.C., Kouchnarenko, O., Voinot, J.: Component Simulation-based Substitutivity Managing QoS Aspects. Electronic Notes in Theoretical Computer Science 260, 109–123 (2010)
4. Cansado, A., Henrio, L., Madelaine, E., Valenzuela, P.: Unifying Architectural and Behavioural Specifications of Distributed Components. Electronic Notes in Theoretical Computer Science 260, 25–45 (2010)
5. The OSGi Alliance: OSGi Service Platform Core Specification, release 4, version 4.2 (2009)
6. Rubio, D.: Pro Spring Dynamic Modules for OSGiTM Service Platform. Apress, USA (2009)
7. Brada, P., Jezek, K.: Ensuring Component Application Consistency on Small Devices: A Repository-Based Approach. In: Proceedings of the 38th Euromicro SEAA Conference. IEEE Computer Society Press (accepted for publication, 2012)
8. Fujimoto, R.M.: Parallel and Distributed Simulation Systems. John Wiley & Sons, New York (2000)
9. Miller, J.A., Ge, Y., Tao, J.: Component-Based Simulation Environments: JSIM as a Case Study Using Java Beans. In: Proceedings of the 1998 Winter Simulation Conference, Washington, DC, pp. 373–381 (1998)
10. Pidd, M., Oses, N., Brooks, R.J.: Component-Based Simulation on the Web. In: Proceedings of the 1999 Winter Simulation Conference, Phoenix, pp. 1438–1444 (1999)
11. Harrell, C.R., Hicks, D.A.: Simulation Software Component Architecture for Simulation-Based Enterprise Applications. In: Proceedings of the 1998 Winter Simulation Conference, Washington, DC, pp. 1717–1721 (1998)
12. Buss, A., Blair, C.: Composability and Component-Bases Discrete Event Simulation. In: Proceedings of the 2007 Winter Simulation Conference, Washington, DC, pp. 694–702 (2007)

13. Moradi, F., Nordvaller, P., Ayani, R.: Simulation Model Composition using BOMs. In: Proceedings of the Tenth IEEE International Symposium on Distributed Simulation and Real-Time Applications, Malaga (2006)
14. Rao, D.M., Wilsey, P.A.: Multi-resolution Network Simulations using Dynamic Component Substitution. In: Proceedings of the 9th Int'l Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, Cincinnati (2001)
15. Becker, S., Koziolek, H., Reussner, R.: The Palladio component model for model-driven performance prediction. The Journal of Systems and Software 82, 3–22 (2009)
16. Verbraeck, A.: Component-based Distributed Simulations. The Way Forward? In: Proceedings of the 18th Workshop on Parallel and Distributed Simulation, Kufstein (2004)
17. de Lara, J.: Distributed Event Graphs: Formalizing Component-based Modelling and Simulation. Electronic Notes in Theoretical Computer Science 127, 145–162 (2004, 2005)
18. Wainer, G.A., Madhoun, R., Al-Zoubi, K.: Distributed simulation of DEVS and Cell-DEVS models in CD++ using Web-Services. Simulation Modelling Practice and Theory 16, 1266–1292 (2008)
19. Yao, Y., Wang, Y.: A Framework for Testing Distributed Software Components. In: Annual Canadian Conference on Electrical and Computer Engineering, Saskatoon, pp. 1566–1569 (2005)
20. Becker, S., Koziolek, H., Reussner, R.: Model-Based Performance Prediction with the Palladio Component Model. In: Proceedings of the 6th International Workshop on Software and Performance, Buenos Aires (2007)
21. An, G., Park, J.S.: Cooperative Component Testing Architecture in Collaborating Network Environment. In: Xiao, B., Yang, L.T., Ma, J., Muller-Schloer, C., Hua, Y. (eds.) ATC 2007. LNCS, vol. 4610, pp. 179–190. Springer, Heidelberg (2007)
22. Openmatics. Applications (2012), http://www.zf.com/brands/content/en/openmatics/-products_services/apps/apps_openmatics.html (cited June 28, 2012)
23. Potuzak, T., Snajberk, J., Lipka, R., Brada, P.: Component-based Simulation Framework for Component Testing using SpringDM. In: Annals of DAAAM for 2010 & Proceedings of the 21st International DAAAM Symposium, Zadar, vol. 20(1) (2010)
24. Šimko, V., Hnětynka, P., Bureš, T.: From Textual Use-Cases to Component-Based Applications. In: Lee, R., Ma, J., Bacon, L., Du, W., Petridis, M. (eds.) SNPD 2010. SCI, vol. 295, pp. 23–37. Springer, Heidelberg (2010)

# Refinement Inference for Sequence Diagrams

Lunjin Lu and Dae-Kyoo Kim

Oakland University
lunjin@acm.org

**Abstract.** Refinement is fundamental to software development. An earlier work proposed a refinement relation between sequence diagrams based on their required behaviors. In this paper, we first generalize the refinement relation by taking into account system variable assignment and event hiding, renaming and substitution. We then give an algorithm as a system of inference rules that does not just verify refinement relationship between two sequence diagrams but also derives sufficient conditions under which such a relationship holds. The algorithm makes use of a semantics preserving transformation on sequence diagrams. The usefulness of refinement inference is demonstrated with a case study.

**Keywords:** sequence diagrams, semantics of sequence diagrams, refinement of sequence diagrams, refinement verification, refinement inference, sequence diagram transformation, pattern conformance.

## 1 Introduction

A fundamental issue in using UML [12] to specify interaction behavior is whether one sequence diagram (SD in short) refines its predecessor: it possesses all mandatory behaviors that are required by the predecessor and rejects all proscribed behaviors that are prohibited by its predecessor. Refinement verification is equally important in other forms of reuse of SD models. Many reusable artifacts such as design patterns and aspect models make use of SDs to specify interaction behaviors. If an application uses a reusable artifact, the SDs in the application must be refinements of their corresponding SDs in the artifact. Otherwise, the intended benefits of the reusable artifact cannot be guaranteed.

There has been some research into refinement of SDs in literature. Refinement has been studied based on trace semantics [4,6,11]. In [4,11], the semantics of an SD is a pair consisting of a set of positive traces and a set of negative traces. Without the fragment operator *xalt* which Haugen et al. [6] introduced to capture the mandatory non-determinism, the semantics proposed by Haugen et al. [6] is equivalent to that of Störrle's [11]. As pointed in [10,8], the set of positive traces does not capture precisely good behaviors required by the SD.

In an earlier work [8], we formulated a semantics for SDs that captures precisely required behaviors of an SD and defined a refinement relation based on that semantics. An SD is partial in that it describes a number of alternative obligations that an implementation may choose to fulfil. For instance, the fragment

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 432–444, 2013.

operator *par* does not mandate that an implementation must be distributed, concurrent or multi-threaded. It rather indicates that the implementation can realize any interleaving of the behaviors of its operands. When an SD is reused, it is made more defined in that the number of alternatives is reduced. In this paper, we first generalize the refinement relation in the earlier work by taking into account system variable assignment and event hiding, renaming and substitution. We then give an algorithm for refinement inference as a system of inference rules. We also generalize the abstract syntax for SDs and present a semantics preserving transformation on SDs. This allows a tool to transform SDs into a normal form. This has the benefits of simplifying design of SD processing algorithms for refinement inference.

The rest of the paper is organized as follows. Section 2 presents the abstract syntax, the trace semantics and the refinement relation for SDs. Section 3 describes a transformation that puts an SD into a normal form. Section 4 presents the inference rules. Section 5 shows how refinement inference can be used to check pattern conformance and Section 6 concludes. Proofs are omitted.

## 2 Syntax, Semantics and Refinement

### 2.1 Syntax

A simple SD which does not have any combined fragment has been modelled as a partial order on event occurrences [3]. In [8], a partial order on arbitrary SDs is used to organize operands of fragment combination operators. This paper proposes to use two partial orders to organize these operands. This new abstract syntax admits a normal form of SDs which simplifies SD manipulation tools.

Let $\mathbb{Name}$ be a denumerable set of names of messages, lifelines and system variables and $\mathbb{Values}$ the set of possible values for system variables. An event sending a message with name $N \in \mathbb{Name}$, sender $S \in \mathbb{Name}$, receiver $R \in \mathbb{Name}$, parameter list $P \in (\mathbb{Name} \cup \mathbb{Values})^*$ is written as $!N(S, R, P)$, and the corresponding receiving event $?N(S, R, P)$. We abstract from details of guard conditions $c$ in $\mathbb{Cnd}$ and require that the collection of guard conditions is closed under classical logical negation ($\neg$), conjunction ($\wedge$) and disjunction ($\vee$) operations. We write $c_1 \models c_2$ iff $c_2$ is true in all value assignments in which $c_1$ is true. Events $e \in \mathbb{Evt}$ are primitive syntactic entities. Other primitive syntactic entities are labels $\ell$ in $\mathbb{Lab}$ and $\tau$ which represents unobservable events. The abstract syntax for SDs in $\mathbb{Sd}$ is given below.

$$\begin{aligned} D ::= \quad & \tau \mid e \mid \mathit{opt}(c, D_1) \mid \mathit{alt}(c, D_1, D_2) \mid \mathit{loop}(c, D_1) \mid \mathit{par}(D_1, D_2) \\ & \mid \mathit{strict}(D_1, D_2) \mid \mathit{seq}(D_1, D_2) \mid \mathit{block}(L, \iota, \twoheadrightarrow, \dashrightarrow) \end{aligned}$$

where the interaction operator *block* is introduced to structure operands of other interaction operators, $L \subseteq \mathbb{Lab}$ is a non-empty set of labels, $\iota$ is a mapping from $L$ to $\mathbb{Sd}$, $\twoheadrightarrow$ and $\dashrightarrow$ are irreflexive and non-transitive relations on $L$ such that $\twoheadrightarrow^*$ and $\dashrightarrow^*$ are partial orders. The relation $\dashrightarrow$ is included in anticipation of SD normalization that eliminates weak sequencing interaction operator *seq*.

A sequence of events satisfies $\dashrightarrow$ iff, for all $\ell_1, \ell_2 \in L$ such that $\ell_1 \dashrightarrow^* \ell_2$, each event $e$ from $\iota(\ell_1)$ occurs before all those events from $\iota(\ell_2)$ that share lifelines with $e$. $\langle L, \iota, \twoheadrightarrow^* \rangle$ and $\langle L, \iota, \dashrightarrow^* \rangle$ are partially ordered multisets. Compared with [8] which uses one ordering relation $\twoheadrightarrow$ to organize sub-SDs in a block, this new syntax uses two ordering relations $\twoheadrightarrow$ and $\dashrightarrow$. This change may seem minor, but it turns out to be a powerful addition that allows us to normalize SDs by eliminating *strict*, *par* and *seq* operators.

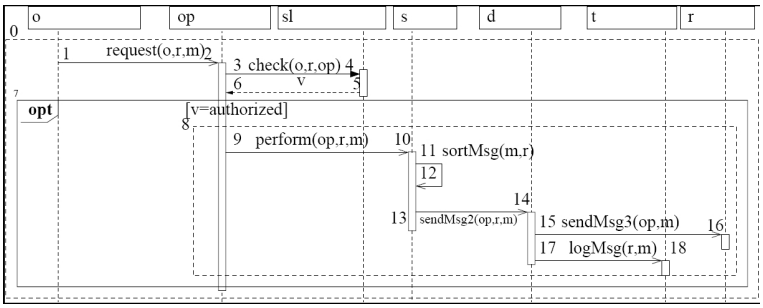



**Fig. 1.** An SD for Mandatory Access Control (MAC)

*Example 1.* Consider the SD in Fig. 1 where fragments and events are labelled. In particular, the *opt* fragment is labelled 7. The sending and receiving events for a message are labelled with two consecutive numbers. Let $e_i$ abbreviate the event labelled $i$. For instance, $e_1$ abbreviates $!request(o, op, \{o, r, m\})$. Then the SD is expressed in the abstract syntax as $D_{MAC} = block(\{1..7\}, \{i \mapsto e_i \mid 1 \leq i \leq 6\} \cup \{7 \mapsto f_7\}, \twoheadrightarrow_0, \emptyset)$ where $\twoheadrightarrow_0 = \{\langle i, i+1 \rangle \mid 1 \leq i \leq 6\}$ and $f_7 = opt(v = authorized, block(\{9..18\}, \{i \mapsto e_i \mid 9 \leq i \leq 18\}, \twoheadrightarrow_1, \emptyset))$ with $\twoheadrightarrow_1 = \{\langle i, i+1 \rangle \mid 9 \leq i \leq 15\} \cup \{\langle 15, 17 \rangle, \langle 17, 18 \rangle\}$. The operand of the *opt* operator is the block labelled 8.

### 2.2 Semantics

We now present a required behavior semantics for SDs generated from the new abstract syntax by generalizing that in [8] and simplifying its presentation.

**Semantic Domain.** The required behavior of an SD is a set of obligations one of which must be fulfilled by an implementation. An obligation is a set of required traces. A trace is a sequence of events, guard conditions and critical segments $(\!|\sigma|\!)$ where $\sigma$ is a sequence of events and guard conditions. A critical segment $(\!|\sigma|\!)$ protects $\sigma$ from interference. Occurring in a trace, $(\!|\sigma|\!)$ will be treated as atomic when the trace is combined with other traces through interleaving and weak sequencing. The domains of tokens and traces are respectively

$$\mathbb{Tk} = \mathbb{Evt} \cup \mathbb{Cnd} \cup (\!|(\mathbb{Evt} \cup \mathbb{Cnd})^*|\!)$$
$$\mathbb{Tr} = \mathbb{Tk}^*$$

and the domain of semantics is $\mathbb{Sem} = \wp(\wp(\mathbb{Tr}))$.

A semantic element $\mathcal{M}$ may contain redundant elements. For example, let $\mathcal{O}_1 = \{!m?m!n?n\}$, $\mathcal{O}_2 = \{!m!n?m?n\}$, $\mathcal{O}_3 = \mathcal{O}_1 \cup \mathcal{O}_2$ and $\mathcal{M} = \{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_3\}$. Then $\mathcal{O}_3$ is redundant in $\mathcal{M}$ since it is not minimal with respect to set inclusion $\subseteq$. Redundant obligations shall be disregarded when semantics of two SDs are compared with each other. An obligation may contain unnecessary decision points. Consider two required traces $c!m$ and $\neg c!m$. Then message $m$ is always sent since it is always the case that either $c$ or $\neg c$ holds. Occurring in an obligation, they represent an unnecessary decision point. Define $\multimap$ by $\mathcal{O} \cup \{\alpha c \beta, \alpha c' \beta\} \multimap \mathcal{O} \cup \{\alpha(c \vee c')\beta\}$. Then $\multimap$ is a convergent rewriting relation [2] on obligations and for a given obligation $\mathcal{O}$ there is a unique obligation denoted $\mathcal{O}^{\neg\circ}$ such that $\mathcal{O} \multimap^* \mathcal{O}^{\neg\circ}$ and $\mathcal{O}^{\neg\circ} \not\multimap$ where $\mathcal{O}^{\neg\circ} \not\multimap$ means that there is no $\mathcal{O}'$ such that $\mathcal{O}^{\neg\circ} \multimap \mathcal{O}'$. Define $\equiv$ by

$$(\mathcal{M}_1 \equiv \mathcal{M}_2) = \begin{pmatrix} \forall \mathcal{O}_1 \in \mathcal{M}_1 . \exists \mathcal{O}_2 \in \mathcal{M}_2 . (\mathcal{O}_1^{\neg\circ} \subseteq \mathcal{O}_2^{\neg\circ}) \\ \wedge \\ \forall \mathcal{O}_2 \in \mathcal{M}_2 . \exists \mathcal{O}_1 \in \mathcal{M}_1 . (\mathcal{O}_2^{\neg\circ} \subseteq \mathcal{O}_1^{\neg\circ}) \end{pmatrix}$$

Then $\equiv$ is an equivalence relation on $\mathbb{Sem}$.

**Semantic Function.** The semantic definition uses some auxiliary functions. Let $\mathbb{Tt} = (\mathbb{Tk} \times \mathbb{Lab})^*$ be the set of traces of tagged tokens. Function $tag : \mathbb{Tr} \mapsto \mathbb{Tt}$ labels each token in a trace by a given label: $tag(\epsilon, \ell) = \epsilon$ and $tag(t \cdot \sigma, \ell) = \langle t, \ell \rangle \cdot tag(\sigma, \ell)$ where $\epsilon$ is the empty trace. Function $untag : \mathbb{Tt} \mapsto \mathbb{Tr}$ does the opposite and is defined $untag(\epsilon) = \epsilon$ and $untag(\langle t, \ell \rangle \cdot \hat{\sigma}) = t \cdot untag(\hat{\sigma})$. Let $\alpha, \beta \in \mathbb{Tt}$ and $\sigma \in \mathbb{Tr}$. Define a rewriting relation $\curvearrowright$ by $\alpha(\!|\sigma|\!)\beta \curvearrowright \alpha\sigma\beta$. Then $\curvearrowright$ is convergent. Define $unwrap : \mathbb{Tr} \mapsto \mathbb{Tr}$ by $unwrap(\alpha) = \beta$ such that $\alpha \curvearrowright^* \beta$ and $\beta \not\curvearrowright$. Define $wrap : (\mathbb{Evt} \cup \mathbb{Cnd})^* \mapsto \mathbb{Tr}$ by $wrap(\sigma) = (\!|\sigma|\!)$. Functions $unwrap$, $wrap$, $tag$ and $untag$ are lifted to sets of sets as follows: $f(\mathcal{M}) = \{\{f(\omega) \mid \omega \in \mathcal{O}\} \mid \mathcal{O} \in \mathcal{M}\}$ where $f \in \{unwrap, wrap, tag, untag\}$. Function $lines : \mathbb{Tk} \mapsto \wp(\mathbb{Name})$ maps a token to the set of the lifelines associated with the token: $lines(!N(P,S,R)) = \{S\}$, $lines(?N(P,S,R)) = \{R\}$, $lines(c) = \emptyset$ and $lines((\!|\sigma|\!)) = \bigcup_{i \in dom(\sigma)} lines(\sigma(i))$. Function $lb : \mathbb{Tk} \times \mathbb{Lab} \mapsto \mathbb{Lab}$ returns the label of a tagged token: $lb(\langle t, \ell \rangle) = \ell$. Functions $st, wk : \wp(\mathbb{Lab} \times \mathbb{Lab}) \mapsto \wp(\mathbb{Tt})$ return the set of the traces of tagged tokens satisfying a strict sequencing order and the set of those satisfying a weak sequencing order respectively.

$$st(\twoheadrightarrow) = \{\hat{\sigma} \in \mathbb{Tt} \mid (lb(\hat{\sigma}[i]) \twoheadrightarrow^* lb(\hat{\sigma}[j])) \Rightarrow (i \leq j)\}$$

$$wk(\dashrightarrow) = \left\{ \hat{\sigma} \in \mathbb{Tt} \middle| \begin{array}{c} (lb(\hat{\sigma}[i]) \dashrightarrow^* lb(\hat{\sigma}[j])) \\ \wedge \\ (lines(\hat{\sigma}[i]) \cap lines(\hat{\sigma}[j]) \neq \emptyset) \Rightarrow (i \leq j) \end{array} \right\}$$

Let $\oplus$ be a binary operation on $S$. Then $\oplus^\sharp$ is a binary operation on $\wp(S)$.

$$X \ \oplus^\sharp \ Y = \{x \ \oplus \ y \mid x \in X \wedge y \in Y\}$$

For instance, $\cap^\sharp, \cup^\sharp$ and $\bullet^\sharp$ are respectively pair wise set intersection, set union and language concatenation where $\bullet$ is the language concatenation operator.

Consider parallel interleave $par(D_1, D_2)$ of two sub-SDs. Let $\mathcal{O}_1$ be an obligation of $D_1$ and $\mathcal{O}_2$ of $D_2$. Parallel interleaving produces a set of alternative obligations from $\mathcal{O}_1$ and $\mathcal{O}_2$ denoted $\mathcal{O}_1 \hat{|||} \mathcal{O}_2$.

$$\mathcal{O}_1 \hat{|||} \mathcal{O}_2 = \left\{ \mathcal{O} \left| \begin{array}{c} \forall \sigma_1 \in \mathcal{O}_1 . \forall \sigma_2 \in \mathcal{O}_2 . \exists \sigma \in \mathcal{O} . (\sigma \in \sigma_1 \mid\mid\mid \sigma_2) \\ \wedge \\ \forall \sigma \in \mathcal{O} . \exists \sigma_1 \in \mathcal{O}_1 . \exists \sigma_2 \in \mathcal{O}_2 . (\sigma \in \sigma_1 \mid\mid\mid \sigma_2) \end{array} \right. \right\}$$

where $\sigma \mid\mid\mid \eta$ is the set of all interleavings of $\sigma$ and $\eta$ [11]. The operator $|||^\flat$ is defined $\mathcal{M} |||^\flat \mathcal{N} = \bigcup_{\mathcal{O}_1 \in \mathcal{M}, \mathcal{O}_2 \in \mathcal{N}} (\mathcal{O}_1 \hat{|||} \mathcal{O}_2)$ and the operator $/\!\!\backslash^\flat$ is defined $\mathcal{M} /\!\!\backslash^\flat \mathcal{N} = untag(\{wk(\{\langle 1, 2\rangle\})\} \cap^\sharp (tag(\mathcal{M}, 1) |||^\flat tag(\mathcal{N}, 2)))$.

The semantics of an SD $D$, denoted $[\![D]\!]$, is defined as

$$\begin{aligned}
[\![e]\!] &= \{\{e\}\} \\
[\![\tau]\!] &= \{\{\epsilon\}\} \\
[\![strict(D_1, D_2)]\!] &= [\![D_1]\!] \bullet^\sharp [\![D_2]\!] \\
[\![alt(c, D_1, D_2)]\!] &= (\{\{c\}\} \bullet^\sharp [\![D_1]\!]) \cup^\sharp (\{\{\neg c\}\} \bullet^\sharp [\![D_2]\!]) \\
[\![opt(c, D)]\!] &= (\{\{c\}\} \bullet^\sharp [\![D]\!]) \cup^\sharp \{\{\neg c\}\} \\
[\![par(D_1, D_2)]\!] &= [\![D_1]\!] \ |||^\flat \ [\![D_2]\!] \\
[\![block(L, \iota, \twoheadrightarrow, \dashrightarrow)]\!] &= untag(\{st(\twoheadrightarrow)\} \cap^\sharp \{wk(\dashrightarrow)\} \cap^\sharp (|||^\flat_{\ell \in L} tag([\![\iota(\ell)]\!], \ell))) \\
[\![seq(D_1, D_2)]\!] &= [\![D_1]\!] /\!\!\backslash^\flat [\![D_2]\!] \\
[\![critical(D)]\!] &= wrap(unwrap([\![D]\!])) \\
[\![loop(c, D)]\!] &= \mu X.((\{\{c\}\} \bullet^\sharp ([\![D]\!] /\!\!\backslash^\flat X)) \cup^\sharp \{\{\neg c\}\})
\end{aligned}$$

where $\mu$ is the least fixpoint operator, $e \in \mathbb{E}\mathrm{vt}$, $c \in \mathbb{C}\mathrm{nd}$ and $D, D_1, D_2 \in \mathbb{S}\mathrm{d}$.

A context is an SD with one of its fragments replaced by a special symbol $\mathbb{x}$. For instance, $seq(\mathbb{x}, e)$ with $e \in \mathbb{E}\mathrm{vt}$ is a context. Let $D$ be an SD and $C$ a context. The embedding of $D$ into $C$, denoted $C[D]$ is the SD obtained from replacing $\mathbb{x}$ with $D$. The following proposition shows that the semantics possesses substitutivity. Substitutivity is a desirable property since it allows any fragment in an SD to be replaced with a semantically equivalent fragment.

**Proposition 1.** *Let $C$ be a context and $D_1, D_2 \in \mathbb{S}\mathrm{d}$. If $[\![D_1]\!] \equiv [\![D_2]\!]$ then $[\![C[D_1]]\!] \equiv [\![C[D_2]]\!]$.*

### 2.3 Refinement

We now define a refinement relation by generalizing that in [8]. When refining an SD, the designer may replace one message ( or event) by another knowing that the latter can simulate the former. For instance, he may add extra parameters into messages. We address this issue by parameterizing the notion of refinement by a simulation oracle that embodies the knowledge of the designer. The oracle

tells which event may simulate which other event. We say a binary relation $\mathcal{S}$ on events is invariant modulo renaming if, for all events $e_1$ and $e_2$ and renaming substitutions $\rho$, $\langle e_1, e_2\rangle \in \mathcal{S}$ implies $\langle \rho(e_1), \rho(e_2)\rangle \in \mathcal{S}$.

**Definition 1.** *A binary relation $\mathcal{S} \subseteq \mathbb{E}\mathrm{vt} \times \mathbb{E}\mathrm{vt}$ is a simulation oracle if it is reflexive, transitive and invariant modulo renaming.*

*Example 2.* The identity relation between events $\mathcal{S}_1 = \{\langle e, e\rangle \mid e \in \mathbb{E}\mathrm{vt}\}$ is an oracle. It is the smallest oracle since the reflexivity requirement on oracles ensures that it is contained in any other oracle. Let $\mathcal{S}_2 = \{\langle !N(S,R,P_1), !N(S,R,P_2)\rangle \mid P_2 \subseteq P_1\} \cup \{\langle ?N(S,R,P_1), ?N(S,R,P_2)\rangle \mid P_2 \subseteq P_1\}$. $\mathcal{S}_2$ states that $e_1$ simulates $e_2$ iff $e_1$ and $e_2$ have the same kind, name, sender and receiver, and $e_1$ has all the parameters of $e_2$. $\mathcal{S}_2$ allows the designer to add parameters to a message sent or received by a lifeline. For instance, a message returning no value in the original model can return a value in the refined model. A more restrictive oracle $\mathcal{S}_3$ can be obtained from $\mathcal{S}_2$ by changing $P_2 \subseteq P_1$ to the condition that $P_2$ is a sub-sequence of that of $P_1$.

**Trace Simulation.** The following definition of a trace simulation relation simplifies that in [8] and parametrizes it with an oracle.

**Definition 2.** *For a given oracle $\mathcal{S}$, $\ltimes_{\mathcal{S}}$ is defined inductively as follows.*

1. $e_1 \ltimes_{\mathcal{S}} e_2$ *if* $\langle e_1, e_2\rangle \in \mathcal{S}$.
2. $(\!|\alpha|\!) \ltimes_{\mathcal{S}} (\!|\gamma|\!)$ *if* $\alpha \ltimes_{\mathcal{S}} \gamma$,
3. $\alpha \ltimes_{\mathcal{S}} \gamma$ *if there is a trace* $\beta$ *such that* $\alpha \curvearrowright \beta$ *and* $\beta \ltimes_{\mathcal{S}} \gamma$.
4. $c_1\alpha \ltimes_{\mathcal{S}} c_2\gamma$ *if* $c_2 \models c_1$ *and* $\alpha \ltimes_{\mathcal{S}} \gamma$.
5. $\alpha \ltimes_{\mathcal{S}} c_2\gamma$ *if* $\alpha \ltimes_{\mathcal{S}} \gamma$.
6. $c_1 \cdot \alpha \ltimes_{\mathcal{S}} \gamma$ *if* $\alpha \ltimes_{\mathcal{S}} \gamma$ *and* $true \models c$.
7. $\epsilon \ltimes_{\mathcal{S}} \epsilon$.

The refinement relation in [8] is defined in terms of a monotonic function $\eta$ between traces and the transitive closure of $\curvearrowright$, which makes it hard to understand. In addition, it does not take into account the oracle. We say $\alpha$ refines $\beta$ under $\mathcal{S}$ when $\alpha \ltimes_{\mathcal{S}} \beta$. Intuitively, a refinement $\alpha$ of a trace $\beta$ can be obtained by oracle verified event substitution (clause 1), protecting subtraces with $(\!|\cdot|\!)$ (clause 3) and weakening guard conditions (clauses 4, 5 and 6). Clause 2 requires that a critical region be refined only by a critical region. For instance, $(\!|e_1 \cdot e_3|\!) \ltimes_{\mathcal{S}_1} c_1 \cdot (\!|e_1 \cdot c_2 \cdot e_2|\!)$ where $e_1, e_2$ and $e_3$ are events and $c_1$ and $c_2$ are guard conditions. But, $c \cdot e_1 \ltimes_{\mathcal{S}_1} e_1$ does not hold unless $true \models c$ since there is no guarantee that the constraint $c$ is satisfied. Relation $\ltimes_{\mathcal{S}}$ is reflexive and transitive since both $\mathcal{S}$ and $\models$ are reflexive and transitive.

### 2.4 Refinement Relation

Reuse of an SD may require a number of changes. These changes include (1) adding new lifelines and messages, (2) renaming lifelines and messages to avoid

name conflicts or better convey intention of the designer, (3) introducing new system variables and associated guard conditions. Those issues were not considered in [8]. We now generalize the refinement relation in [8] by taking them into account. The first issue is resolved by parameterizing the refinement relation with respect to a set $\mathcal{U}$ of unobservable events. It induces a hiding function $hide_{\mathcal{U}}$ on $\mathbb{Sd}$ such that $hide_{\mathcal{U}}(D)$ is the SD obtained from replacing all occurrences of $e$ with $\tau$ for each $e \in \mathcal{U}$. The second and the third issues are resolved by parameterizing the refinement relation with a mapping $\rho$ from $\mathbb{Name}$ to $\mathbb{Name} \cup \mathbb{Values}$. For $D_1$ to be a refinement of $D_2$ with respect to $\rho$ and $\mathcal{U}$, we need to make sure that any implementation of $\rho(hide_{\mathcal{U}}(D_1))$ is also an implementation of $D_2$. Thus, for any obligation $\mathcal{O}_1$ of $\rho(hide_{\mathcal{U}}(D_1))$, we require that there is an obligation $\mathcal{O}_2$ of $D_2$ such that each trace in $\mathcal{O}_2$ is simulated by a trace in $\mathcal{O}_2$. It is also necessary to make sure that events in $\mathcal{U}$ are not those that are used to simulate events in $D_2$ modulo $\rho$.

**Definition 3.** *Let $\rho \in (\mathbb{Name} \mapsto \mathbb{Name} \cup \mathbb{Values})$, $\mathcal{U} \subseteq \mathbb{Evt}(D_1)$, $D_1, D_2 \in \mathbb{Sd}$ and $\mathcal{S}$ a simulation oracle. We say $D_1$ refines $D_2$ with respect to $\mathcal{S}, \rho$ and $\mathcal{U}$, denoted $D_1 \sqsupseteq_{\mathcal{S},\rho,\mathcal{U}} D_2$, iff both*

1. *$\rho(\mathcal{U}) \cap \{e_1 \mid \exists e_2 \in \mathbb{Evt}(D_2).\langle e_1, e_2\rangle \in \mathcal{S}\} = \emptyset$, and*
2. *$\forall \mathcal{O}_1 \in [\![\rho(hide_{\mathcal{U}}(D_1))]\!].\exists \mathcal{O}_2 \in [\![D_2]\!].\forall t_2 \in \mathcal{O}_2^{-\circ}.\exists t_1 \in \mathcal{O}_1^{-\circ}.(t_1 \ltimes_{\mathcal{S}} t_2)$.*

*We write $D_1 \sqsupseteq_{\mathcal{S}} D_2$ if $D_1 \sqsupseteq_{\mathcal{S},\rho,\mathcal{U}} D_2$ for some $\rho$ and $\mathcal{U}$.*

E.g., $strict(D_1, D_2) \sqsupseteq_{\mathcal{S}_1,id,\emptyset} seq(D_1, D_2)$ and $seq(D_1, D_2) \sqsupseteq_{\mathcal{S}_1,id,\emptyset} par(D_1, D_2)$ where $id$ is the identity function and $\mathcal{S}_1$ is given in Example 2.

**Theorem 1.** *$\sqsupseteq_{\mathcal{S}}$ is reflexive and transitive, i.e., refinement can be done in a stepwise manner.*

## 3 SD Transformation

We now present a transformation that transforms an SD with *par*, *strict* and *seq* operators to an equivalent SD without them and eliminates those block fragments that are immediate constituents of block fragments.

Let $R \subseteq D \times D$ and $d_1, d_2 \in D$. If $d_1 R d_2$ then $d_1$ is a left neighbor of $d_2$ and $d_2$ a right neighbor of $d_1$. The set of left (resp. right) neighbors of an element $d$ is denoted $left(R, d)$ (resp. $right(R, d)$). Let $\sqsubseteq$ be a partial order on $D$ and $d \in D$. Then $d$ is a minimum (resp. maximum) element with respect to $\sqsubseteq$ if there is not any other element $e$ in $D$ such that $e \sqsubseteq d$ (resp. $d \sqsubseteq e$). The set of minimum (maximum) elements in $D$ with respect to $\sqsubseteq$ is denoted $min_D(\sqsubseteq)$ ($max_D(\sqsubseteq)$). The subscript $D$ will be omitted if it is obvious from context.

When a block fragment contains another block fragment as an immediate constituent, the inner block is dissolved by a transformation introduced below. The labels of a block fragment are like formal parameters of a procedure and can be changed without affecting the semantics of the block fragment so long as the change is consistent. Let $\eta$ be an invertible mapping on $\mathbb{Lab}$ and

$D = block(L, \iota, \twoheadrightarrow, \dashrightarrow)$. Define $D_\eta = block(L_\eta, \iota_\eta, \twoheadrightarrow_\eta, \dashrightarrow_\eta)$ where $L_\eta = \eta(L)$, $\iota_\eta = \iota \circ \eta^{-1}$, $\twoheadrightarrow_\eta = \{\langle \eta(\ell_1), \eta(\ell_2)\rangle \mid \langle \ell_1, \ell_2\rangle \in \twoheadrightarrow\}$ and $\dashrightarrow_\eta = \{\langle \eta(\ell_1), \eta(\ell_2)\rangle \mid \langle \ell_1, \ell_2\rangle \in \dashrightarrow\}$ and $\eta^{-1}$ is the inverse of $\eta$. $D$ and $D_\eta$ are semantically equivalent.

An immediate component block fragment of another block fragmemt can be dissovled if the labels in the inner block fragment are different from those in the outer block. Dissolving an inner block labelled $\ell$ promotes its immediate constituents to become immediate constituents of the outer block. The strict sequencing order $\twoheadrightarrow$ for the outer block is modified to become $\twoheadrightarrow''$ as follows. The strict sequencing relationships among old immediate constituents are preserved and those among new immediate constituents are inherited from the inner block. Each minimal element of the strict sequencing order $\twoheadrightarrow'$ for the inner block is made a right neighbour of each left neighbour of $\ell$ in $\twoheadrightarrow$; and each right neighbour of $\ell$ in $\twoheadrightarrow$ becomes a right neighbour of each maximal elements of $\twoheadrightarrow'$. Pairs in which $\ell$ occurs are removed. The weak sequencing order $\dashrightarrow$ for the outer block is modified to become $\dashrightarrow''$ in the same way.

**Definition 4 ($\Longrightarrow$).** *Let $D_1, D_2 \in \mathbb{Sd}$ and $\ell_1, \ell_2 \in \mathbb{Lab}$ and $D'$ a block fragment. Transformation $\Longrightarrow$ is defined as follows.*

- $C[par(D_1, D_2)] \Longrightarrow C[block(\{\ell_1, \ell_2\}, \{\ell_1 \mapsto D_1, \ell_2 \mapsto D_2\}, \emptyset, \emptyset)]$
- $C[seq(D_1, D_2)] \Longrightarrow C[block(\{\ell_1, \ell_2\}, \{\ell_1 \mapsto D_1, \ell_2 \mapsto D_2\}, \emptyset, \{\langle \ell_1, \ell_2\rangle\})]$.
- $C[strict(D_1, D_2)] \Longrightarrow C[block(\{\ell_1, \ell_2\}, \{\ell_1 \mapsto D_1, \ell_2 \mapsto D_2\}, \{\langle \ell_1, \ell_2\rangle\}, \emptyset)]$.
- $C[block(L \cup \{\ell\}, \iota \cup \{\ell \mapsto D'\}, \twoheadrightarrow, \dashrightarrow)] \Longrightarrow C[block(L'', \iota'', \twoheadrightarrow'', \dashrightarrow'')]$ *where $\eta$ is an invertible mapping $\eta$ on $\mathbb{Lab}$ such that $D'_\eta = block(L', \iota', \twoheadrightarrow', \dashrightarrow')$ and $L \cap L' = \emptyset$, $L'' = L \cup L'$, $\iota'' = \iota \cup \iota'$, $\twoheadrightarrow'' = (\twoheadrightarrow \cup \twoheadrightarrow' \cup left(\twoheadrightarrow, \ell) \times min(\twoheadrightarrow') \cup max(\twoheadrightarrow') \times right(\twoheadrightarrow, \ell)) \cap (L'' \times L'')$ and $\dashrightarrow'' = (\dashrightarrow \cup \dashrightarrow' \cup left(\dashrightarrow, \ell) \times min(\dashrightarrow') \cup max(\dashrightarrow') \times right(\dashrightarrow, \ell)) \cap (L'' \times L'')$.*

*Example 3.* Let $Login' = strict(D_1, D_2)$ with $D_1 = strict(seq(D_{id}, D_{pwd}), D_{chk})$, $D_2 = opt(OK = true, D_{cmd})$, $D_{id} = strict(e_1, e_2)$, $D_{pwd} = strict(e_3, e_4)$, $D_{chk} = strict(e_5, e_6)$, $D_{cmd} = strict(e_7, e_8)$. Then $D_{cmd} \Longrightarrow block(\{7, 8\}, \{7 \mapsto e_7, 8 \mapsto e_8\}, \{(7, 8)\}, \emptyset) = D'_{cmd}$ and $D_2 \Longrightarrow opt(OK = true, D'_{cmd}) = SD_a$. It can be verified that $D_1 \Longrightarrow^* block(\{1..6\}, \{i \mapsto e_i \mid 1 \le i \le 6\}, \{\langle 1, 2\rangle, \langle 2, 5\rangle, \langle 3, 4\rangle, \langle 4, 5\rangle, \langle 5, 6\rangle\}, \{\langle 1, 3\rangle, \langle 1, 4\rangle, \langle 2, 3\rangle, \langle 2, 4\rangle\})$. By two more steps, we have $Login' \Longrightarrow^* Login$ where $Login = block(\{1..6, a\}, \{i \mapsto e_i \mid 1 \le i \le 6\} \cup \{a \mapsto SD_a\}, \twoheadrightarrow_0, \emptyset)$ and $\twoheadrightarrow_0 = \{\langle 1, 2\rangle, \langle 1, 3\rangle, \langle 3, 4\rangle, \langle 2, 4\rangle, \langle 4, 5\rangle, \langle 5, 6\rangle, \langle 6, a\rangle\}$.

**Theorem 2.** *Let $D_1, D_2 \in \mathbb{Sd}$. If $D_1 \Longrightarrow D_2$ then $[D_1] \equiv [D_2]$.*

When applied to an SD, $\Longrightarrow$ replaces a fragment of the SD with a semantically equivalent fragment. It can be repeatedly applied until it is no longer applicable. By that time, the transformed SD does not contain interaction operator *par*, *seq* or *strict*. Nor does it have a block that contains other blocks as its immediate components. As a consequence, an SD refinement reasoning tool only has to deal with normalized SDs. In the sequel, we shall consider only normalized SDs.

## 4 Inference Rules

Table 1 presents an inference system for refinement verification. We now provide some explanation about the inference rules. That an event $e_1$ refines another $e_2$ is checked by the rule scheme named M1. M1 contains meta-variables $e_1$ and $e_2$ that are substituted for during inference. In general, a rule (scheme) has a number of premises and a conclusion separated by a horizontal line. Both a premise and a conclusion are of the form $\rho,\mathcal{U} \vdash H \rhd_{\mathcal{S}} I$ indicating that $H$ refines $I$ (with respect to $\mathcal{S}$, $\rho$ and $\mathcal{U}$). The condition $(\langle\rho(e_1), e_2\rangle \in \mathcal{S}) \wedge \forall e_1' \in \mathcal{U}.\langle\rho(e_1'), e_2\rangle \notin \mathcal{S}$ in M1 is called a side condition for its application. The side condition must be satisfied when meta-variables are substituted for.

**Table 1.** Inference rules where $\kappa_1 = \forall 1 \leq i,j \leq n.(((\ell_i' \twoheadrightarrow_2 \ell_j') \Rightarrow (\ell_i \twoheadrightarrow_1^* \ell_j)) \wedge ((\ell_i' \dashrightarrow_2 \ell_j') \Rightarrow ((\ell_i \dashrightarrow_1^* \ell_j) \vee (\ell_i \twoheadrightarrow_1^* \ell_j))))$ and $\kappa_2 = \forall \ell \in L.\mathcal{U} \supseteq \mathbb{Evt}(\iota(\ell))$

(M1) $\dfrac{}{\rho,\mathcal{U} \vdash e_1 \rhd_{\mathcal{S}} e_2}\ \langle\rho(e_1), e_2\rangle \in \mathcal{S} \wedge \forall e_1' \in \mathcal{U}.\langle\rho(e_1'), e_2\rangle \notin \mathcal{S}$

(B1) $\dfrac{\rho,\mathcal{U} \vdash \iota_1(\ell_1) \rhd_{\mathcal{S}} \iota_2(\ell_1') \quad \cdots \quad \rho,\mathcal{U} \vdash \iota_1(\ell_n) \rhd_{\mathcal{S}} \iota(\ell_n')}{\rho,\mathcal{U} \vdash block(\{\ell_1..\ell_n\} \cup L, \iota_1, \twoheadrightarrow_1, \dashrightarrow_1) \rhd_{\mathcal{S}} \langle block(\{\ell_1'..\ell_n'\}, \iota_2, \twoheadrightarrow_2, \dashrightarrow_2)}\ \kappa_1 \wedge \kappa_2$

(B2) $\dfrac{\rho,\mathcal{U} \vdash \iota(\ell) \rhd_{\mathcal{S}} I}{\rho,\mathcal{U} \vdash block(\{\ell\} \cup L, \iota, \twoheadrightarrow, \dashrightarrow) \rhd_{\mathcal{S}} I}\ \kappa_2$ (T1) $\dfrac{}{\rho,\mathcal{U} \vdash H \rhd_{\mathcal{S}} \tau}\ \mathcal{U} \supseteq \mathbb{Evt}(H)$

(A1) $\dfrac{\rho,\mathcal{U} \vdash H_1 \rhd_{\mathcal{S}} H_2 \qquad \rho,\mathcal{U} \vdash I_1 \rhd_{\mathcal{S}} I_2}{\rho,\mathcal{U} \vdash alt(C_1, H_1, I_1) \rhd_{\mathcal{S}} alt(C_2, H_2, I_2)}\ C_1 \Leftrightarrow C_2$

(A2) $\dfrac{\rho,\mathcal{U} \vdash H_1 \rhd_{\mathcal{S}} I_2 \qquad \rho,\mathcal{U} \vdash I_1 \rhd_{\mathcal{S}} H_2}{\rho,\mathcal{U} \vdash alt(C_1, H_1, I_1) \rhd_{\mathcal{S}} alt(C_2, H_2, I_2)}\ C_1 \Leftrightarrow \neg C_2$

(A3) $\dfrac{\rho,\mathcal{U} \vdash H \rhd_{\mathcal{S}} J}{\rho,\mathcal{U} \vdash alt(C_1, H, I) \rhd_{\mathcal{S}} opt(C_2, J)}\ (C_2 \Leftrightarrow C_1) \wedge (\mathcal{U} \supseteq \mathbb{Evt}(I))$

(A4) $\dfrac{\rho,\mathcal{U} \vdash I \rhd_{\mathcal{S}} J}{\rho,\mathcal{U} \vdash alt(C_1, H, I) \rhd_{\mathcal{S}} opt(C_2, J)}\ (C_2 \Leftrightarrow \neg C_1) \wedge (\mathcal{U} \supseteq \mathbb{Evt}(H))$

(A5) $\dfrac{\rho,\mathcal{U} \vdash J \rhd_{\mathcal{S}} H \qquad \rho,\mathcal{U} \vdash J \rhd_{\mathcal{S}} I}{\rho,\mathcal{U} \vdash J \rhd_{\mathcal{S}} alt(C, H, I)}$ (A6) $\dfrac{\rho,\mathcal{U} \vdash H \rhd_{\mathcal{S}} J \qquad \rho,\mathcal{U} \vdash I \rhd_{\mathcal{S}} J}{\rho,\mathcal{U} \vdash alt(C, H, I) \rhd_{\mathcal{S}} J}$

(L1) $\dfrac{\rho,\mathcal{U} \vdash H_1 \rhd_{\mathcal{S}} H_2}{\rho,\mathcal{U} \vdash loop(C_1, H_1) \rhd_{\mathcal{S}} loop(C_2, H_2)}\ C_2 \Leftrightarrow C_1$

(L2) $\dfrac{\rho,\mathcal{U} \vdash H_1 \rhd_{\mathcal{S}} H_2}{\rho,\mathcal{U} \vdash loop(C_1, H_1) \rhd_{\mathcal{S}} opt(C_2, H_2)}\ C_2 \Leftrightarrow C_1$

(O1) $\dfrac{\rho,\mathcal{U} \vdash H_1 \rhd_{\mathcal{S}} H_2}{\rho,\mathcal{U} \vdash opt(C_1, H_1) \rhd_{\mathcal{S}} opt(C_2, H_2)}\ C_2 \Leftrightarrow C_1$

(R1) $\dfrac{\rho,\mathcal{U} \vdash H \rhd_{\mathcal{S}} I}{\rho,\mathcal{U} \vdash critical(H) \rhd_{\mathcal{S}} critical(I)}$ (R2) $\dfrac{\rho,\mathcal{U} \vdash H \rhd_{\mathcal{S}} I}{\rho,\mathcal{U} \vdash critical(H) \rhd_{\mathcal{S}} I}$

Let $H_1 = block(L_1, \iota_1, \twoheadrightarrow_1, \dashrightarrow_1)$ and $H_2 = block(L_2, \iota_2, \twoheadrightarrow_2, \dashrightarrow_2)$. Then $H_1$ refines $H_2$ if there is a one-to-one mapping $\eta : L_2 \mapsto L_1$ such that (1) for each $\ell' \in L_2$, $\iota_1(\eta(\ell'))$ refines $\iota_2(\ell')$, and (2) $\twoheadrightarrow_2$ and $\dashrightarrow_2$ do not impose any sequence ordering constraint that is not imposed by $\twoheadrightarrow_1$ and $\dashrightarrow_1$, and (3) $hide_{\mathcal{U}}$ hides all events in $\iota(\ell)$ for each $\ell \in L_1 \setminus \eta(L_2)$. This is captured by rule B1 in which the mapping $\eta$ is implicit and the condition (2) is expressed as $\kappa_1$ and the condition (3) $\kappa_2$. Note that $\kappa_1$ allows $H_2$ to have more non-determinism than $H_1$. Rule B2 expresses that a block $block(L, \iota, \twoheadrightarrow, \dashrightarrow)$ refines a fragment $I$ if one of its components $H$ refines $I$ and $hide_{\mathcal{U}}$ hides all events in all other components.

Rules A1 and A2 stipulate the condition for an alternation fragment to refine another. Furthermore, an option fragment may be refined by one of the branches of an alternation fragment if the condition in the option fragment is equivalent to the condition for that branch and all events in the other branch are hidden by $hide_{\mathcal{U}}$. This is captured by rules A3 and A4. A fragment refines an alternation fragment if it refines both of its branches. This is captured by rule A5. Rule A6 states that if both branches of an alternation refines a fragment $J$ then the alternation refines $J$.

An option fragment $opt(C_1, H_1)$ refines another option fragment $opt(C_2, H_2)$ when $H_1$ refines $H_2$ and $C_2$ is equivalent to $C_1$ according to rule O1. An option fragment does not refine other kind of fragment since there is no guarantee that the condition of the option fragment is satisfied. A loop fragment $loop(C_1, H_1)$ refines another loop fragment $loop(C_2, H_2)$ when $H_1$ refines $H_2$ and $C_2$ is equivalent to $C_1$ according to rule L1. A loop $loop(C_1, H_1)$ also refines an option fragment $opt(C_1, H_2)$ when $H_1$ refines $H_2$ and $C_2$ is equivalent to $C_1$. This is captured by rule L2. The interaction operator *critical* designates that the combined fragment represents a critical region. A critical region can only be refined by a critical region. Rule R2 states that $critical(H)$ refines $I$ if $H$ refines $I$. Rule R1 stipulates that $critical(H)$ refines $critical(I)$ if $H$ refines $I$. Finally, a fragment refines $\tau$ if all of its events are hidden by $hide_{\mathcal{U}}$ according to rule T1.

The following proposition establishes the soundness of the inference rules.

**Proposition 2.** *If* $\rho, \mathcal{U} \vdash D_1 \rhd_{\mathcal{S}} D_2$ *then* $D_1 \sqsupseteq_{\mathcal{S},\rho,\mathcal{U}} D_2$.

We do not know whether the refinement inference is decidable or whether the proposed refinement inference system is complete. Both model checking of MSCs [1] and bisimulation of Triggered MSCs [10] are undecidable, which suggests that the refinement relation for SDs is likely undecidable as well, implying that any refinement inference system is likely incomplete.

## 5 Case Study: Access Control

This section illustrates via an example how the conformance of an SD to an Interaction Pattern Specification (IPS) in RBML [5] can be verified. An IPS captures the interaction behavior of a design pattern in the SD view. It is formed of

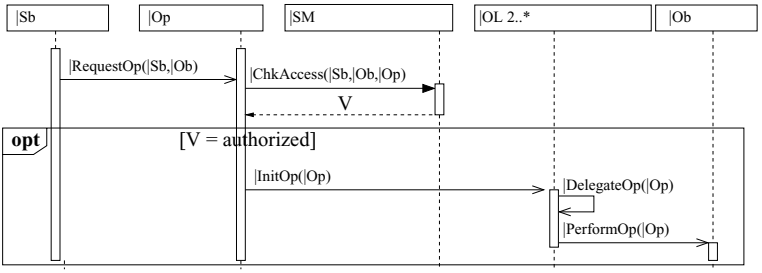
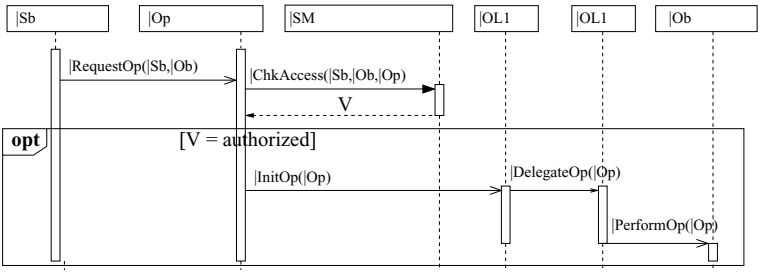



**Fig. 2.** Simplified MAC IPS

lifeline roles, message roles, UML interaction operators and metamodel operators. Every role has a realization multiplicity that constrains the number of elements playing the role. An IPS characterizes a family of SDs and a member in the family is said to conform to the IPS.

Fig. 2 shows an IPS for Mandatory Access Control (MAC) [7]. Each message or combined fragment is labelled with a letter. Roles are denoted by the symbol "|". The MAC IPS specifies the following. A subject requests an operation. The request is checked for accessibility by the *ChkAccess* operation which enforces the *restricted-* property*. The *opt* fragment specifies that if the access is authorized, it is sent to the target object through object liaisons (intermediate lifelines delegating the requests).

We now outline how the SD in Example 1 is checked to conform to the IPS. The IPS involves the lifeline role "$|OL$" which has multiplicity of 2..*. The multiplicity constraint is dealt with by repeatedly generating a pattern instance with $k$ instances of the role with $k = 2, 3, \cdots$ and verifying the SD against the instance until the conformance is verified or the instance becomes too big for the SD to conform to. The pattern instance for $k = 2$, depicted below, has role instances $|OL_1$ and $|OL_2$ that are the sender and the receiver of $|DelegateOp(Op)$.



Let $D_1$ be the SD in Example 1 and $D_2$ the above SD. Let $\mathcal{S}$ be $\mathcal{S}_2$ from Example 2, the inference system derives $\rho, \mathcal{U} \vdash D_1 \rhd_{\mathcal{S}} D_2$ where $\mathcal{U}$ contains events labeled 11, 12, 17 and 18 in Fig. 1, and

$$\rho = \left\{ \begin{array}{c} request \mapsto RequestOp, check \mapsto ChkAccess, perform \mapsto InitOp, \\ sendMsg2 \mapsto DelegateOp, sendMsg3 \mapsto PerformOp, \\ result\_checkresult\_checkResult\_ChkAccess, v \mapsto V, o \mapsto Sb, op \mapsto Op, \\ r \mapsto Ob, sl \mapsto SM, s \mapsto OL1, d \mapsto OL2 \end{array} \right\}$$

Since $D_2$ is an instance of the IPS, we conclude that $D_1$ conforms to the IPS.

## 6 Conclusion and Future Work

Refinement of SDs is an important issue in software development processes. In this paper, we have defined a formal refinement relation on SDs based on a semantics for required behavior of SDs, presented an algorithm in the form of inference rules for refinement inference and verification. A transformation has been presented that puts an SD into a normal form to simplify design and implementation of the algorithm.

A future work will be to extend the semantics, the transformations and the refinement relation to include the interaction operator *neg*. This requires to take into account the proscribed behaviors of SDs. Another will be to integrate SD refinement inference with class diagram refinement inference [9]. A more challenging task is to study the decidability of the refinement relation and study completeness of the proposed refinement inference system should it is decidable.

**Acknowledgments.** This work is supported in part by the Korea Institute of Energy Technology Evaluation and Planning (KETEP) under the international collaborative R&D program (20118530020020).

## References

1. Alur, R., Yannakakis, M.: Model Checking of Message Sequence Charts. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 114–129. Springer, Heidelberg (1999)
2. Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press (1998)
3. Cardoso, J., Sibertin-Blanc, C.: Ordering actions in sequence diagrams of UML. In: 23rd Int. Conf. on Information Technology Interfaces, pp. 3–14 (2001)
4. Cengarle, M.V., Knapp, A.: UML 2.0 interactions: Semantics and refinement. In: 3rd Int. Wsh. Critical Systems Development with UML, pp. 85–99 (2004)
5. France, R., Kim, D., Ghosh, S., Song, E.: A UML-Based Pattern Specification Technique. IEEE Transactions on Software Engineering 30(3), 193–206 (2004)
6. Haugen, Ø., Husa, K.E., Runde, R.K., Stølen, K.: STAIRS towards formal design with sequence diagrams. Software and System Modeling 4(4), 355–367 (2005)

7. Kim, D.-K., Lu, L.: Pattern-Based Transformation Rules for Developing Interaction Models of Access Control Systems. In: Mei, H. (ed.) ICSR 2008. LNCS, vol. 5030, pp. 306–317. Springer, Heidelberg (2008)
8. Kim, D., Lu, L.: Required behavior of sequence diagrams: Semantics and refinement. In: 16th IEEE ICECCS, pp. 127–136. IEEE Computer Society (2011)
9. Kim, D., Lu, L., Zhu, Y., Kim, S.: Verification of structural pattern conformance using logic programming. Universal Computer Science 16(17), 2455–2474 (2010)
10. Sengupta, B., Cleaveland, R.: Triggered message sequence charts. IEEE Trans. Software Eng. 32(8), 587–607 (2006)
11. Störrle, H.: Semantics of Interactions in UML 2.0. In: 2003 IEEE Symposium on Human Centric Computing Languages and Environments, pp. 129–136 (2003)
12. The Object Management Group. OMG Unified Modeling Language$^{TM}$ (OMG UML), Superstructure. Version 2.4, OMG Document: ptc/2010-11-14 (2011)

# Improving Relevance of Keyword Extraction from the Web Utilizing Visual Style Information

Milan Lučanský and Marián Šimko

Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava, Ilkovičova 3, 842 16 Bratislava
lucansky.milan@gmail.com, simko@fiit.stuba.sk

**Abstract.** Information growth is faster than ever before. We need to provide advanced services facilitating information "consumption" (e.g., recommendation, personalized navigation). At least a lightweight semantics is necessary for such services. Nowadays keyword paradigm is widely used and seems to achieve satisfactory results in fields such as social bookmarking or ontology learning. In this paper we explore impact of web site visual style on relevant keywords extraction. We propose a method for relevant keywords extraction from web pages combining traditional automatic term recognition algorithms with web site's visual style processing. We particularly focus on cascade style sheets. The evaluation conducted on 200 "wild" Web documents from 12 different web sites showed that our method increases the relevance of extracted keywords.

**Keywords:** automatic term recognition, cascade style sheets, keywords extraction, web documents processing, lightweight semantics.

## 1 Introduction

Since the dawn of the World Wide Web in 1991 there is a significant leap forward in technology, spread and the way we use the Web. However, the main 'mission' is the same, to freely access and share any kind of information. Every year, millions of web sites and even more blogs [12] emerge. We need to assign descriptive metadata to web pages to facilitate their further utilization. That introduces the demand for automatic processing of vast collection of web documents. The RDF standard formerly created for embedding semantics to web page structure is rarely used[1], which makes it not as useful as we would like for a major part of the Web. It turns out that keywords are suitable representation of web content, despite that they do not reach the semantic power of more complex domain representations such as ontologies. Keywords rather constitute a lightweight semantics and they form a basis for advanced semantic representations. They are utilized in ontology engineering [4], web search, user modeling for adaptive web-based systems [2], or in social services to categorize, group and search in user generated content (e.g., bookmarks space).

[1] http://trends.builtwith.com/docinfo/RDF

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 445–456, 2013.

Due to the actual size and permanent increase in count of web documents it is impossible to process whole corpora manually. We need an automatic approach to keywords acquisition. In offline documents collections there are various approaches to automatic term recognition (ATR), e.g. [6], [9], [13]. ATR algorithms are used to get single- or multiword terms from text corpora and are frequently used in domains such as medicine or biology. ATR algorithms use statistical and probabilistic features to get relevant keywords and are widely used on plain text documents (with no internal structure). If used on web documents, they could possibly benefit from hidden semantic of HTML elements used to format and visualize text content. It has been already shown that some HTML tags flag the semantic content [7] and that this information improves ATR algorithms' accuracy [10]. In our work we consider beside web documents' representational structure the visualization of text content to be another promising source of (lightweight) semantics. Our research aims at cascade style sheets (CSS) as additional source for identification or updating relevance of relevant keywords.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 presents a method for keyword extraction utilizing web site visual style. Section 4 describes evaluation of our method. Finally, we conclude our work.

## 2 Related Work

ATR algorithms are used to retrieve keywords from documents in vast document collections. They typically consider only textual features – linguistic and/or statistical. We can divide ATR algorithms into the two groups following the measure used to rank candidate terms [8]: termhood algorithms and unithood algorithms.

Termhood algorithms try to find the degree of domain specificity of linguistic unit (candidate term) and are based on its frequency in a corpus, e.g., by introducing the probability of occurrence for every candidate term assuming that a candidate term occurs more often in a domain specific collection of documents than in the rest of a corpus (e.g., [1]). In contrast to termhood algorithms, unithood algorithms measure the strength of collocation in terms, e.g., by investigating mutual probability of occurrence for words in candidate term [3]. There are also approaches, which combine both types of measures such as C-value [6], Glossex [9] and Termex [13] algorithms.

Beside plain text content processing, approaches to keyword extraction seek to utilize additional source of information – external annotations or internal document structure. Cascading Style Sheets (CSS) was developed by World Wide Web Consortium (W3C), to visually format the content of web sites. Research conducted by Opera Software on 3.5 millions of web pages showed that 80.39 % of them use either external or internal style sheets [15]. The research also showed ten most common properties, among which seven are used to format text content (e.g., color, font-size, font-family, font-weight). We consider style sheet properties an additional source of information for keyword extraction. The text highlighted by a different color or by a bold text could refer to information that is important to reader and therefore should be extracted from the text.

The state-of-the-art approaches use ATR algorithms to extract potential keywords from different document collections. They show satisfactory results on large text corpora with domain specific content. By contrast we use ATR algorithms while processing collections of *web* documents, which have diverse length and cover much more different topics. These differences yield lower accuracy of ATR algorithms that are primarily built for textual corpora [10].

We believe an approach to keyword extraction from web pages could possibly benefit from other sources of information that the Web offers – either external such as social annotations [14], or internal, e.g., HTML tags are used to get potential keywords from web pages [10]; however, the idea of utilization of CSS in co-operation with ATR algorithms is not well explored. In our work we try to make use of emergent semantics of cascade style sheets. We see potential in utilizing such emerging semantics for improving results of ATR algorithms in the Web environment, especially in web documents containing textual content while using rich visual formatting such as blogs and news portals. Such content is presented on the Web using content management systems (CMS). Advantageously, visual style of web pages for particular CMS is often similar. The number the blogs exceeds 30 % of the Web [12], hence it is important to consider visual properties of web pages for relevant keywords extraction.

## 3 Method for Keyword Extraction

Textual content of a web page typically contains words, phrases, sentences and paragraphs, which are visually formatted. Parts of content that are emphasized against style of main textual content could possibly contain words and phrases, that author of an article wanted to be noticed by readers. Our hypothesis is that the more visually differentiated a part of text is from the rest of a document, the more relevant it is for the reader.

Our approach to keyword extraction from web pages is based on ATR algorithms and utilizes visual information stored in style sheets. Extracting relevant terms from *web* documents needs extra processing because textual content important for us resides within HTML mark up and often is mixed with parts of document, which are not relevant (e.g., navigation, advertisement, footer, ...).

Our method consists of the following steps:

1. web document preprocessing,
2. keywords extraction and visually differentiated phrases identification,
3. keywords weights update.

First we get a web document and all external style sheet files used within the document. We merge all style definitions to a single virtual style and subsequently we identify and extract main textual content. We extract keywords using an ATR algorithm from plain text and identify phrases emphasized within main textual content. Finally we update weights for keywords which were extracted by an algorithm and are also emphasized within the text. Top-$k$ keywords with the highest weights we consider to be the most relevant.

### 3.1 Web Document Preprocessing

The preprocessing consists of two parts: style sheets processing and main text content identification and extraction.

**Style Sheets Processing**
First, we download all external style sheet files linked from the head section of a document by the `<link>` tag or inserted by an import rule either in internal style declarations or at the beginning of external files. The second step is to process all style information to create a virtual style for the web document. We create the virtual style by cascading available style information according to importance and order in which they appear in a web document. The cascading order is visualized in Figure 1.

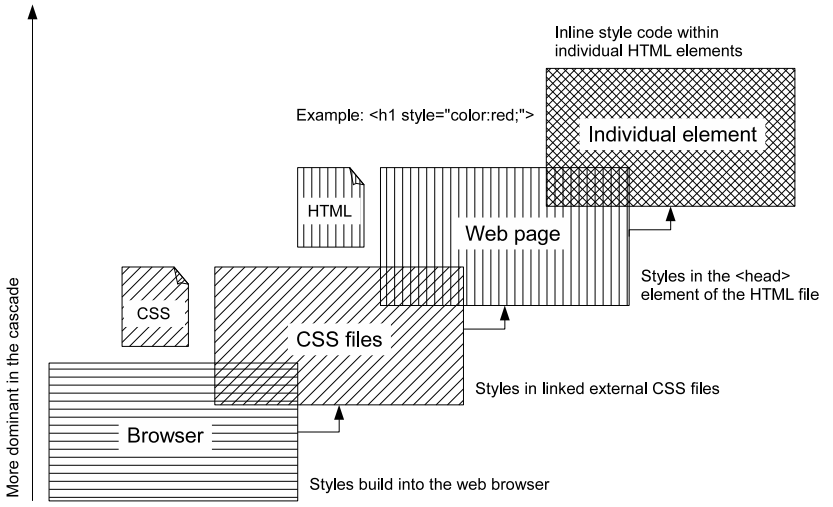



**Fig. 1.** Style sheets cascading order and priority [11]

Every browser uses a default style sheet if there is no style definition in a web document. We use the default style recommended by W3C[2]. Next step in cascading is to process the external style sheet files in an order they were linked to the document. Styles that were linked to document later override the style definition in files that were linked before. We apply the internal styles after external style sheets processing. Internal styles reside within style tag in head section of HTML document. Finally, the inline style definitions (styles defined within a style attribute of a particular HTML element) are incorporated.

Creating a virtual style is an important part of our method as we need to identify the style attributes and style values for particular HTML elements.

[2] http://www.w3.org/TR/CSS2/sample.html

**Main Textual Content Identification and Extraction**
It is common that a web document contains besides main content also various additional information such as navigation or advertisements. The most important part is main textual content. In this step we extract such content from a web document.

### 3.2 Keyword Extraction and Visually Differentiated Phrase Identification

The ATR algorithms extract keywords from plain text only, therefore we first remove HTML markup. By using ATR algorithms we extract weighted candidate keywords (words and phrases). Different algorithms assign different weights for extracted terms, based on termhood, unithood or combined approaches.

We perform visually differentiated phrase identification within the main textual content. We find HTML elements, which have different values for selected CSS attributes, extract the text from the elements and identify noun phrases. Text within elements contains prepositions or other insignificant words, the noun phrases should ensure we will acquire the most descriptive words. The selected CSS attributes that we consider are: *font-size*, *font-style*, *font-variant*, *font-weight*, *colour*, *background-colour*, *text-decoration* and *text-transform*. We refer visually differentiated phrases to as *CSS phrases*.

### 3.3 Keywords Weight Update

We utilize addition information about text formatting stored in style sheets and confer visually differentiated candidate keywords an advantage over candidate keywords which are formatted by a *default style* (i.e., the selected CSS attributes and their values that are common for majority text of article).

We adjust candidate keyword relevance weight as follows:

$$w_f(k) = w_a(k) + CssRel(k') \cdot p \tag{1}$$

where $w_f(k)$ is the final weight for candidate term $k$, $w_a(k)$ is the weight of candidate term $k$ assigned by an ATR algorithm, $CssRel(k')$ is the CSS relevance coefficient for phrase $k'$, $k'$ is the CSS phrase within main textual content and $p$ is intersection rate of candidate term $k$ and $k'$.

Sometimes the candidate term and CSS phrase are equal, but in many situations they are not. There is an intersection; some words from $k$ are also present in the $k'$. For example, let $k$ be the candidate term "*job at GuruIT*" and $k'$ be the phrase "*dream job*", the intersection is in the word *job* which is common for both phrases. The intersection rate $p$ for candidate term $k$ and phrase $k'$ is computed as follows:

$$p = \frac{|k \cap k'|}{|k'|} \tag{2}$$

The value of $CssRel$ coefficient represents to what extent the CSS phrase is visually differentiated from the main textual content. It depends on two factors:

– word count,
– visibility of text – visibility factor.

The first factor is count of words within HTML element that has different formatting from main textual content. Visibility factor is derived from CSS attributes applied to a phrase and represents the degree of visual differentiation from default style. We compute $CssRel(k')$ coefficient for CSS phrase $k'$ as follows:

$$CssRel(k') = (1 + \frac{1}{\log wc}) \cdot vf(k') \tag{3}$$

where $wc$ is the word count within HTML element containing the phrase and $vf(k')$ is the visibility factor for phrase $k'$.

**Word Count.** $CssRel$ negatively correlates with the number of words within HTML element for which we compute the coefficient. It is based on an assumption that the less words the HTML element contains, the more likely it contains relevant keyword. Let us see the following example:

```
<div id='perex' style='font-style:italic'>
   A view of Berlin its cycles of destruction and renewal and the evolu-
   tion of its food scene through the lens of a one-night gastro tour.
</div>
<div id='content'>
   <h3 style='font-weight:bold'> Berlin Food: The Cuisine Scene </h3>
   This snack cafe catches my eye ...
</div>
```

The first *div* element contains 25 words, where is quite a lot of words which are not considered as keywords, while *h3* element contains only five words, which are considered more relevant to the topic of blog post, which is Berlin food scene.

**Visibility Factor.** Visibility factor represents a degree of visual differentiation of the phrase from the main text. It is computed as a combination of selected CSS attributes applied to the phrase.

$$vf(k') = 1 + fsz + fs + fw + fv + td + tt + lum \tag{4}$$

where $vf(k')$ is visibility factor for CSS phrase $k'$, $fsz$, $fs$, $fw$, $fv$, $td$, $tt$ are coefficients for font-size, font-style, font-weight, font-variant, text-decoration and text-transform attributes, respectively. $lum$ is coefficient for luminosity, which denotes the contrast between the colour of text and background.

Coefficients $fsz$ and $lum$ are computed proportionally with respect to formatting used in main textual content.

$$fsz(k') = \frac{fs - \min fs}{\Delta fs} \cdot a \tag{5}$$

where $fsz(k')$ is the coefficient for font-size of CSS phrase $k'$, $fs$ is the font-size of CSS phrase, $\min fs$ is the minimal font-size in main textual content and $\Delta fs$ is the difference between the maximal and minimal font-size in main textual content and $a$ is the weight of font-size attribute according to a weighting scheme.

$$lum(k') = \frac{li - \min li}{\Delta li} \cdot b \tag{6}$$

where $lum(k')$ is the coefficient for luminosity of CSS phrase $k'$, $li$ is the luminosity of CSS phrase, $\min li$ is the minimal luminosity found in the main textual content, $\Delta li$ is the difference between the maximal and minimal luminosity and $b$ is the weight assigned to luminosity according to a weighting scheme.

Coefficients $fs$, $fw$, $fv$, $td$, $tt$, $a$ and $b$ represent a weighting scheme. A weighting scheme can differ for various sets of web resources (web sites). Weights of coefficients reflect visual formatting specifics for a given web site or data set. It is possible to set up each coefficient separately. The weighting scheme is designed to be easily changed according to knowledge of style scheme for a specific web site and employed ATR algorithms. Let us take for example the BBC web site (bbc.co.uk). It uses bold text in perex and in headings. Those parts tend to contain descriptive and important terms. It is easy to increase the partial weight of font-weight property in visibility factor, which results in a higher weight of the extracted terms present in text that is formatted by bold font.

Since our aim was to devise method performing on a general corpus covering many different web sites with different formatting, the weighting scheme is generic and it is not optimized for a specific web site. The potential of the method lies in its configurability for different web content types. Significant portion of the Web are blogs and news portals, which are typically presented within content management systems (CMSs). Web pages in a CMS often have equal or similar visual styles, hence one method setup may perform correctly for hundreds or thousands of web pages.

Our approach is based on re-ordering extracted candidate keywords by changing the weight of keywords which are visually differentiated in text. The idea is to increase weight to keywords which author of text wanted the readers to notice. Such keywords are more relevant, therefore better for document representation. A result is that a final set of extracted keywords may contain keywords not considered relevant when using ATR algorithm only.

## 4 Evaluation

When evaluating the proposed method our aim was (i) to verify method performance, i.e., if the proposed method is able to improve the relevance of extracted keywords, and (ii) to evaluate the most important CSS properties. We implemented the method in Java language using JATE toolkit v1.0[3] for ATR keywords extraction, the Java port of Readability algorithm[4] for identification

[3] JATE toolkit http://code.google.com/p/jatetoolkit/

[4] Java Readability https://github.com/basis-technology-corp/Java-readability

of main textual content, OpenNLP toolkit[5] for CSS phrase extraction and CSS Parser library[6] for cascade styles processing.

We performed evaluation of the proposed method by conducting a user experiment. Our goal was to assess accuracy of our method and compare relevance of extracted keywords with baseline ATR algorithms.

The data set used for evaluation consists of 200 web pages from 12 different web sites covering various topics from everyday life, e.g., politics, economy, sport, cooking, business, music or celebrities. The sites could be divided into two main groups: blogs and news portals. We selected the sites to the data set by a particular CMS they use. We incorporated the most popular CMSs to date and divided the web pages as follows: there were 106 web pages for Wordpress, 29 for Drupal and 5 for Joomla. We also involved a set of pages with no or unknown CMS consisting of 60 web pages.

We extracted the plain text from each web page and applied ATR algorithms to acquire list of ranked terms. We chose the top 10 % of ranked terms for each web page. Then we processed web page content and extracted visually differentiated CSS phrases. We used our method and changed weights of original terms if the terms matched with emphasized CSS phrases. Finally we got the original set of terms with changed weights and we chose the top 10 % of terms and assigned them to each web page.

We involved four ATR algorithms for relevant term extraction: Weirdness (termhood based); Cvalue, GlossEx, TermEx (termhood, unithood combination). For each algorithm we chose 10 % of best rated terms and assigned them to web page. Used ATR algorithms assign to terms relevance weight values from an interval $\sim \langle 0; 20 \rangle$. Our intent was to set $CssRel$ coefficient values, which were able to effectively change the order of extracted terms. We chose coefficient values by analysing a randomly selected web site set with respect to relevance weighs of ATR algorithms to 0.2 for $fw$, $fv$, $tt$ and $td$ coefficients. The values of coefficients $a$ in (5) and coefficient $b$ in (6) we set to 0.5.

To assess accuracy of the method, we let a group of human judges rate the relevance of extracted keywords. The web page and extracted terms were displayed to raters who had to decide, which term is relevant to the content of the web page and which is not. The judges was presented a merged set of relevant keywords – the original 10 % of the most relevant terms together with 10 % the most relevant terms acquired by using our method. The judges did not know an origin of terms. We developed the web application to enable rating. The judges had only two options (relevant, not relevant); however, they were not forced to asses the every term. If they did not answer any term, after submitting the form a dialog box appeared and notified the judge, that (s)he did not rate all the terms. If (s)he chose to send the form anyway, we did not force her/him to rate terms. The experiment lasted until each web page had been rated by at least three judges. A total number of 55 judges participated on experiment, 602 web pages were involved and 21,394 terms were assessed.

[5] OpenNLP toolkit `http://opennlp.apache.org/`

[6] Cascade style sheet parser `http://cssparser.sourceforge.net/`

We computed Fleiss' kappa to determine a degree of an agreement of judges upon keyword relevance [5]. We obtained $\kappa \sim 0.502$ denoting moderate agreement among judges. Note that in contrast to similar measures the Fleiss' kappa is pruned of agreement, which judges would achieve by chance. We consider this value very reasonable. Since the data set was not related to a specific domain and therefore the judges were not domain experts, some judges' knowledge of terms and concepts from different topics was not perfect.

We considered a term relevant if at least two judges marked it as relevant. The final relevance for particular algorithm and web page we computed as follows:

$$rel_a(d) = \frac{cnt_{a,p}}{cnt_{a,d}} \tag{7}$$

where $rel_a(d)$ is relevance of algorithm $a$ for web document $d$, $cnt_{a,p}$ is the number of terms extracted by algorithm $a$ from web document $d$ and marked as relevant, $cnt_{a,d}$ is the number of all terms extracted by algorithm $a$ from web document $d$. In fact, relevance is similar to precision measure known from information retrieval.

The final average relevance for each algorithm we computed as follows:

$$rel_a^{avg} = \frac{\sum_{i=1}^{|D|} rel_a(d_i)}{|D|} \tag{8}$$

where $rel_a^{avg}$ is the average relevance for algorithm $a$, $D$ is the set of web documents, $rel_a(d_i)$ is the relevance achieved by algorithm $a$ for document $d_i$.

We computed the average relevance for each ATR algorithm. The results are presented in Table 1. Our method improved the relevance of all used ATR algorithms. We can divide improvement into two groups. The highest average improvement was achieved by Weirdness 9.62% along with GlossEx 9.34%; the Cvalue and TermEx achieved improvement 2.46% and 2.73% respectively.

We see the reason of this distribution in the approach to term extraction. The Weirdness is a termhood-based algorithm taking into account domain specificity of terms. It works well on scientific corpora (e.g., biochemical and medicine documents), where it can find terms, which do not appear frequently in background corpus. We assume the Weirdness algorithm was not as appropriate for our general data set as other algorithms, because the data set did not contain as many specific terms as the scientific corpora do. The GlossEx algorithm is

**Table 1.** Average relevance of ATR algorithms and its improvement

| Algorithm ($a$) | Relevance $rel_a^{avg}$ [%] | | Relevance improvement $\Delta$ |
|---|---|---|---|
| | Base ATR alg. | Improved ATR alg. | |
| Cvalue | 73.22 | 75.67 | 2.46 |
| GlossEx | 50.87 | 60.22 | 9.34 |
| TermEx | 72.30 | 75.03 | 2.73 |
| Weirdness | 59.61 | 69.23 | 9.62 |

both a termhood- and unithood-based approach, which includes measuring the strength of terms collocation (if the term appear as a part of a longer term). The termhood component of GlossEx algorithm is based on the Weirdness algorithm, what reflects into similar improvement of both GlossEx and Weirdness.

Cvalue and TermEx both consist of termhood and unithood measures; however none of the algorithms uses as termhood component the Weirdness algorithm. We consider the both algorithms better for general corpus processing and therefore the average improvement ranges from 2.46% to 2.73%.

In order to determine statistical significance of the results, we performed Student's t-test. For each pair of compared algorithms (baseline ATR, improved ATR) we obtained statistical significance at $p < 0.05$ for Cvalue and Termex and $p < 0.001$ for Glossex and Weirdness.

We also evaluated the impact of the method on Content Management System (CMS) used. The results in Table 2 suggest that relevance improvement does not depend on a particular CMS system used. Despite this finding we confirm our assumption that keywords extraction from web sites using rich text formatting will be improved more than from web sites using weak or no formatting within main textual content. The first two blogs use anchors, bold, italic and different colored text in the content and achieved the highest improvement among the web sites. The exception is Joomla-based wrestling web site, which relevance worsen, despite of quite rich text formatting. The reason we see in the content of web page; not many people in our country are interested in wrestling, therefore the judges labelled some domain specific terms and wrestlers names as not relevant.

**Table 2.** Average relevance improvement among web sites in data set

| Content management system | Web site | Relevance improvement |
|---|---|---|
| Wordpress | http://blog.ups.com | 16.78 % |
| Wordpress | http://dghoang.com | 15.25 % |
| N/A | http://www.bbc.co.uk | 15.21 % |
| Drupal | http://www.cargoh.com | 14.95 % |
| N/A | http://www.csmonitor.com | 10.88 % |
| Wordpress | http://brighterlife.ca | 9.50 % |
| Wordpress | http://saltandserenity.com | 7.38 % |
| Wordpress | http://thepatternedplate.wordpress.com | 6.80 % |
| Wordpress | http://techcrunch.com | 5.94 % |
| Drupal | http://www.mtv.co.uk | 5.18 % |
| Drupal | http://www.fastcompany.com | -0.36 % |
| Joomla | http://www.impactwrestling.com | -8.70 % |

In addition to evaluation of method performance, we also assessed an influence of particular CSS attributes. During the evaluation the judges labelled terms which are relevant to the web page content. We explored, which CSS phrases helped to increase the weight of terms that were labelled as relevant.

Our goal was to discover which property helps the most in instances classification, i.e., which property has the highest information gain. The properties of CSS phrases was converted into instances, where each instance had following features: the web site from where it come from, font-style, font-variant, font-weight, text-decoration, text-transform, font-size and luminosity values and label whether the phrase helped increase relevance of some extracted terms or not. We employed well known *information gain* measure. The top three properties were font-size (0.0645), luminosity (0.0379) and font-style (0.0092). Those three attributes contribute the most while deciding if the phrase is relevant or not.

## 5 Conclusions

In this paper we presented the method for keyword extraction from web sites combining state-of-the-art ATR algorithms and the semantic potential of web sites' visual style realized by cascade style sheets. Our method represents a contribution to ATR algorithms by utilizing the additional information source.

The method was experimentally evaluated on data set of 200 web pages from the "wild" Web. The data set was general, consisting of web documents from 12 different web sites. The terms extracted from web pages were labelled as relevant or non-relevant by 55 judges. Generally, the method performed better when used on web pages with rich visual formatting. It is applicable only for English texts; however any language can be processed, if the ATR algorithms support it.

The results showed that in the used general corpus we improved average relevance of each ATR algorithm. Better improvement was achieved for algorithms with more significant termhood-based component. The highest improvement was achieved for Weirdness (9.62%) and GlossEx (9.34%). We can conclude that our method is more suitable to combine with termhood-based ATR algorithms, which does not perform as well as ATR algorithms utilizing unithood approach. We observed that intensity of different text formatting usage within main textual content influences the performance of our method rather than particular used content management system. We identified the most useful CSS attributes affecting the relevance of keywords: font-size, luminosity and font-style.

Our method aims to improve relevance of extracted relevant keywords for general corpus web documents. The potential of our approach lies in ability to adjust the method for a particular web site and its formatting specifics by modifying values of visibility factor related to particular CSS attributes (changing the weighting scheme). We strongly believe that customizing the scheme could increase the relevance of keywords even in unithood approach that did not perform in the experiment as well as termhood approach. Our further research is focused on further investigation of weighting scheme modification including automation of weighting scheme creation with respect to various web page properties.

While the large portion of today's Web is created by blogs and news portals we see the potential of our method to be used for relevant keywords extraction. Many pages or blogs share the same visual template scheme; therefore one setup of weighting scheme could help to find relevant keywords for hundreds of thousands of web pages.

**Acknowledgment.** This work was partially supported by the grants VG1/0675/1/2011-2014, APVV-0208-10 and it is a partial result of the Research and Development Operational Program for the projects Support of Center of Excellence for Smart Technologies, Systems and Services SMART, ITMS 26240120005 and SMART II, ITMS 26240120029, co-funded by ERDF.

## References

1. Ahmad, K., Gillam, L., Tostevin, L.: University of Surrey participation in TREC 8: Weirdness indexing for logical document extrapolation and retrieval (WILDER). In: Proc. of The Eighth Text REtrieval Conference, TREC 8 (1999)
2. 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)
3. Church, K.W., Hanks, P.: Word association norms, mutual information, and lexicography. Computational Linguistics 16(1), 22–29 (1991)
4. Cimiano, P.: Ontology Learning and Population from Text: Algorithms, Evaluation and Applications, 347 p. Springer (2006)
5. Fleiss, J.L.: Measuring nominal scale agreement among many raters. Psychological Bulletin 76(5), 378–382 (1971)
6. Frantzi, K.T., Ananiadou, S., Mima, H.: Automatic recognition of multi-word terms: the C-value/NC-value method. Int. J. on Digital Libraries 3(2), 115–130 (2000)
7. Hodgson, J.: Do HTML Tags Flag Semantic Content? IEEE Internet Computing 5(1), 20–25 (2001)
8. Knoth, P., Schmidt, M., Smrž, P., Zdráhal, Z.: Towards a Framework for Comparing Automatic Term Recognition Methods. In: Znalosti 2009, pp. 83–94 (2009)
9. Kozakov, L., Park, Y., Fin, T., Drissi, Y., Doganata, Y., Cofino, T.: Glossary extraction and utilization in the information search and delivery system for IBM technical support for IBM System. IBM Systems J. 43(3), 546–563 (2004)
10. Lučanský, M., Šimko, M., Bieliková, M.: Enhancing automatic term recognition algorithms with HTML tags processing. In: Proc. of Int. Conf. on Computer Systems and Technologies, CompSysTech 2011, pp. 173–178. ACM, New York (2011)
11. Lynch, P., Horton, S.: Web Style Guide: Basic Design Principles for Creating Web Sites, 3rd edn. (2009), http://webstyleguide.com/
12. NM Incite: Buzz in the Blogosphere: Millions more bloggers and blog readers (2012), http://nmincite.com/buzz-in-the-blogosphere-millions-more-bloggers-and-blog-readers/
13. Sclano, F., Velardi, P.: TermExtractor: a Web Application to Learn the Shared Terminology of Emergent Web Communities. In: Proc. of the 3rd Int. Conf. on Interoperability for Enterprise Software and Applications, pp. 287–290 (2007)
14. Uherčík, T., Šimko, M., Bieliková, M.: Utilizing Microblogs for Web Page Relevant Term Acquisition. In: van Emde Boas, P., Italiano, G.F., Nawrocki, J., Sack, H., Groen, F.C.A. (eds.) SOFSEM 2013. LNCS, vol. 7741, pp. 457–468. Springer, Heidelberg (2013)
15. Wilson, B.: MAMA: Key findings (2008), http://dev.opera.com/articles/view/mama-key-findings/

# Utilizing Microblogs for Web Page Relevant Term Acquisition

Tomáš Uherčík, Marián Šimko, and Mária Bieliková

Slovak University of Technology in Bratislava,
Faculty of Informatics and Information Technologies,
Ilkovičova 3, 842 16 Bratislava, Slovakia
uhercik07@student.fiit.stuba.sk, {simko,bielik}@fiit.stuba.sk

**Abstract.** To allow advanced processing of information available on the Web, the web content necessitates semantic descriptions (metadata) processable by machines. Manual creation of metadata even in a lightweight form such as (web page) relevant terms is for us humans demanding and almost an impossible task, especially when considering open information space such as the Web. New approaches are devised continuously to automate the process. In the age of the Social Web an important new source of data to mine emerges – social annotations of web content. In this paper we utilize microblogs in particular. We present a method for relevant domain terms extraction for web resources based on processing of the biggest microblogging service to date – Twitter. The method leverages social characteristics of the Twitter network to consider different relevancies of Twitter posts assigned to the web resources. We evaluated the method in a user experiment while observing its performance for different types of web content.

**Keywords:** automatic term recognition, keyword extraction, user-generated content, social annotations, microblog, twitter.

## 1 Introduction

World Wide Web has become one of the most important sources of information. The amount of information on the Web is huge, so a user is often overloaded and traditional information retrieval is no longer effective. Web content necessitates semantic descriptions (also referred to as metadata) that allow advanced processing such as intelligent search and personalization. This vision of machine-processable layer of metadata is for more than a decade addressed by the Semantic Web movement [3]. However, despite the increase, "semantization" of the Web is not as spread as we would like [15]. It is due to the fact, that manual creation of metadata for huge amounts of web content is almost impossible for a human being. Hence approaches for automated creation of metadata had emerged. They differ both in source of data they process and in an expressivity of semantics they produce.

The most fundamental form of metadata is term-based representation that we refer to as relevant terms (RTs). They constitute a basis for advanced semantic representations such as ontologies (either lightweight or heavyweight) [18]. With no proper terms that are relevant for a domain, construction of a more expressive semantic representation is hardly possible.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 457–468, 2013.

Recently there has been a huge spread of data that are an intended or not intended result of social activities and interaction on the Web. User-created annotations in various forms are produced by crowds for better creating, enriching, organizing and sharing additional information often related to existing web resources. Due to their amounts, social data represent a very promising source for mining and discovery of potentially useful semantic structures. We believe that despite the fact that they contain more noise, they can be used with an advantage to *supplement* traditional approaches to semantics discovery. In our work we particularly focus on microblogs as a phenomenon of today's web.

In this paper we propose a method for relevant domain term extraction for web resources that are referenced in the microblog Twitter by utilizing social aspect of microblogging. We selected the Twitter since it is the largest microblogging network to this date and it has less conversational focus than other social networks [9]. Other social networks like Facebook are used mostly for user interactions, which would result in metadata of lower quality.

The rest of the paper is structured as follows. In Section 2 we discuss related work. In Section 3 we present a method for relevant terms extraction from Twitter posts. Section 4 describes evaluation of the method. Conclusions are drawn in Section 5.

## 2 Related Work

Approaches related to relevant terms extraction can be divided according to source of information utilized for processing. There are two basic groups of approaches in keyword extraction area – extraction directly from web page textual content (utilizing Automatic Term Recognition algorithms; ATR) and extraction from external sources.

There are two main groups of ATR algorithms: termhood and unithood algorithms. Termhood algorithms are based on term frequency, for example assuming that the term candidate will occur more often in domain specific documents than in the rest of them [1]. Unithood algorithms measure collocation strength in terms. It can be done for example by investigating mutual occurrence probability of words in term candidate [8]. Both approaches can be further extended, e.g., by processing document formatting or visual style [11]. Another method for keyword extraction from text is TextRank [13], which utilizes graph-based text representation as a source for keyword relevance calculation.

Works related to the area of rating (web) resources based on graph analysis are important to our work. Page and Brin introduced PageRank for web pages' ranking by processing topology of the Web [5]. This method was adapted to Twitter's characteristics to calculate user rankings by Tunkelang [16]. Its main idea is that user has big influence when is followed by many influential users. This recursive algorithm is executed, until it converges. Another ranking method TwitterRank [17] is used to find topic-sensitive influential twitterers. It considers topic similarity between users to compute user's influence to others. Unfortunately this algorithm is badly scalable for large amounts of tweets, which arise every day. TweetRank ranking was introduced to measure ranking of web resources referenced in Twitter posts [12].

Twitter as a social network and microblogging service is used for a lot of tasks nowadays, e.g., for topical news recommendation [14], trends detection [6] or user modeling

[10,19]. To our best knowledge, there are no approaches which acquire keywords about Web resources using Twitter posts as a source of metadata extraction similarly to our concept.

## 3 Utilizing Twitter for Relevant Domain Term Extraction

Acquiring metadata from microblogs has a lot in common with ordinary methods, which extract keywords from documents, but it has its specifics.

### 3.1 Twitter – Source of Annotations for the Web Content

We can see the structure of the microblog in the context of our approach in Figure 1. Users are interconnected by a followership relationship, i.e., they subscribe for content produced by others. Followers value content of followees, they can be influenced by them. We can compute authors' rating or popularity using this kind of a relationship. There is also a relationship between a user and posts – tweets – he or she published. It is important, because author's relevance can be a good indicator of tweet's relevance. Another relationship is present between tweets and web resources they refer to. When the web resource is referred by the tweet, it can be described by that tweet. We believe that tweet could contain useful metadata about the resources.

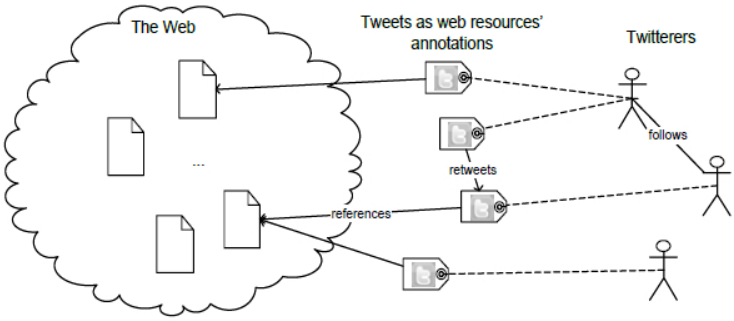



**Fig. 1.** Twitter posts as web resources' annotations. We leverage Twitter graph consisting of users and tweets to extract relevant terms for a resources the tweets refer to.

The most important cons of Twitter are presence of slang, risk of user account abused for advertisement and spam spread. Specifics of microblogs that can be used with an advantage are the hashtags (descriptive tags in tweets marked by symbol #), mentions (references to particular users marked by symbol @), retweets (re-posted tweets) and structure of implicit social network of microblog.

We see Twitter posts as annotations of web resource they refer to. Tweets' content, Twitter graph and aforementioned Twitter specifics we consider a potential source of data to mine in order to improve relevant term acquisition from web resources referenced on Twitter.

### 3.2 Method for Web Page Relevant Terms Extraction

We proposed a method for web page[1] relevant terms extraction utilizing Twitter posts. It consists of the following steps:

1. Tweet processing,
   (a) Web resource lookup,
   (b) Tweet ranking,
   (c) Relevant terms extraction,
2. Web resource content processing,
3. Combining results.

In the first step we process tweets – we look up the web resource's URL in tweets and get tweets referring to the resource, we rank the tweets and extract terms with relevance weights. Then we extract relevant terms from web resource's content. Finally, we combine the results and select top-$k$ relevant terms according to relevance weight.

**Tweet Processing**

*Web Resource Lookup*
In this step we find all occurrences of a given web resource's URL in tweets. We consider public tweets, which we access through Twitter search API. Twitter search API allows searching in tweets, which are not older than one week (approximately). This is a problem not only because we have a limited number of user annotations to exploit, but also because they are influenced by trends of that short time period. To overcome this drawback, we have to store tweets for relevant terms extraction purposes. However, the number of tweets produced daily is so big that it is very difficult to store all the Tweets. Since our method leverages social network of the Twitter, reasonable trade-off is to store only relevant tweets, e.g., tweets produced by relevant users (authorities), or tweets meeting stricter requirements related to content quality.

*Tweet Relevance Ranking*
In this step we rank tweets to distinguish more and less relevant ones. Our assumption here is the more relevant post, the more accurate terms it contains. The relevance of a tweet we compute by considering relevance of an author of the tweet. We utilize TunkRank algorithm [16], which basically is PageRank [5] adapted to Twitter authors ranking. We selected this algorithm since it is scalable, simple and has reasonable results in author ranking.

Besides TunkRank, we also consider user ranking that reflects frequency of author's posts. We slightly changed the TunkRunk computation to reflect frequency of posting tweets. Our change is based on the assumption that users who post tweets very often may write tweets with less valuable information, while authors, who tweets less often, write about the topics, they are highly interested in and they can contribute more valuable content [7]. We compute user ranking dependent on time gap between published tweets as follows:

[1] In this paper we use terms web page and web resources interchangeably.

$$TTunkRank(u) = \sum_{f \in Foll(u)} \frac{1 + \frac{p}{log(TG)} TTunkRank(f)}{|Foll(f)|}, \quad (1)$$

where $TTunkRank(u)$ is the time-aware TunkRank for user (author) $u$, $Foll(u)$ is the set of users following $u$, p is convergence constant and $TG$ is time gap median, which is defined as:

$$TG = Med(TG_0, TG_1, ..., TG_n), \quad (2)$$

where $TG_i$ is the time gap between published tweets of the author. $TG_0$ is time gap between publishing first and second tweet, $TG_1$ is time gap between second a third tweet etc. *TTunkRank* can be of value from an interval $< 0; 1 >$.

For each tweet referencing a web resource we compute tweet ranking as a maximal ranking of a user who either created or retweeted the tweet.

$$TweetRelRank(t) = max_{u \in U_t}(URank(u)), \quad (3)$$

where $TweetRelRank(t)$ is relevance ranking of tweet $t$, $URank(u)$ is user rating of user $u$. (note that we can use any known user ranking algorithm, e.g., $URank(u) = TTunkRank(u)$) and $U_t$ is a set of all users who created or retweeted the tweet $t$.

In addition to user ranking tweet ranking can be determined by considering other structural properties, e.g., number of retweets, or properties of content, e.g., formal quality of tweet (less slang) or emotional characteristics derived by natural language processing. These features have not yet been incorporated to our method since we currently focus on exploring an impact of user rating.

*Relevant Term Extraction from Tweets*

In this step we extract the relevant terms from the gathered tweet posts. In contrast with traditional text processing methods we consider information about tweet relevance. For extraction of relevant terms we can use any known automatic text recognition (ATR) algorithm or their combination.

In our implementation we use TextRank algorithm [13], but we could use another algorithm for textual ranking of words from documents (in our case tweets), or a text processing service such as AlchemyAPI or OpenCalais[2].

We compute final microblog-based relevance *MBRel* of a relevant term *rt* for web resource $d$ as follows:

$$MBRel(rt, d) = \max_{t \in T_d}(TweetRelRank(t)) * TRank(rt, T_d), \quad (4)$$

where $TweetRelRank(t)$ is relevance ranking of tweet $t$ and *TRank* is textual ranking of a relevant term *rt* in a set of tweets $T_d$ referencing a web resource $d$. Note that for computation of *TRank* the whole set of tweets is considered. We follow an approach of Wu et al., who showed that multiple posts processing results in better performance of term extraction [19].

**Relevant Term Extraction from Content of Web Resources.** In addition to microblog-based term extraction we extract terms from a web resource's content. This step is a

[2] http://www.alchemyapi.com/,http://www.opencalais.com/

part of our method since the content of the resource remains a very relevant source for relevant term extraction and our aim is to leverage microblog to enrich relevant domain terms extracted from content and provide more accurate results.

Similarly to tweets' content processing, we can employ any existing method to content processing in order to extract relevant terms from the content. We denote the relevance of relevant domain terms extracted from content as *CRel*. The detailed description of relevant domain terms extraction from a web resource content is beyond the scope of this paper.

**Combining Results.** An input of this step is set of top-$k$ relevant terms acquired from microblogs and the set of top-$l$ relevant terms obtained by the resource content processing. We combine relevant terms' relevancies from the both sources by computing combined ranking.

For every relevant term extracted from microblog we have relevance *MBRel* and for every keyword gained from textual content analysis, we have relevance *CRel*. Final relevance *Rel* of relevant term *rt* in web resource $d$ we compute as follows:

$$Rel(rt,d) = \alpha MBRel(rt) + (1-\alpha)CRel(rt,d), \tag{5}$$

where *MBRel*(*rt*,$d$) and *CRel*(*rt*,$d$) are a microblog-based relevance and content-based relevance of relevant term *rt* in a web resource $d$, respectively. $\alpha$ is weighting coefficient determining the importance of both rankings.

Finally, we select top-$m$ ranking relevant terms to be a final set of relevant terms for a given resource.

## 4 Evaluation

We evaluated our method by conducting a user experiment, where accuracy of the results provided by the method was assessed by a selected group of users – experts in particular domains selected.

### 4.1 Experiment Setup

We obtained more than 90 GB dataset from Twitter using its streaming API during 10 days. We selected the set of 60 web resources (URLs) for evaluation. The main criteria of choosing the web resources for evaluation were (i) our ability to judge if the keywords are appropriate, and (ii) the representativeness of the web resource set (i.e., we chose different types of resources – videos, pictures, news articles, etc.). We preferred to choose web resources with rather higher PageRank, because we were not intended to gain metadata about spam pages. We utilized TextRank algorithm for tweets' content processing and the Web service AlchemyAPI for relevant term extraction from web resources' content. We set the weighting coefficient $\alpha$ determining the importance of the both relevancies to balanced value 0.5.

Since our aim was to evaluate the method with respect to different types of web resources' content, we classified web resources into the following groups:

- *Comprehensive text* – news articles or other types of dense text. This type represents approximately 20% of evaluated set.
- *Brief text* – pages, which consist of different types of sparse text, page navigation etc. This type represents approximately 20% of evaluated set.
- *Product pages* – pages, which describe products. One of the typical characteristics of this product is highlighting of products' important features in text. This type represents approximately 20% of evaluated set.
- *Minimal text content pages* – this type involves videos, pictures, music, radios etc. Due to big variety of particular subtypes of this category, it represents approximately 40% of evaluated set.

We evaluated our method by an experiment in which independent experts fulfilled surveys, where they rated extracted terms as relevant or irrelevant to shown web pages. Our survey was completed by 46 active users, who produced 4400 assessments of offered terms. The assessment in this context means marking a term as relevant or irrelevant. For evaluation we considered only terms, which had three or more assessments.

We employed three measures that evaluate precision of the whole set of extracted terms, enrichment rate based on tweets and the extension proportion of relevant terms from tweets. Precision of the method in web resource $d$ we define as follows:

$$prec(d) = \frac{|RT_{rel}(d)|}{|RT(d)|}, \tag{6}$$

where $RT_{rel}(d)$ is set of all relevant terms from web resource $d$ judged really as relevant ad $RT(d)$ is set of all relevant terms extracted by our method. The second measure is the enrichment rate *enr*, which shows the rate of enrichment, how terms from Twitter enrich the whole set of web resource relevant terms obtained by our method.

$$enr(d) = \frac{|RT_T(d) \backslash RT_c(d)|}{|RT_T(d) \cup RT_T(d)|}, \tag{7}$$

where $RT_T(d)$ is the set of relevant terms for web resource $d$ obtained from microblog Twitter and a $RT_c(d)$ is the set of terms obtained from the content of the web resource. We consider enrichment rate very important because it shows, how much the terms from Twitter participate in the whole set of relevant terms – it shows the potential of user-generated data for relevant terms acquisition.

The third measure we observed shows the proportion of the relevant terms from microblog Twitter, which are not present in the web resource's content.

$$ext(d) = 1 - \frac{|RT_T(d) \cap AT(d)|}{|RT_T(d)|}, \tag{8}$$

where $RT_T(d)$ is the set of relevant terms obtained from microblog Twitter and $AT(d)$ is the set of all terms which are present in the web resource's content. This measure shows the extension proportion of metadata set by terms acquired externally, i.e., those not present in the web resource's content and relevant for the resource.

### 4.2 Results and Discussion

For all aforementioned measures we computed averaged values per web resource. Results obtained for the whole set of web resources, which were involved in the experiment, are shown in Table 1. We are particularly interested in precision of terms acquired from both content and microblog ($prec^{merged}$), precision of terms acquired from microblog only ($prec^{twitter}$), enrichment rate (*enr*) and extension proportion (*ext*).

**Table 1.** Results of experiment for the whole set of evaluated web resources

| | $prec^{merged}$ | $prec^{twitter}$ | *enr* | *ext* |
|---|---|---|---|---|
| **Comprehensive text** | 0.700 | 0.760 | 0.387 | 0.770 |
| **Brief text** | 0.590 | 0.594 | 0.429 | 0.207 |
| **Product pages** | 0.595 | 0.550 | 0.211 | 0.810 |
| **Minimal text content pages** | 0.645 | 0.685 | 0.448 | 0.224 |
| **All** | 0.639 | 0.663 | 0.380 | 0.155 |

Average precision obtained seems to be high, despite the fact that we offered users all terms excluding only those with zero relevance. Interesting and quite surprising finding was that precision of terms extracted from microblog was higher than precision of the whole set of terms merged with terms extracted from web resource's content. Another positive observation is that Twitter terms enrich the whole content with 38% rate, which means that 38% of relevant terms were present only in the set originating from Twitter. Terms extracted in duplicate by both methods were considered as obtained from content to show the improvement made by Twitter terms. To deal with the fact that terms extracted from content and terms extracted from Twitter can be similar but not equal (while representing the same thing), we measured the equality of terms with a small toleration based on *Levenshtein* distance.

**Precision.** We obtained the best precision for comprehensive text. It is not surprising because web resources of this type are mostly described in microblog posts using words that describe the content. This type of resource is also frequently shared by users of microblog, who tend to express to/comment the content of page *objectively*. Sometimes they are authors themselves and use microblog posts to promote the web resource they created while providing additional useful descriptions. The fact that content based method we used for term extraction is successful for this type is not surprising at all, because these methods have in general good results for comprehensive content.

The worst precision we obtained for product pages, which are in microblogs mentioned probably by persons, who are interested in sale of products and profiting from the sale. They can provide descriptions, which could be confusing. However, the merged precision is higher because the basic characteristics of product on the page are often appropriate. The precision comparison we can see in Figure 2.

**Enrichment Rate.** The highest enrichment rate *enr* was obtained for web resources with minimal text content, i.e., pictures, videos or similar content. The method was able to reasonably improve results for resources where traditional content-based methods are

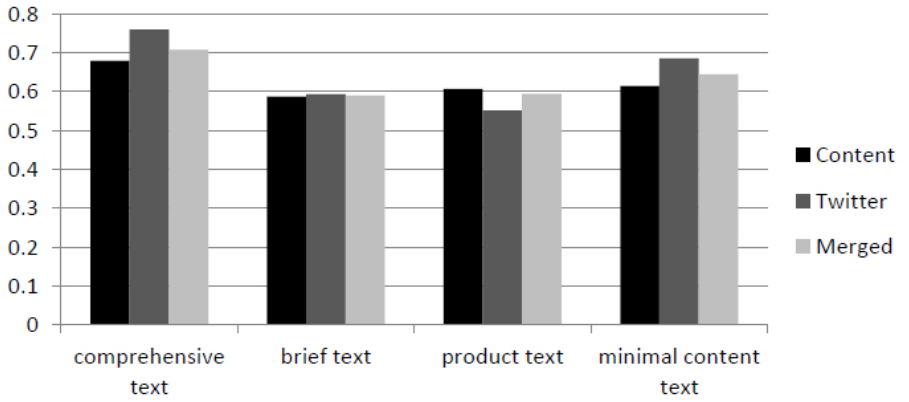



**Fig. 2.** Precision comparison for particular types of pages

not very successful. Surprising is the fact, that this rate is not significantly higher than for other types of web resources. This can be explained by our focus on web resources, which contain at least few words (e.g., a picture with at least some description). We were particularly interested in improvement comparison and by considering non textual web resources only, the enrichment rate would be obviously 100%.

The lowest enrichment rate we obtained for product pages. This is caused by the fact, that these pages are shared together with advertisement phrases and they are published in the same form by many salesmen. We obtained only narrow set of terms. Moreover, they could be even confusing. Comparison of improvement rate for individual types of pages we can see in Figure 3.

**Extension Rate.** The highest extension rate was obtained for pages with minimal text content and brief text as we supposed, because there are only few words contained within, so Twitter terms extend the merged set a lot. On the other hand, the lowest extension rate was achieved for comprehensive text containing rich text, and product pages, which often include different phrases, trying to confuse search engines. We can see the comparison of the extension rate in Figure 4.

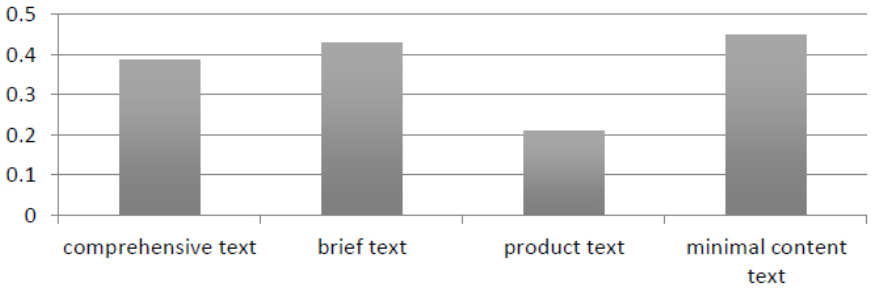



**Fig. 3.** Comparison of enrichment rate for individual types of web resources

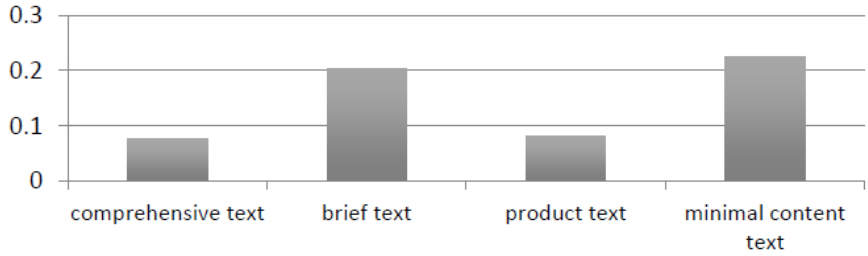



**Fig. 4.** Comparison of the extension rate for individual types of resources

Finally, Figure 5 shows the correlation of extracted terms' relevance calculated by our method and relevance assessed by human judges (we computed average judge rating for each resource). As we can see, the measures are correlated, i.e., our method succeeds in extracting relevant terms that are considered relevant also by the human judges.

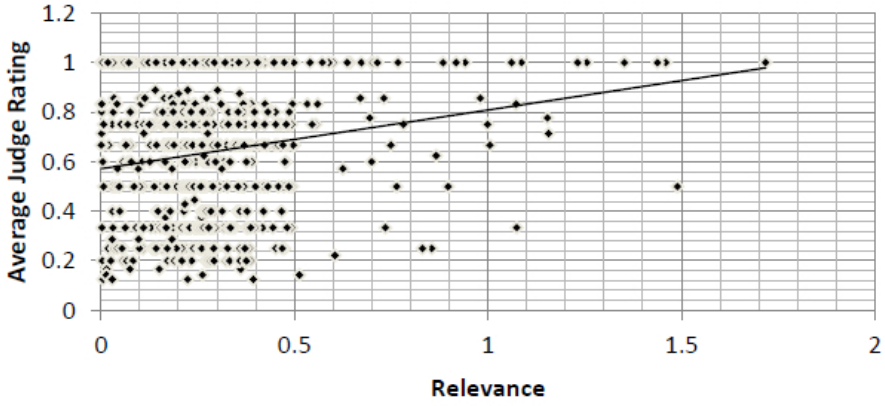



**Fig. 5.** Correlation of relevance computed by our method and ratings of judges

## 5 Discussion and Conclusions

Relevant domain terms are the cornerstone of domain and user representation for advanced information processing web-based applications [4,2]. In this paper we presented the method for web resources' relevant domain terms acquisition by utilizing microblogs created by huge masses of various users. We see microblog posts as web resources' annotation with a tremendous potential for relevant information to mine. Increasing amount of such data together with ability to select only relevant contributions makes microblog a potential source for improvement of traditional content-based approaches to relevant terms extraction.

Our method is able to enrich the set of relevant terms extracted from web resource's content, especially for resources, which do not contain a lot of self-describing text. We rely on the Twitter graph, which enables us to filter extracted terms by considering

tweets relevance derived from authors' ranking. A further enhancements of the method can be done by more complex processing of tweets, e.g. by considering their language features (hence recognizing informal and emotional posts from objective ones) or employing additional data mining while considering Twitter specifics (e.g., by clustering related tweets by hashtags or using other similarity measure).

We showed that our approach significantly enriches the relevant terms extracted from web resource's content. Depending on the type of a resource to extract relevant terms for, the results differ. The potential of the method is to treat different content type differently. Our future work covers further analysis of different resource types and incorporation of resource type identification into the method. Currently we investigate setting of parameter $\alpha$ and its automatic derivation with respect to a web resource type.

Such massive data sources as microblogs deserve a special attention when considering scalability and efficiency of processing. User rating computation represents a particular challenge, when more advanced criteria or hybrid approaches would be considered to select the most relevant users. This also calls for further research in adopting such approaches to take full advantage of parallel computing and cloud-based services.

**Acknowledgement.** This work was partially supported by the grants VG1/ 0675/1/2011 -2014, 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. Ahmad, K., Gillam, L., Tostevin, L.: University of Surrey participation in TREC 8: Weirdness indexing for logical document extrapolation and retrieval (WILDER). In: Proc. of the Eighth Text REtrieval Conference, TREC 8 (1999)
2. Barla, M.: Towards Social-based User Modeling and Personalization. Information Sciences and Technologies Bulletin of the ACM Slovakia 3(1), 52–60 (2011)
3. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American Magazine (May 2001)
4. Bieliková, M., Barla, M., Šimko, M.: Lightweight Semantics for the "Wild Web". In: White, B., Isaías, P., Santoro, F.M. (eds.) Proc. of the IADIS Int. Conf. on WWW/Internet, ICWI 2011, pp. xxv–xxxii. IADIS Press (2011)
5. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. In: Proc. of the 7th Int. Conf. on World Wide Web, pp. 107–117 (1998)
6. 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)
7. Chen, J., Nairn, R., Nelson, L., Bernstein, M., Chi, E.: Short and tweet: experiments on recommending content from information streams. In: Proc. of the 28th Int. Conf. on Human Factors in Computing Systems, pp. 1185–1194. ACM (2010)
8. Church, K.W., Hanks, P.: Word association norms, mutual information, and lexicography. In: Computational Linguistics, pp. 22–29. MIT Press (1991)
9. Java, A., Song, X., Finin, T., Tseng, B.: Why we twitter: understanding microblogging usage and communities. In: Proc. of the 9th WebKDD and 1st SNA-KDD 2007 Workshop on Web Mining and Social Network Analysis, pp. 56–65 (2007)

10. Kanta, M., Šimko, M., Bieliková, M.: Trend-Aware User Modeling with Location-Aware Trends on Twitter. In: Proc. of Semantic Media Adaptation and Personalization, SMAP 2012. IEEE Computer Society (to appear, 2012)
11. Lučanský, M., Šimko, M.: Improving Relevance of Keyword Extraction from the Web Utilizing Visual Style Information. In: van Emde Boas, P., Italiano, G.F., Nawrocki, J., Sack, H., Groen, F.C.A. (eds.) SOFSEM 2013. LNCS, vol. 7741, pp. 443–455. Springer, Heidelberg (2013)
12. Majer, T., Šimko, M.: Leveraging Microblogs for Resource Ranking. In: Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G. (eds.) SOFSEM 2012. LNCS, vol. 7147, pp. 518–529. Springer, Heidelberg (2012)
13. Mihalcea, R., Tarau, P.: TextRank: Bringing Order into Texts. In: Proc. of Conf. on Empirical Methods in Natural Language Processing, pp. 404–411. ACL (2004)
14. Phelan, O., McCarthy, K., Smyth, B.: Using twitter to recommend real-time topical news. In: Proc. of the 3rd ACM Conf. on Recommender Systems, pp. 385–388. ACM (2009)
15. Sabou, M., Gracia, J., Angeletou, S., D'Aquin, M., Motta, E.: Evaluating the Semantic Web: A Task-Based Approach. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ISWC/ASWC 2007. LNCS, vol. 4825, pp. 423–437. Springer, Heidelberg (2007)
16. Tunkelang, D.: A Twitter Analog to PageRank (2009), http://thenoisychannel.com/2009/01/13/a-twitter-analog-to-pagerank/
17. Weng, J., Lim, E., Jiang, J., He, Q.: TwitterRank: Finding Topic-sensitive Influential Twitterers. In: Proc. of the 3rd Int. Conf. on Web Search and Data Mining, pp. 261–270 (2010)
18. Wong, W., Liu, W., Bennamoun, M.: Ontology learning from text: A look back and in the future. ACM Computing Surveys (CSUR) 44(4), Article No. 20 (2012)
19. Wu, W., Zhang, B., Ostendorf, M.: Automatic generation of personalized annotation tags for twitter users. In: The 2010 Annual Conf. of the North American Chapter of the Association for Computational Linguistics, pp. 689–692. ACL (2010)

# State Coverage: An Empirical Analysis Based on a User Study

Dries Vanoverberghe*, Emma Eyckmans, and Frank Piessens

Katholieke Universiteit Leuven, Leuven, Belgium
{dries.vanoverberghe,frank.piessens}@cs.kuleuven.be

**Abstract.** State coverage is a relatively new metric to evaluate the quality of test suites. While most existing test adequacy criteria measure the degree of exploration of the code under test, state coverage estimates the strength of the assertions in the test suite. Initial experiments have shown that there is a correlation between state coverage and mutation adequacy, and that expert users can discover new faults by modifying the test suite to increase state coverage. Since the faults injected by mutation testing are relatively simple, it is not clear whether these experiment are valid in a broader setting. Furthermore, these results may not be reproducible by average users, since they usually lack full understanding of the internals of the tool.

This paper presents a user-based experiment to evaluate whether the state coverage of a test suite correlates with the number defects it discovers. While the results of the experiments fail to confirm this hypothesis, they do raises important questions. First, test suites with high state coverage should be good in finding logical software faults, but these faults occur less frequently than structural faults. Second, state coverage is not monotonic in the size of the test suite. Therefore, adding new test cases which check new properties and detect new faults can often reduce state coverage. Based on this, we suggest a number of improvements.

**Keywords:** state coverage, test adequacy metric, user study.

## 1 Introduction

In order to detect software faults, programmers often develop test sets. The question that arises when developing a test set is how the programmer will know when his test set is sufficiently elaborated to have a good chance that a large amount of faults have been detected. To make that decision, a lot of test adequacy metrics were defined. Those metrics indicate the reliability of a test set with respect to the detection of faults in the code under test. The existing metrics are divided into three classes based on their underlying approach [13]: structural metrics, fault-based metrics and error-based metrics.

Although a lot of metrics for test adequacy already exist, there seems to be a void in the range of available metrics. Currently used metrics tend to focus on measuring a certain amount of code that is executed by the test set. Both structural and error-based metrics apply this technique. On the other hand, test adequacy metrics that verify whether

* 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.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 469-480, 2013.

a test set sufficiently checks the program's behavior are rather rare [7]. Checking the behavior of programs is nevertheless important to verify that the program correctly implements the desired functionality. Fault-based metrics do focus on the behavior of the code under test, but most fault-based techniques are very expensive in terms of execution time. State coverage was originally developed to fill up this void in the range of available metrics for test adequacy. State coverage focuses on checking the behavior of the code under test like fault-based metrics, but pursues the low execution cost and low effort in use of structural metrics [7]. Due to its specific focus on behavior checking, state coverage is best used in combination with a metric that concentrates on measuring an amount of code executed by the test set [11]. This way, both the amount of executed code and the behavior of the code are monitored.

While the theory behind state coverage sounds promising, its practical merits still need to be demonstrated. Previous experiments to evaluate state coverage indicate that it is a promising metric [7,11]. However, both earlier experiments have disadvantages. Koster et. al. [7] have shown a correlation between the state coverage of test suites and their mutation adequacy score [3,5]. Unfortunately, it is unlikely that intricate logical faults can be created by using simple syntactic mutations. Therefore, the mutation adequacy score may not be a good proxy for measuring a test suites' capacity to detect faults. Vanoverberghe et. al. [11] performed a case study on an open source project. By increasing the state coverage of the test suite, they discovered several new faults. However, since the user of the case study had expert knowledge about state coverage, it is not clear whether these results can be reproduced by average users.

In this paper, we perform a two-phase experiment to compare the state coverage of a test suite with its capacity to detect faults. First, we follow the process of Vanoverberghe et. al. [11], and try to find new faults by increasing state coverage. Afterward, we manually inject more faults according to a documented distribution of common faults in software [8]. Second, we ask external users to write a test suite, and we compare the state coverage of the resulting test suites with the number of detected faults. This process avoids the the influence of pre-existing knowledge about state coverage, and it allows more intricate faults.

Unfortunately, both phases of the experiment fail to confirm that state coverage correlates with the number of detected faults. Therefore, we verify a number of potential causes for these surprising results. Most importantly, state coverage is designed to measure the strength of the properties that are checked using assertions. Therefore, test suites with high state coverage focus on logical software faults, which did not occur frequently in the experiment. Alternatively, it is surprising that the state coverage of a test suites can often be reduced by adding new test cases to the test suite. This behavior occurs when the new test cases perform many write operations, but only check a few of them using assertion. This lack of monotonicity adversely affects the correlation with the number of detected faults.

The remainder of this paper is structured as follows. Section 2 introduces the definition of state coverage and illustrates it by means of an example. Section 3 describes the design of the experiment. In Section 4, the results of this experiment are presented and possible causes for the results are discussed. Finally, Section 5 describes the conclusion.

## 2 State Coverage

In [11] state coverage is defined as the ratio of the number of state updates which are read by assertions to the total number of state updates.

For example, consider the class *Person* in Figure 1a. This class contains a field name and provides a getter, a setter and a default setter. Figure 1b shows a test case for this class. During the execution of this test case, there are three status updates to the field name: in the constructor, *setName* and *setNameDefault*. The test case uses *getName* to read the values written in the constructor and *setName* and checks them in an assertion. Consequently, the test case has $2/3$ or 66 percent state coverage.

```
class Person {
 String name;

 Person(String name) {
      this.name = name;
 }
 void setName(String name) {
     this.name = name;
 }

 void setNameDefault() {
     this.name = "John Doe";
 }
}
```

(a) The Person class

```
class TestPerson {
 @Test void test() {
  Person anne = new Person("Anne");
  assertEquals("Anne", anne.getName());
  //SC Person.ctor: 100%

  anne.setName("Anne-Maria");
  assertEquals("Anne-Maria", anne.getName());
  //SC Person.setName: 100%

  anne.setNameDefault();
  //SC Person.setNameDefault: 0%
 }
}
```

(b) The TestPerson class

**Fig. 1.** Example to illustrate state coverage

The existing definitions of state coverage [7,11] use a complex dependency algorithm to decide whether a test case reads a written value in an assertion. In this paper, we use *lightweight state coverage*, a variant which monitors the fields that are read while evaluating the expression of the assertion and remembers the writes that are responsible for it. While this is slightly less precise, it is easier to understand and more efficient.

## 3 Design of the Experiment

### 3.1 Goal

As discussed in the first chapter, the earlier experiments regarding the efficiency of state coverage [7,11] show that state coverage could be a promising metric for test adequacy. However, both experiments exhibit serious disadvantages. The main goal of the experiment described in this paper is to evaluate the efficiency of state coverage without the manifestation of these disadvantages so that the results of this experiment become more generally applicable. The experiment will therefore investigate the direct relationship between state coverage and the number of detected faults. Furthermore, non expert users will be employed to develop test sets instead of experts in the matter of state coverage.

### 3.2 Hypotheses

The experiment described in this paper evaluates two different hypotheses. The first hypothesis, hypothesis 1, states that a user will find more faults in the code under test by elevating the state coverage percentage than when state coverage is not being applied. The user will thus discover new faults by applying state coverage to an existing test set. Hypothesis 2 reads as follows: Consider a given program and different test sets that verify that program. The higher the state coverage percentage of a test set, the more faults the test set will detect in the program code when that test set is applied to the program. In other words, hypothesis 2 states that a correlation between the state coverage percentage and the number of detected faults in a program exists.

Hypothesis 1 is weaker than hypothesis 2, but can be checked more easily. Hypothesis 1 will therefore serve as a quick check to offer a first indication concerning the efficiency of state coverage. It simply states that elevating the state coverage percentage will cause more faults to be found than when state coverage is not applied. The experiment is designed so that this hypothesis can quickly be examined. The stronger hypothesis 2 will further investigate the relation between state coverage and the number of faults found in the program under test, guided by the results for hypothesis 1.

### 3.3 Structure of the Experiment

The experiment designed to evaluate the two discussed hypotheses is based on different existing experiments [1,2,4,9,10,12] to evaluate the efficiency of other test adequacy metrics. In general, these experiments have the same structure: In a chosen program, faults are introduced. For that program, different test sets are created. Each of those test sets has a certain coverage percentage. By executing the test sets and monitoring the amount of faults in the program code found by the test sets, the relation between the coverage percentage and the detection of faults in the program can be observed. The remainder of this section describes the choices that were made regarding the employed program, the injected faults and the creation of test sets.

For the program, we chose a Java application (the MOP-application) which was developed in the software design course of the third year of the bachelor in informatics at the KULeuven. We made this choice because the voluntary users involved in the experiment need to be able to write a test set for the program in a limited amount of time. Hence, the program needs to be rather small. In addition, it allows minor modifications to the program to avoid tool-related problems.

In the first phase of the experiment, we developed a test set for the MOP-application. First, the existing test set for the MOP-application was extended to reach a code coverage percentage[1] of approximately 90 percent. The faults discovered in this step were registered for later consultation. The use of code coverage is important, because state coverage does not measure the degree of exploration of a test suite. By combining code coverage and state coverage in the experiment, we ensure that both the amount of executed code and the behavior of that executed code is monitored.

In a second step of this first phase, the state coverage percentage of the developed test set was also increased to reach approximately 90 percent. To that end, assertions were

[1] Instruction coverage as measuerd by EclEmma [6].

added to the existing test set. Thus, the code coverage percentage of the test set stayed the same. The faults that were found while increasing the state coverage percentage, were recorded together with the state coverage percentage of the test set on the moment the fault was found.

In this first phase it became clear that developing a test set for the original MOP-application was too time-consuming for the voluntary users' time constraints. The original MOP-application was therefore shortened. In the rest of this text, the original MOP-application will always be referred to using the word "original". Other occurrences of the word MOP-application refer to the shortened version. After this shortening, additional faults were injected manually into the program. In total, 20 faults were present in the simplified MOP-application. Four of those faults were faults that were discovered in the first phase of the experiment. Nine faults were explicitly injected in the program code. The choice of the faults to inject was based on a documented distribution of common faults in software [8]. A last group contains the additional seven faults that users of the experiment detected while writing a test set for the program in the second phase of the experiment.

In this second phase, the users developed a test set for the MOP-application with injected faults. The users of the experiment were voluntary students in the last year of the master in computer science. Like in the first phase of the experiment, the users initially developed a test set for the MOP-application that reached a code coverage percentage of approximately 90 percent. In this first step, the users were kept in the dark about the existence and practice of state coverage. After this step, the users were informed about the existence of state coverage and the Java state coverage tool. They were asked to use that tool to increase the state coverage percentage of their developed test sets to approximately 90 percent. During this phase, all the faults that the users detected were registered together with the code and state coverage percentage on the moment the fault was found.

### 3.4 Threats to Validity

As for any other experiment, there are a number of threats to the validity of the conclusions of the experiment.

**Internal Validity.** The experiment took place in two different sessions. Therefore, there may be a discrepancy between the results of these sessions. In practice, we have not observed significant differences between both sessions.
Since the users first tested the program using code coverage, the users may have become more proficient in the programming language and the testing environment before using state coverage. However, both the programming language and the test environment were well-known to the users before the start of the experiment.

**External Validity.** In the experiment, a self-written application is used. It is possible that the chosen program is not representative enough for realistic programs. The chosen application was developed by students instead of experienced programmers. Moreover, there is a lot of variation in software applications, so a single program can never represent them all. This threat to validity is inherent to our choice for the MOP-application and will have to be kept in mind while analyzing the results.

Another threat is the result of the choice of the injected faults. The faults that were already present in the MOP-application and that were found by either us or the users of the experiment are real faults. Although the injected faults were selected by using a distribution of common software faults, there is no guarantee that this distribution is correct nor that the injected faults correctly represent the determined classes. This threat is inherent to our choices and will have to be kept in mind while analyzing the results.

The last threat to the validity of the experiment is the fact that students may not be the best users for the experiment. Students may not achieve the same results as experienced programmers. In addition, experiences state coverage users may achieve better results. On the other hand, state coverage was developed to help users develop good test sets, so even less experienced users should be able to benefit from it.

## 4 Evaluation of the Results

### 4.1 Phase 1

The results of phase one of the experiment, where we developed a test set for the original MOP-application, are limited. The goal of this phase was to perform a preliminary evaluation of hypothesis 1 by checking whether increasing the state coverage percentage would cause additional faults to be found in the original MOP-application. However, during the part of this phase where the state coverage percentage was increased, no additional faults were found. Based on these results, hypothesis 1 can not be accepted. The results of phase two will further investigate hypothesis 1 and 2 based on the data gathered from the user experiment, since the data gathered in this phase is too limited to draw any general conclusions.

Most likely, no additional faults were found because state coverage assists in finding a specific class of faults, and these faults are rather rare in the original MOP-application. State coverage was designed to measure the strength of the assertions in a test suite. Most discovered faults were structural faults, such as null pointer exceptions due to improper handling of null arguments. This kind of fault can not be detected by adding new assertions. Instead, test suites with high state coverage focus on logical faults, such as violations of an invariant or a post condition. Of the 13 discovered faults, only one could be found by increasing state coverage. However, it was already discovered while increasing code coverage.

### 4.2 Phase 2

A first goal of the experiment was to evaluate whether elevating the state coverage percentage of a test set would lead to more detected faults in the program code (hypothesis 1). In this phase also, like in phase one, none of the 13 users found an additional fault during the state coverage part of the experiment. This result suggests that the explicit augmentation of state coverage has little effect on the number of faults found. However, before such a rigorous conclusion can be drawn, a more detailed investigation is required.

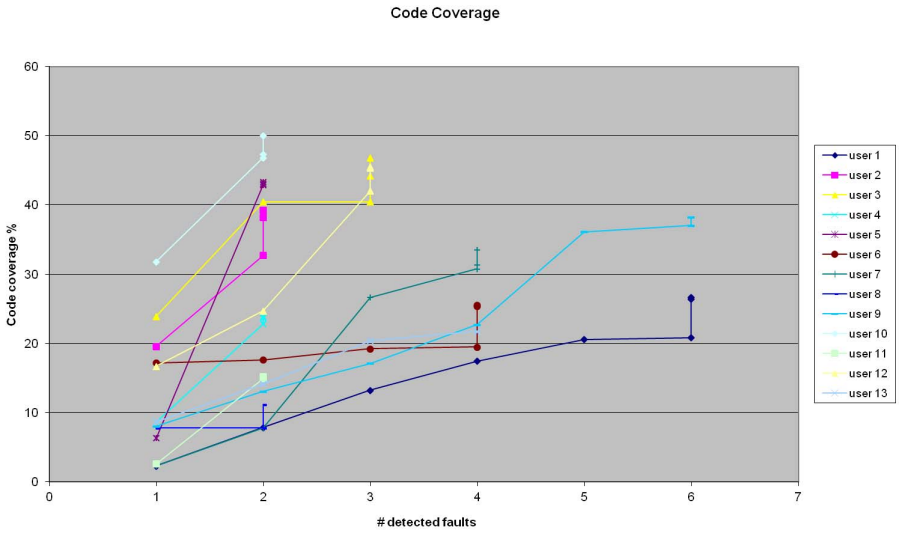



**Fig. 2.** Code coverage evolution

Before proceeding to the evaluation of hypothesis 2, we check a control hypothesis about code coverage which states that there is a positive correlation between the code coverage of a test suite and the number of faults it detects. During the experiment, users were required to gather snapshots of their test set on certain points in the development. The users were asked to create a snapshot every time they detected a fault, at the end of the code coverage part of the experiment and at the end of the state coverage part of

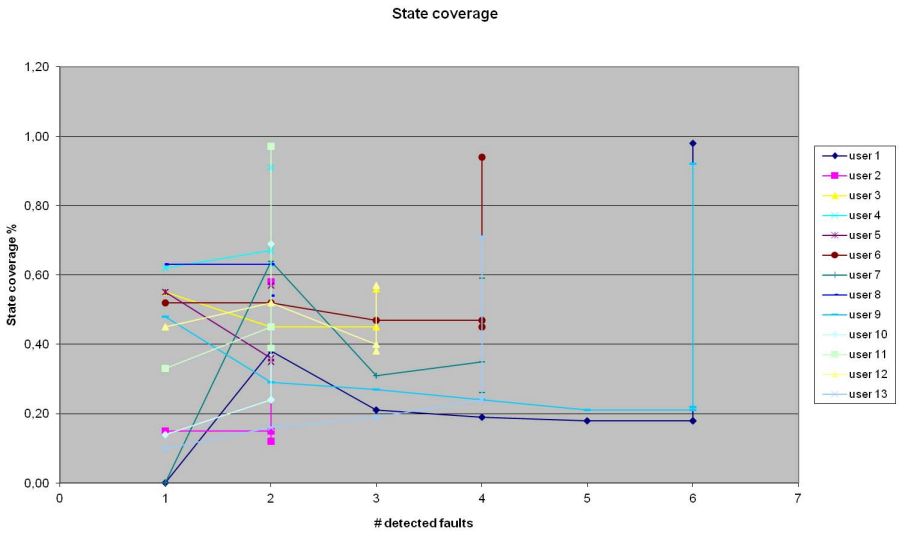



**Fig. 3.** State coverage evolution

the experiment. These snapshots are the data that is used to evaluate the hypotheses. In Figure 2, a chart is shown that displays the code coverage percentage of these snapshots on the y-axis and the number of detected faults on the x-axis. A visual inspection of the chart reveals a correlation between the code coverage percentage and the number of faults found. To confirm hypothesis 2, we need a similar correlation between state coverage and the number of detected faults.

The chart that displays the state coverage percentage of the users' snapshots on the y-axis and the number of detected faults on the x-axis is shown in Figure 3. Unfortunately, there is no correlation visible in this chart. There does not seem to be a positive relation between the state coverage percentage and the number of detected faults. Both the results for hypothesis 1 and 2 thus seem to indicate that state coverage is not an efficient test adequacy metric. In the remainder of this section, two possible causes for these unexpected results will be discussed. Afterwards state coverage will be critically analyzed and suggestions for the improvement of state coverage will be presented.

### 4.3 Potential Causes for the Results

**Frequency of Logical Faults.** As discussed before, state coverage was designed to measure the strength of the assertions in a test suite. Consequently, it will probably only correlate with the number of logical faults in the code base. Because state coverage measures the fraction of the state updates that are read in assertions, it implicitly focuses on incorrect state updates. After executing the experiment, we can conclude that these faults are rather rare in the MOP-application. Of the 20 faults in the MOP-application, only three faults are caused by an incorrect state update. Unfortunately, only one of those three faults was actually discovered by users. Therefore, there is insufficient data to be representative for all such faults.

While state coverage focuses on missing state updates, logical faults can also be caused by missing state updates. Since test suites with high state coverage usually contain more assertions, it is likely that they will also discover more missing state updates and therefore we can broaden the scope to general logical faults. Of all detected faults in the MOP-application, 35.7 percent of the total amount of detected faults can be discovered using assertions. When we only consider these faults, there is an indication that state coverage influences the detection of faults.

To investigate this, the appropriate test classes for finding a fault were divided into two groups. One group contained the test classes that were implemented by a user, but were not able to detect a certain fault in the class under test. The other group contained the test classes that were able to detect a certain fault in the class under test. For both groups, the average state coverage percentage was calculated. On average, test classes that discover a fault have a higher state coverage percentage (42 percent) than test classes that do not discover the fault (34 percent). These results indicate that, when focusing on logical faults, the correlation between state coverage and the detected number of faults may improve. Since the variance between the state coverage within one group is rather high and the number of data points is still small, the results are not significant enough to draw strong conclusions.

The most important question that remains based on this analysis is the question whether faults that are caused by incorrect state updates are as rare in real applications as they

were in the MOP-application. If so, the specific focus of state coverage on such faults is probably not justifiable.

**Monotonicity.** Another reason why state coverage may not have a correlation with the number of detected faults is that it is not monotonic in the size of the test suite. Most test adequacy criteria can only increase when new test cases are added to the test suite. Since state coverage is relative to the exploration of the test suite, exploring a new path in the program may introduce new state updates and thus reduce state coverage, even if it discovers new faults by rereading a small fraction of them in an assertion.

Consider the chart depicted in Figure 4. This chart displays the number of detected faults on the x-axis and the absolute number of covered state updates on the y-axis. The represented points are again the snapshots of the users' test sets. In this chart, there is a positive correlation between the number of detected faults and the number of covered state updates. Thus, the absolute number of covered state updates has an influence on the detection of faults after all. However, this influence is not reflected in the state coverage percentage, as shown in Figure 3.

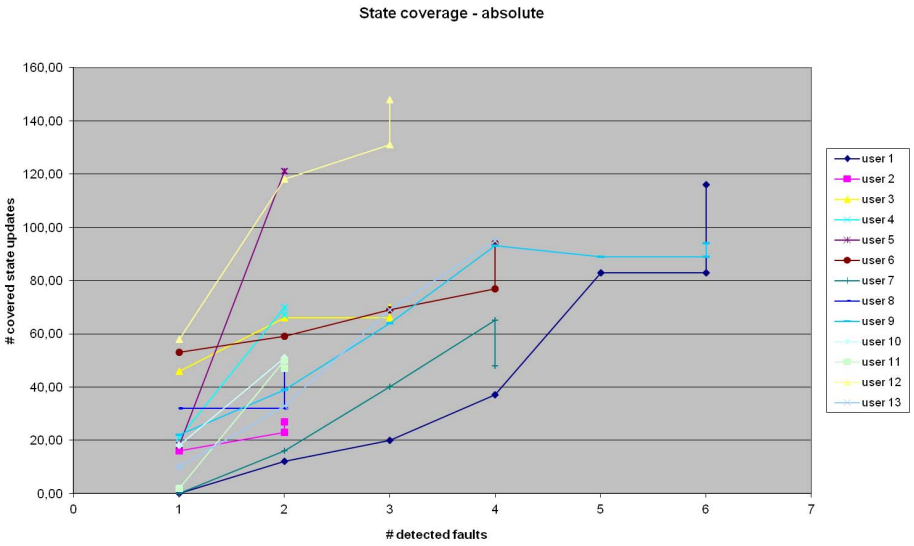



**Fig. 4.** Absolute state coverage evolution

Since the state coverage percentage can both increase and decrease when additional code is added to the test set, it may require much more effort from the tester to maintain a high state coverage ration. Therefore, high state coverage may be too strict to discover a high number of faults.

However, covering state updates in assertions seems promising since a correlation between the absolute number of covered state updates and the number of detected faults is present.

### 4.4 Analysis of State Coverage

In this section, the current form of state coverage is thoroughly analyzed. First, the benefits and drawbacks of state coverage will be discussed. Afterwards, several suggestions for the improvement of state coverage will be presented to eliminate some of the current drawbacks.

**Benefits.** The most important benefit of state coverage is the fact that the metric focuses the programmers attention on checking the behavior of the code under test. This focus is different than the focus of most existing metrics for test adequacy. Hence, state coverage is not another metric of the same kind, but it is an innovative metric with an original approach. Another benefit of state coverage is that it is easily understandable. Comprehensibility is an essential quality for a test adequacy metric. State coverage is easily explicable by means of a definition and some simple examples. The users of the experiment confirmed state coverage's understandability and had little or no problems learning to work with state coverage.

A last benefit of state coverage is the fact that it can be calculated efficiently. Unlike the mutation adequacy score, state coverage can be measured by monitoring a single execution of the test suite. Both state coverage and mutation testing focus on the behavior of the code under test, but the execution cost of state coverage is far lower than that of mutation testing.

**Drawbacks.** A first drawback of state coverage that needs to be considered is the fact that state coverage in its current form always has to be used in combination with another metric for test adequacy. State coverage only considers the code that is executed by the test suite to calculate the state coverage percentage. Therefore, state coverage will always have to be applied in combination with a metric that monitors the amount of executed code.

A second disadvantage of state coverage is the fact that in the current implementation of state coverage, a lot of useless assertions may need to be written to achieve a high state coverage percentage. On the one hand, not all state updates are equally likely to be incorrect. On the other hand, the state coverage percentage is currently calculated per test method. If a certain piece of code occurs in different test methods, state coverage will require to check the state updates for that piece of code in each test method separately, creating a bunch of duplicated assertions.

A last disadvantage arises from the disappointing results of the experiment. State coverage by design focuses on logical faults. This class is implicitly reduced to incorrect state update. These faults were rare in the MOP-application. If broader experiments suggest that logical faults do not occur frequently enough in real applications, achieving high state coverage may require too much effort with respect to the rewards that can be gained.

**Suggestions for Improvement.** A first aspect of state coverage that can be improved is the implementation. As discussed in the previous section, the current implementation calculates the state coverage percentage per test method. A possible improvement of this approach is to calculate state coverage for the whole test and thus join the results of

all test methods. In this way, a lot of repeated assertions can be avoided by registering the same state updates only once. A problem that could arise using this new approach is the fact that the memory usage might increase to unacceptable levels since all state updates will need to be kept in memory during the entire execution of the test set. In addition, this requires an intelligent way to join the state updates of different test cases.

As discussed before, the definition of state coverage currently allows the state coverage percentage to increase as well as decrease on expansion of a test set. It may be possible to create a variant of state coverage that is monotonously increasing. For example, this can be done by using a more static version of state coverage where the maximum number of state updates is fixed statically. The advantage is that the new definition will more strongly comply with the expectations of the users. A drawback of this approach is that the definition of state coverage may be less dynamic and therefore easier to achieve.

## 5 Conclusion and Future Work

State coverage is a promising metric to evaluate the strength of the assertions in a test suite. Unfortunately, the results of our experiment fail to confirm that there is a correlation between state coverage and the number of detected faults.

Based on an analysis of the results, the main open question is whether logical software faults occur frequently enough to justify the increased effort to achieve high state coverage. In addition, it seems worthwhile to explore new variants of state coverage. On the one hand, state coverage only helps to find logical software faults which are caused by incorrect state updates, as opposed to the full spectrum of logical software faults. On the other hand, a monotonic metric is more convenient to work with and may have a better correlation with the number of detected faults.

## References

1. Basili, V., Selby, R.: Comparing the effectiveness of software testing strategies. IEEE Transactions on Software Engineering SE-13(12), 1278–1296 (1987)
2. Cai, X., Lyu, M.R.: The effect of code coverage on fault detection under different testing profiles. In: Proceedings of the 1st International Workshop on Advances in Model-Based Testing, A-MOST 2005, pp. 1–7. ACM, New York (2005)
3. 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)
4. Hutchins, M., Foster, H., Goradia, T., Ostrand, T.: Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In: Proceedings of the 16th International Conference on Software Engineering, ICSE 1994, pp. 191–200. IEEE Computer Society Press, Los Alamitos (1994)
5. Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. IEEE Transactions on Software Engineering 37(5), 649–678 (2011)
6. Eclemma: Java code coverage for eclipse, http://www.eclemma.org
7. 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, pp. 541–544. ACM, New York (2007)

8. Li, Z., Tan, L., Wang, X., Lu, S., Zhou, Y., Zhai, C.: Have things changed now? An empirical study of bug characteristics in modern open source software. In: Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability, ASID 2006, pp. 25–33. ACM, New York (2006)
9. Lyu, M.R., Huang, Z., Sze, S.K.S., Cai, X.: An empirical study on testing and fault tolerance for software reliability engineering. In: Proceedings of the 14th International Symposium on Software Reliability Engineering, ISSRE 2003, pp. 119–130. IEEE Computer Society, Washington, DC (2003)
10. Mockus, A., Nagappan, N., Dinh-Trong, T.T.: Test coverage and post-verification defects: A multiple case study. In: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009, pp. 291–301. IEEE Computer Society, Washington, DC (2009)
11. Vanoverberghe, D., de Halleux, J., Tillmann, N., Piessens, F.: State Coverage: Software Validation Metrics beyond Code Coverage. In: Bieliková, M., Friedrich, G., Gottlob, G., Katzenbeisser, S., Turán, G. (eds.) SOFSEM 2012. LNCS, vol. 7147, pp. 542–553. Springer, Heidelberg (2012)
12. Wong, W., Horgan, J., London, S., Mathur, A.: Effect of test set size and block coverage on the fault detection effectiveness. In: Proceedings of the 5th International Symposium on Software Reliability Engineering, pp. 230–238 (1994)
13. Zhu, H., Hall, P.A.V., May, J.H.R.: Software unit test coverage and adequacy. ACM Comput. Surv. 29, 366–427 (1997)

# Surrogate Model for Mixed-Variables Evolutionary Optimization Based on GLM and RBF Networks

Lukáš Bajer[1,2] and Martin Holeňa[2]

[1] Faculty of Mathematics and Physics, Charles University
Malostranské nám. 25, Prague 1, Czech Republic
bajer@cs.cas.cz
[2] Institute of Computer Science, Czech Academy of Sciences
Pod Vodárenskou věží 2, Prague 8, Czech Republic
http://www2.cs.cas.cz/~martin/

**Abstract.** Approximation of costly objective functions by surrogate models is an increasingly popular method in many engineering optimization tasks. Surrogate models can substantially decrease the number of expensive experiments or simulations needed to achieve an optimal or near-optimal solution. In this paper, a novel surrogate model is presented. Compared to the most of the surrogate models reported in the literature, it has an advantage of explicitly dealing with mixed continuous and discrete variables. The model use radial basis function networks for continuous and clustering and a generalized linear model for the discrete covariates. The applicability of the model is shown on a benchmark problem, and the model's regression performance is further measured on a dataset from a real-world application.

**Keywords:** surrogate modelling, RBF networks, genetic algorithms, mixed-variables optimization, continuous and discrete variables.

## 1 Introduction

Different kinds of optimization tasks are encountered in many of todays engineering or industrial applications. Frequently, they are characterized by a high number of both continuous and discrete variables [1, 2]. Such tasks are called *mixed-variables optimization problems*, often abbreviated as MVOP, and they are similar to *mixed-variable non-linear programming* (MINLP) tasks. These tasks are common especially in situations where the value of the objective function is obtained through some measurement, experiment or simulation.

A popular approach to tasks with costly objective functions is substituting an approximating model for the empirical objective function. This approach, called surrogate modelling [3, 4], is widely used in connection with evolutionary algorithms (EAs), in spite of having been originally introduced in the area of smooth optimization.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 481–490, 2013.

Assessing some of the individuals with not necessary accurate, but much faster model brings an important benefit: a notably larger population can be evolved in parallel. Even though the original fitness function can be evaluated only on a limited number of individuals, the EA can explore a larger part of the input space.

This paper describes a particular surrogate model based on radial basis function (RBF) networks. As far as we know, the existing publications about surrogate modelling in evolutionary optimization deal with only continuous domains or combination with integer variables [5]. Outside the evolutionary area, the work of Holmström [6] is known, for example. Interesting applications can be found in some articles [7–9], or SO-MI algorithm has appeared recently [10]. However, we are aware of no surrogate models for MVOP in evolutionary context. And even outside that context, such models are rather few.

In our model, multiple RBF networks are first trained on the continuous part of the data – in this phase, discrete variables are used to focus training of the networks on the most appropriate data. Next, a generalized linear model is trained taking discrete variables as the independent and residuals of the RBF networks as the dependent variables for training.

The paper is organized as follows: in the next section, we recall the optimization task and GLM principles. Section 3 describes our approach to constructing a surrogate model and using it in optimization. Finally, Section 4 provides the results of testing on a benchmark function and real-world data.

## 2 Optimization Task and Involved Methods

For any given objective function $f : \mathbf{S} \to \mathbb{R}$, we consider the *mixed-variable optimization problem* as finding $x^*$ such that

$$f(\mathbf{x}^\star) = \max_{\mathbf{x}\in\mathbf{S}} f(\mathbf{x}). \tag{1}$$

where $\mathbf{x}^\star = (x_1^{(\mathrm{C})}, \ldots, x_n^{(\mathrm{C})}, x_1^{(\mathrm{D})}, \ldots, x_d^{(\mathrm{D})}) \in \mathbf{S}$. The problem includes $n$ continuous and $d$ discrete variables; their values belong to corresponding subspaces $\mathbf{S}^{(\mathrm{C})}$ and $\mathbf{S}^{(\mathrm{D})}$. This holds for maximization problem, minimization can be defined analogously. In addition, we suppose that the value sets $V_s(X_i^{(\mathrm{D})})$, $i = 1, \ldots, d$ of the discrete variables are finite and we do not differentiate between ordinal or nominal categorical variables – we do not require any ordering on any of the $V_s(X_i^{(\mathrm{D})})$.

### 2.1 Involved Methods

Evolutionary optimization, RBF networks and surrogate modelling were recalled already in the preceding paper [11]. For a more detailed treatment of them, the reader is referred to specialized monographs, in particular to the recent monograph [12] for global genetic optimization, [13] and [14] for the traditional linear and polynomial response-surface models, which inspired the modern nonlinear surrogate models such as Gaussian processes and artificial neural networks [3, 15, 16].

*Generalized Linear Models.* The new surrogate model presented in section 3 relies in addition on *generalized linear models* (GLM). A GLM is a natural generalization of a linear regression model [17]. It consists of three parts: (1) the *random component* – observed values $\mathbf{Y}$ following a distribution from an exponential family with mean $E(\mathbf{Y}) = \boldsymbol{\mu}$ and constant variance $\sigma^2$, (2) the *systematic component* which relates values of explanatory (input) variables $(\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^d)$ through a linear model with parameters $\beta_1, \ldots, \beta_d$ to a *linear predictor* $\boldsymbol{\eta}$

$$\boldsymbol{\eta} = \sum_{j=0}^{d} \mathbf{x}^j \beta_j, \tag{2}$$

and (3) a *link function* $g$ that connects the random and systematic components together: $\boldsymbol{\eta} = g(\boldsymbol{\mu})$.

## 3 Proposed Surrogate Model for MVOP Evolutionary Optimization

In the preceding paper [11], we described a surrogate-assisted genetic algorithm (GA), in which the proposed surrogate model obtained in the following steps:

1. the training data is clustered according to $\mathbf{S}^{(\mathrm{D})}$,
2. for each cluster, separate RBF network is trained on $\mathbf{S}^{(\mathrm{C})}$.

Let us call this model *RBF/discrete clustering*, or shortly *DSCL*.

The DSCL model does not use any relationships of the discrete input variables to the response variable. That motivated further surrogate models for MVOP, which already use such relationships. A new version of a model combining RBF networks and GLM, called a RESID model, is introduced in the following text.

### 3.1 Model Construction

Construction of the RESID model starts with building previously described DSCL model $\hat{f}_{\mathrm{DSCL}} : \mathbf{S} \to \mathbb{R}$ which is composed of several RBF networks fitted on the continuous input variables on respective clusters from clustering made using the discrete variables. If different combinations of values of the discrete variables are possible, then different RBF networks on $\mathbf{S}^{(\mathrm{C})}$ need to be constructed to reflect the differences in those combinations. In fact, in the rare situation that there are enough training data to construct separate networks for all combinations of values of discrete variables, DSCL constructs all those separate networks.

In the next step, residuals of this DSCL submodel are computed for all $N$ available training data

$$\delta_i = y_i - \hat{f}_{\mathrm{DSCL}}(\mathbf{x}_i^{(\mathrm{C})}, \mathbf{x}_i^{(\mathrm{D})}), \quad i = 1, \ldots, N. \tag{3}$$

Further, a GLM is fitted taking discrete variables as independent and these $\delta_i$'s as a dependent variable. The final value of the RESID model then sums the resulting values of DSCL and GLM models.

**FitTheModel**($s^{(1)\min}$, **D**, $e$)
**Arguments**: $s^{(1)\min}$ – min. size of clusters,
**D** – training data, $e$ – type of error estimate:
MSE, AIC, or BIC
**Steps of the procedure**:
(1) $\{C_j\}_{j=1}^{m} \leftarrow$ cluster **D** into clusters of size at least $s^{(1)\min}$ according to discrete variables
(2) **for** each cluster $C_j$, $j = 1, \ldots, m$
(3) $rbf_j \leftarrow$ parameters of the RBF network with $g_j^\star$ components fitted with data from $C_j$, and with error $e_j$
(4) $e_j \leftarrow e[j, g_j^\star]$
(5) $\delta_i \leftarrow (y_i - \hat{f}_{\mathrm{DSCL}}(\mathrm{x}_i^{(C)}, \mathrm{x}_i^{(D)}; rbf))$ for $i = 1, \ldots, N$
(10) $glm \leftarrow$ fit the GLM on $(\delta_i)_{i=1}^{N}$ choose the best link function $g$ and coding of variables via cross-validation
**Output**: $\{(rbf_j, e_j)_{j=1}^{m}, glm\}$

**Fig. 1.** Pseudo-code of the fitting procedure

A pseudo-code of our algorithm for the RESID model fitting is given in Figure 1. Important parts of the algorithm are briefly explained below.

### 3.2 Fitting the RESID Model

RBF networks enable us to use only continuous variables for their fitting. Construction of the DSCL submodel starts with clustering of the available training data according to their discrete values into several clusters in order to focus the RBF networks training on the most similar data points. Subsequently, separate networks are fitted with the data of each such a cluster using the data points' continuous variables.

Further, relation between values of the discrete input variables and the response variable is utilized via GLM which form an important part of our new RESID surrogate model.

**Parameters of the DSCL Model.** The sizes of the clusters $C_j, j = 1, \ldots, m$ have to fulfill

$$|C_j| \geq s_{\min}^{(g_{\min})} \tag{4}$$

where $s_{\min}^{(g)}$ is the minimal number of data needed for fitting an RBF network with $g$ components, and $g_{\min}$ is the required minimal number of components of each of the fitted networks (implicitly, $g_{\min} = 1$, but the user can increase this value). The value $s_{\min}^{(g_{\min})}$ depends on the employed radial basis functions, and on

the way of estimating the fitting errors $e(j,g)$ of RBF networks In particular, for Gaussian networks with $d$-dimensional continuous inputs and $e(j,g)$ estimated using $k$-fold cross-validation:

$$s_{\min}^{(g)} = \begin{cases} g\frac{k}{k-1}(d+2) & \text{if uncorrelated cont. variables with identical variance,} \\ g\frac{k}{k-1}(2d+1) & \text{if uncorrelated cont. variables with diagonal variance,} \\ g\frac{k}{k-1}\frac{d(d+3)+1}{2} & \text{else.} \end{cases} \tag{5}$$

If $e(j,g)$ are estimated without cross-validation, the coefficient $\frac{k}{k-1}$ is left out. One separate RBF network $rbf_j$ is trained on the data of each cluster $C_j$, $j = 1,\ldots,m$. The maximal number of components of each network is upper-bounded by

$$g_j^{\max} = \lfloor (\frac{k-1}{k}|C_j|)/s_{\min}^{(g_{\min})} \rfloor. \tag{6}$$

**GLM Model.** Generalized linear model is used in its continuous-response form. In the constructed GLM, the response's distribution is assumed to belong to an exponential family: either normal, gamma or inverse Gaussian distribution is supported. The most proper distribution is chosen through cross-validation. Further, the corresponding link function for each of the three distributions is used.

Before using or fitting the GLM, the discrete values must be converted to a proper representation. Since we do not assume any ordering of the discrete values, our default choice is *dummy coding* [17] which establishes one binary indicating variable $I_{ij} \in \{0,1\}$ for each nominal value from the value sets $V_s(X_i^{(\mathrm{D})})$, $i = 1,\ldots,d$, $j = 1,\ldots,|V_s(X_i^{(\mathrm{D})})|$ of the original discrete variables.

Needless to say, the number of GLM parameters for GLM fitting can grow rapidly which restricts the applicability of the dummy coding, moreover default integer representation with ordering taken from the original integer values can be beneficial to the regression quality. Therefore, as an alternative representation, integer coding can be used in situation when there are not enough data for dummy coding. As default, coding resulting in a higher regression quality in terms of cross-validation error is used.

### 3.3 Evaluation with the Surrogate Model

Once the surrogate model is built, it can be used for evaluating individuals resulting from the evolution. The parameters of the model can be summarized as $\{(rbf_j, e_j)_{j=1}^m, glm\}$. Here, $rbf_j$ are RBF/DSCL network parameters, $e_j$ are errors obtained from cross-validation, and $glm = (\beta_0,\ldots,\beta_r; \mathrm{coding}, g)$ are parameters of the GLM.

Given a new individual $(\mathbf{x}^{(\mathrm{C})}, \mathbf{x}^{(\mathrm{D})})$, evaluation with the surrogate model starts with finding the index $c$ of the cluster with the data closest to the individual's discrete values

$$c = \arg \min_{j=1,\ldots,m} \frac{1}{|\mathbf{N}_j|} \sum_{\mathbf{y} \in \mathbf{N}_j} d_{\mathrm{DISCR}}(\mathbf{x}^{(\mathrm{D})}, \mathbf{y}). \tag{7}$$

Here, $d_{\mathrm{DISCR}}$ denotes Hamming (used in testing) or Jaccard metric. Then, the RBF network with parameters $rbf_c$ corresponding to this cluster $c$ is used as a surrogate model of the original fitness by computing its return value on continuous dimensions $y_{\mathrm{DSCL}} = \hat{f}_{\mathrm{DSCL}}(\mathbf{x}^{(\mathrm{C})}; rbf_c)$. If more than one cluster is at the same distance from the individual, the RBF network with the lowest error $e$ is chosen.

The resulting value is obtained by summing with the GLM response on the discrete variables

$$\hat{y} = y_{\mathrm{DSCL}} + g^{-1}\left(\sum_{j=0}^{d} x^{j(\mathrm{D})}\beta_j\right) \tag{8}$$

where $g^{-1}$ is an inverse of the link function chosen during GLM fitting.

## 4 Implementation and Results of Testing

Our algorithms were implemented in the MATLAB environment. We utilized the Global Optimization Toolbox whose modified genetic algorithm was used as a platform for testing the model on a benchmark optimization task. Similarly, clustering method extends the standard bottom-up hierarchical cluster analysis from the Statistical Toolbox in order to guarantee clusters of a minimum specified size. Statistical Toolbox provide us with GLM fitting procedure, too, and we employ a nonlinear curve-fitting from the Optimization Toolbox for fitting RBF networks.

### 4.1 Model Fitting

Our models have been tested on two different kinds of data. The first, real-world dataset is the same as in our preceding articles [11, 18], and comes from optimization in the domain of chemical catalysis. In the latter article [18], more information about this specific real-world application is provided as well as one particular boosted surrogate model based on multilayer perceptrons. The second dataset is a set of individuals resulted from one GA optimization run of the modified Schwefel's benchmark function [19].

The first dataset comes from a real application in chemical engineering – optimization of chemical catalysts for Hydrocyanic acid (HCN) synthesis [20]. Solutions of this task are composed of two discrete and 11 continuous variables, the whole dataset has 696 items. The dataset was randomly split into training (556) and testing (140) part.

**Table 1.** Surrogate-models' regression results on HCN and Schwefel dataset, average results from 50 trainings. RMSE on the testing and training set is supplemented by average medians of residuals on the testing set.

| dataset | model | RMSE (test set) | RMSE (train set) | medians of residuals |
|---|---|---|---|---|
| HCN | DSCL | $8.2918 \pm 0.4373$ | 7.6337 | 4.9433 |
| | RESID | $7.6212 \pm 0.3187$ | 6.6131 | 4.0678 |
| | SUMO | $115.196 \pm 34.1387$ | 0.5711 | 35.0051 |
| Schwefel | DSCL | $62.712 \pm 3.4248$ | 83.553 | 4.955 |
| | RESID | $55.605 \pm 5.4867$ | 73.468 | 5.411 |
| | SUMO | $64576 \pm 1.9\text{e}{+05}$ | 33.964 | 44048 |

As the second dataset, individuals from the first 10 generations of a run of GA optimization of modified Schwefel's function were taken. The original Schwefel's function $y = \frac{1}{p}\sum_{i=1}^{p} -x_i \sin(\sqrt{|x|})$ was modified in order to be defined on both continuous $(x_1^{(\mathrm{C})}, \ldots, x_p^{(\mathrm{C})}) \in [-512{,}512]^p$ and discrete $(x_1^{(\mathrm{D})}, \ldots, x_p^{(\mathrm{D})}) \in \{-10, -9, \ldots, 10\}$ variables in the following form

$$\begin{aligned} y \quad = \quad & \frac{1}{p}\sum_{i=1}^{p} -x_i^{(\mathrm{C})} \sin(\sqrt{|x_i^{(\mathrm{C})}|}) + \frac{10}{p}\sum_{i=1}^{p} \sin(\sqrt{|\pi(x_i^{(\mathrm{D})}) .* x_i^{(\mathrm{C})}|}) \\ & + \frac{20}{p}\sum_{i=1}^{p} \pi(x_i^{(\mathrm{D})})^2 \end{aligned} \tag{9}$$

where $\pi : \{-10, \ldots, 10\} \to \{-10, \ldots, 10\}$ is a random permutation. For simplicity, we have chosen only two continuous and two discrete variables ($p = 2$); the training set has 681 and testing set 101 data.

Our models (RESID and former DSCL) were compared with the RBF network from SUMO toolbox [15] using approximately the same computational time. Each of the three models (DSCL, RESID and SUMO) were 50 times trained on the training sets of the two tasks.

Results in Table 1 show that the new RESID model achieves about 10% lower root-mean-square error (RMSE) than the former DSCL model; the improvement is statistically significant (one-sided Mann–Whitney U test, $p_{\mathrm{HCN}} = 2.28 \cdot 10^{-12}$, $p_{\mathrm{Schwefel}} = 1.11 \cdot 10^{-10}$). The testing errors of both our models are considerably lower than errors of RBF networks from SUMO toolbox. Moreover, in the case of the HCN real-world dataset, the improvement of RESID model is even more than 25% compared to the results in [11]. Even though SUMO toolbox's training error is low, very large testing error shows poor generalization capabilities; see also Figure 2.

### 4.2 Genetic Algorithm Performance on the Benchmark Fitness

The Schwefel's benchmark fitness enabled us to test the model as a surrogate model for genetic optimization. The parameters of the function, especially the number of variables, was the same as in the case of the regression test.

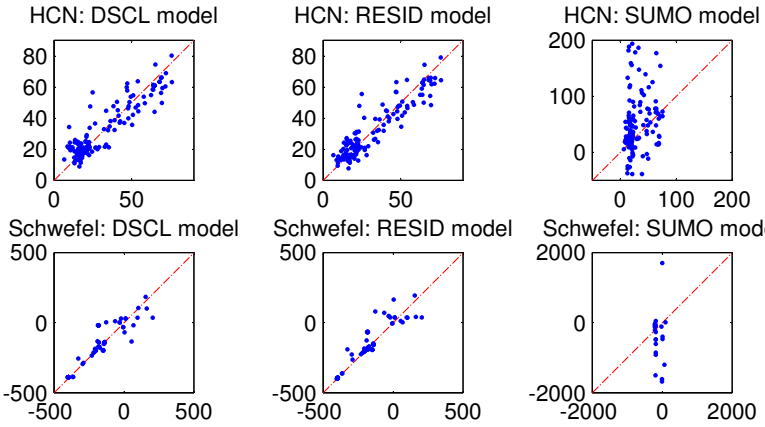



**Fig. 2.** Scatter plots of the DSCL, RESID and SUMO's RBF models on testing data

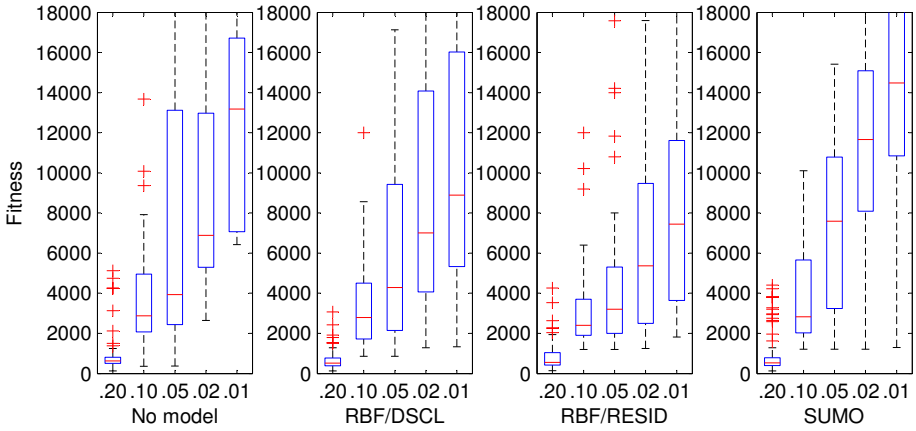



**Fig. 3.** The numbers of original evaluations needed to reach 1.2-, 1.1-, 1.05-, 1.02- and 1.01-multiple of minimum, measured on 100 GA runs

Crucial criterion of successful optimization of empirical functions is the number of original fitness evaluations needed for reaching sufficient near-optimal solution. Therefore, the number of original fitness evaluations was measured in each of 100 runs in the moment when the following thresholds were reached: 1.2-, 1.1-, 1.05-, 1.02- and 1.01- multiple of the global minimum (corresponding to reaching value 20, 10, 5, 2 and 1 per cent above optimal solution). The numbers of needed evaluations for these limits are shown in Figure 3. Since not every single run converged to all of these values, the numbers of evaluations belonging to each threshold were divided by the ratio of runs successfully reaching respective thresholds.

The average number of evaluations needed to get 1 %-above-optimal solution has been significantly decreased from 13224 by more than 35 % to 8310 with our RESID model ($p = 0.011$, one-sided Mann–Whitney U test), and by nearly 25 % (to 10014) with the DSCL model; this result is not significant, though. RBF networks from the SUMO toolbox did not improve results of this optimization task (the average number of original evaluations was 13249).

## 5 Conclusion

This paper presented a novel kind of surrogate model for mixed-variable continuous and discrete optimization. It utilizes a clustered model with RBF networks (one network per cluster) and a generalized linear model. The model focuses training of the RBF networks using clustering on the discrete part of the data. Generalized linear model trained on the discrete input variables further improves regression capabilities. Results of testing on two different datasets showed that the model is a competitive kind of regression for costly objective functions. Using this surrogate model for the optimization of modified Schwefel's benchmark fitness function resulted in saving up to 35 per cent of the original evaluations.

**Acknowledgement.** This work was supported by the Grant Agency of the Charles University (GA UK), grant No. 278511/2011, and by the Czech Science Foundation (GA CR), grant No. P202/11/1368 and No. 201/08/0802.

## References

1. Olhofer, M., Arima, T., Sendhoff, T.S.B., Japan, G.: Optimisation of a Stator Blade Used in a Transonic Compressor Cascade with Evolution Strategies. In: Evolutionary Design and Manufacture, pp. 45–54 (2000)
2. Holeňa, M., Cukic, T., Rodemerck, U., Linke, D.: Optimization of catalysts using specific, description-based genetic algorithms. Journal of Chemical Information and Modeling 48(2), 274–282 (2008)
3. Booker, A., Dennis, J., Frank, P., Serafini, D., Torczon, V., Trosset, M.: A rigorous framework for optimization by surrogates. Structural and Multidisciplinary Optimization 17, 1–13 (1999)
4. Gorissen, D., Dhaene, T., DeTurck, F.: Evolutionary model type selection for global surrogate modeling. Journal of Machine Learning Research 10, 2039–2078 (2009)
5. Hemker, T., Fowler, K., Farthing, M., von Stryk, O.: A mixed-integer simulation-based optimization approach with surrogate functions in water resources management. Optimization and Engineering 9(4), 341–360 (2008)
6. Holmström, K., Quttineh, N.H., Edvall, M.: An adaptive radial basis algorithm (ARBF) for expensive black-box mixed-integer constrained global optimization. Optimization and Engineering 9(4), 311–339 (2008)
7. Abramson, M.A.: Mixed variable optimization of a load-bearing thermal insulation system using a filter pattern search algorithm. Optimization and Engineering 5(2), 157–177 (2004)

8. Younis, A., Dong, Z.: Global optimization using mixed surrogate models for computation intensive designs. In: 2nd International Symposium on Computational Mechanics, Hong Kong, vol. 1233, pp. 1600–1605 (2010)
9. Singh, G., Grandhi, R.V.: Mixed-Variable optimization strategy employing multifidelity simulation and surrogate models. AIAA Journal 48(1), 215–223 (2010)
10. Müller, J., Shoemaker, C.A., Piché, R.: SO-MI: a surrogate model algorithm for computationally expensive nonlinear mixed-integer black-box global optimization problems. Computers & Operations Research (in press, 2012)
11. Bajer, L., Holeňa, M.: Surrogate Model for Continuous and Discrete Genetic Optimization Based on RBF Networks. In: Fyfe, C., Tino, P., Charles, D., Garcia-Osorio, C., Yin, H. (eds.) IDEAL 2010. LNCS, vol. 6283, pp. 251–258. Springer, Heidelberg (2010)
12. Schaefer, R., Telega, H.: Foundations of global genetic optimization. Springer (2007)
13. Buhmann, M.D.: Radial basis functions: theory and implementations. Cambridge Univ. Press (2003)
14. Myers, R.H., Montgomery, D.C., Anderson-Cook, C.M.: Response surface methodology: process and product optimization using designed experiments, vol. 705. John Wiley & Sons Inc. (2009)
15. Gorissen, D., Couckuyt, I., Demeester, P., Dhaene, T., Crombecq, K.: A surrogate modeling and adaptive sampling toolbox for computer based design. Journal of Machine Learning Research 11, 2051–2055 (2010)
16. Buche, D., Schraudolph, N., Koumoutsakos, P.: Accelerating evolutionary algorithms with gaussian process fitness function models. IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews 35(2), 183–194 (2005)
17. McCullagh, P., Nelder, J.A.: Generalized linear models. Chapman & Hall (1989)
18. Holeňa, M., Linke, D., Rodemerck, U., Bajer, L.: Neural Networks as Surrogate Models for Measurements in Optimization Algorithms. In: Al-Begain, K., Fiems, D., Knottenbelt, W. (eds.) ASMTA 2010. LNCS, vol. 6148, pp. 351–366. Springer, Heidelberg (2010)
19. Schwefel, H.: Numerische Optimierung Von Computer-Modellen Mittels Der Evolutionsstrategie: Mit Einer Vergleichenden Einführung in Die Hill-Climbing-und Zufallsstrategie. Birkhäuser (1977)
20. Möhmel, S., Steinfeldt, N., Endgelschalt, S., Holeňa, M., Kolf, S., Dingerdissen, U., Wolf, D., Weber, R., Bewersdorf, M.: New catalytic materials for the high-temperature synthesis of hydrocyanic acid from methane and ammonia by high-throughput approach. Applied Catalysis A: General 334, 73–83 (2008)

# Computing Semantic Similarity Using Large Static Corpora

András Dobó and János Csirik

University of Szeged, Institute of Informatics, Szeged, Hungary
{dobo,csirik}@inf.u-szeged.hu

**Abstract.** Measuring semantic similarity of words is of crucial importance in Natural Language Processing. Although there are many different approaches for this task, there is still room for improvement. In contrast to many other methods that use web search engines or large lexical databases, we developed such methods that solely rely on large static corpora. They create a binary or numerical feature vector for each word making use of statistical information obtained from the corpora. These vectors contain features based on context words or grammatical relations extracted from the corpora and they employ diverse weighting schemes. After creating the feature vectors, word similarity is calculated using various vector similarity measures. Beside the individual methods, their combinations were also tested. Evaluated on both the Miller-Charles dataset and the TOEFL synonym questions, they achieve competitive results to recent methods.

**Keywords:** semantic similarity, static corpora, co-occurrence statistics.

## 1 Introduction

For many Natural Language Processing (NLP) tasks, such as information extraction, spelling correction or word sense disambiguation, knowing the semantic similarity of words can be very helpful. Therefore, in the last approximately 20 years, much research has been done on developing methods that can automatically compute the semantic similarity of words. Most of the best performing methods employ web search engines (for example Google or Yahoo!) or large lexical databases (such as WordNet or Roget's Thesaurus) in order to compute word similarity. Although their application can be advantageous for many reasons, and systems using them tend to perform well, they also have many disadvantages.

Using web search engines in NLP tasks can have many drawbacks, as noted by Kilgarriff [9] among others. First, the returned page hit counts are not exact counts and they change over time. Furthermore, queries can have no linguistic restrictions and punctuation cannot be used. Moreover, their use can be limited and time consuming. Finally, they usually have a constraint on the number of pages returned per query.

Employing large lexical databases induce other problems. Methods that can automatically compute the semantic similarity of words are especially useful for uncommon words not included in lexical databases and thesauri. However, an algorithm that solely relies on lexical databases is not able to compute the similarity of such words,

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 491–502, 2013.

and therefore cannot be used in those cases when the most useful they would be. Furthermore, as languages evolve over time and new words are created every day, these lexical databases should be revised constantly, which is a costly task. Moreover, every manually created database is prone to human errors: important words, word meanings (called synsets in WordNet) and relations can be missing from them.

Summing up, there are situations, where the usage of web search engines or large lexical databases is not suitable or feasible because of the above mentioned problems. In those cases, such methods are needed that use neither web search engines nor lexical databases. Therefore, we constructed methods that solely rely on large static corpora.[1] They first process the used corpora and create a feature vector for each word using context words or grammatical relations as features and some weighting scheme. Then, they compute word similarity based on the similarity of these word vectors. Beside using the created methods by themselves, a number of combination of the different methods were also examined. Tested on two different datasets, namely the TOEFL synonym questions and the Miller-Charles word pairs, they give comparable results to other methods.

The rest of the paper is structured as follows. We first give a short overview of the different kinds of existing methods used for computing semantic similarity in Section 2. Then, in Section 3, we describe our methods in detail. Finally, in Sections 4 and 5, we demonstrate our results and draw conclusions from them.

## 2 Related Methods

The methods computing semantic similarity can use a variety of sources and can compute the semantic similarity of words differently. There exist methods that make use of large lexical databases, such as the WordNet or the Roget's Thesaurus. Others issue web search engine queries and process the results. Further, there are also numerous methods that employ large static corpora to extract statistical information in order to solve the problem of semantic similarity. In this section, we would like to give a short overview of all these approaches.

### 2.1 Methods Using Large Lexical Databases

The methods using large lexical databases access the information stored in these databases and compute word similarity based on the extracted information. Most of them use the WordNet, but others apply the Roget's Thesaurus.

As an example, Jarmasz and Szpakowicz [8] defines the similarity of two words based on their distance in Roget's Thesaurus, i.e. the number of edges between them.

A bag-of-words method based on WordNet was proposed by Patwardhan and Pedersen [15]. For each input word it creates a feature vector from the words contained in their gloss and the words with distance 1 from that input word. The similarity of words is then defined as the cosine similarity of their vectors.

Tsatsaronis et al. [19] also defines a similarity score using WordNet. To compute this, they consider the distance of words in WordNet, the depth of the nodes between them

[1] Although WordNet is used for obtaining the lemmas of words, it is not used for anything else. This could be substituted with other methods though.

and the types of relations on the route between them (they use all the relation types that can be found in WordNet). They extended their measure, so that it is able to compare not only words, but also longer texts.

### 2.2 Web Search-Based Methods

There are also numerous methods that try to estimate the similarity of words by issuing web search queries with the given words and then using the returned page hit counts and snippets. The most important of these are presented next.

Higgins [7] first issues queries with the words to be compared independently. Then, he also issues queries in which the two words are next to each other. Finally, the similarity of these words is defined as their pointwise mutual information computed from the returned page hit counts.

Sahami and Heilman [18] collect the snippets returned for input queries. For each snippet they create a vector with TF-IDF weighting. Then, the vectors are normalized and the centroids of the set of vectors returned for a query are computed. The similarity of queries is computed as the inner product of the centroids associated with them.

The method of Kulkarni and Caragea [10] is composed of two parts. The first part assigns to any input word a set of the most associated words with it, thus creating a concept cloud for each input word. Then, in the second part, these concept clouds are compared, and the semantic similarity of the words is determined by the similarity of their concept clouds. In both parts they issue web search queries.

### 2.3 Methods Employing Large Static Corpora

Methods in this category usually build a vector for each word based on their contexts found in the used static corpora and define word similarity as the similarity of their vectors. Although our methods are similar to those below, ours use new features, weighting schemes and vector similarity measures in addition to the existing ones.

The method called Latent Semantic Analysis (LSA), introduced by Landauer and Dumais [11], is very similar to Latent Semantic Indexing (LSI). It first creates a matrix of words and chunks of text (e.g. sentences or paragraphs), where the cells contain the weight of the words regarding the chunks of text. Then, it applies Singular Value Decomposition (SVD) to compress this matrix. Finally, it computes the similarity of words based on the similarity of their vectors in this compressed matrix.

The method of Lin [12] assigns a feature set to every input word, which contains those (grammatical relation, feature word)-pairs that co-occurred with the input word in a corpus. Similarity is then defined using the information content of the feature sets of the words as well as the information content of the intersection of their feature sets.

The method proposed by Rapp [16] creates a numerical feature vector for words based on the contexts they have in a corpus. In these vectors those words are contained, that occur within a 2 word window in the used corpus, and their score is based on word association measures such as pointwise mutual information. The matrix formed by these feature vectors is then compressed using SVD. Finally, the similarity of the words is computed as the similarity of their compressed feature vectors.

Gabrilovich and Markovitch [6] maps input texts into weighted vectors over Wikipedia articles, based on the similarity of the texts and the articles. Then, they compute the similarity of input texts as the cosine similarity of their vectors.

### 2.4 Combined Methods

All methods of the three categories above have their advantages and disadvantages. Methods using lexical databases are usually exact for the words included in them but have poor coverage on specific professional domains. Using large static corpora provides a great opportunity for inspecting the distribution of neighboring words for any word, but they can still provide less than enough data for very rare words. By issuing web search queries there is enough information for almost every word, but such methods have the many disadvantages described in Section 1. Because of these different characteristics, a number of researches tried to combine different types of methods in order to combine the best properties of them, thus creating new methods.

Resnik [17] assigns a probability for each synset in WordNet, based on the probability of the occurrence of its words in a static corpus. The similarity of words is then given as the maximum information content of their least common subsumer synset.

Lin [12] defined another measure, which is very similar to Resnik's [17]. The difference is that beside the information content of the least common subsumer of the input words, it employs the information content of the words' synsets too.

The method of Turney et al. [20] is a combination of 4 different methods. One is the LSA [11], the second is a web search based method called PMI-IR, the third searches in an online thesaurus (Wordsmyth thesaurus online) and the last processes the snippets returned by web queries. These 4 methods were then combined in different ways, such as with the product rule, to provide a final similarity measure.

The hyperlink structure of the Wikipedia was utilized by Milne and Witten [14]. They employ the anchors found in the articles (links to other articles), and they represent each article with the set of its incoming and outgoing links. First, they identify candidate articles for each input word using the text of anchors in the article. Then, after resolving ambiguities using different approaches, they compute the similarity of the articles associated with the input words using the page hit counts of web queries.

Agirre et al. [1] use a combination of two methods. The first assigns a vector to each word by running the PageRank algorithm on WordNet. The other uses statistical co-occurrence data from a corpus of 1.6 Terawords, and has 3 versions: a bag-of-words approach, a version using a static context window and another using dependency relations. In both methods, word similarity is computed using a vector similarity measure. To combine the two main methods, they train an SVM.

## 3 Our Methods

The main idea behind our approach, as behind most other ones, is that semantically similar words behave similarly and occur in similar contexts. Therefore, our algorithms first create a feature vector for each word based on statistical co-occurrence information gathered from corpora, which is followed by the comparison of word pairs using a vector similarity measurement. There are several variations of our method, using different

features, vector types, weights and vector comparison measures. An earlier version of some of these methods was previously described in Dobó [4].

### 3.1 Feature Extraction

For the task of extracting features from the used corpora we applied two main approaches. The first and simpler one is the bag-of-words approach. It finds each occurrence of the selected word in the used corpora, then includes every word in a window of 3 words within that occurrence in the feature vector. However, it is different from regular bag-of-words approaches, since (in case of using numerical feature vectors) in addition to the weighting scheme it uses, described in Section 3.3, it counts the occurrences of close words multiple times. Specifically, the frequency this method assigns to a feature word is based on the distance of the observed word and the feature word. Several different techniques were tested, the best was found to be using frequencies that scale quadratically with the distance (with a window size of 3, frequency 9 is assigned to distance 1, frequency 4 to distance 2 and frequency 1 to distance 3).

Our other approach uses features based on grammatical relations obtained from the used corpora. Grammatical relations were extracted using the C&C CCG parser [2]. For each word, (grammatical relation, feature word)-pairs are included in the feature vector, where the feature words are those that are in a grammatical relation with the original word, similarly as in Lin [12]. Some example features are (subject-of, word), (object-of, word) and (preposition, word) among others. It is important to note that in this approach paraphrases, prepositions, patientive ambitransitive verbs and passive verbs were treated similarly as in Dobó [4] and in Dobó and Pulman [5].

Both approaches were tested using three corpora, namely the British National Corpus (BNC), the Web 1T 5-gram Corpus (only the 4 and 5-grams), and the corpus of the English Wikipedia[2]. Since any corpus can be used to create the feature vectors, our methods can be easily adapted to different domains and languages if needed.

### 3.2 Creating and Comparing the Feature Vectors

For creating and comparing the feature vectors of words two main approaches were tested. First, the approach presented by Lin [12] (the one using static corpora and not using WordNet, presented in Section 2.3) was re-implemented with some modifications. This method uses binary feature vectors (i.e. feature sets), indicating whether a feature has occurred with the given word, without weight. Then, to compute the similarity of words, it compares these feature vectors using the similarity measure of Lin [12], which assigns a similarity score of 0 to 1 (inclusive) for a word pair. Although the base of this approach is the same as described in Lin [12], in order to improve its performance there were some changes made in its implementation regarding the treatment of paraphrases, patientive ambitransitive verbs and passive verbs, similarly as in Dobó [4] and in Dobó and Pulman [5].

[2] Pre-processed using the wikipedia2text_rsm_mods toolkit by Rafael Mudge, available from http://blog.afterthedeadline.com/2009/12/04/generating-a-plain-text-corpus-from-wikipedia

The previous approach does not take into account the frequency with which a feature co-occurred with a word. But, this co-occurrence frequency also contains useful information, so it is logical to try to use that information too. Therefore another method was created that does not only store the features for a word in a set, rather it creates a weighted numerical vector from them. Then, the similarity of these vectors provides the similarity of words. The types of weights and the vector comparison measures used in this approach are described in Section 3.3 and 3.4, respectively.

### 3.3 Weighting Inside the Numerical Feature Vectors

Weighting can be used to assign importance to the features, and thus to consider different aspects of the features significant. Seven different weighting schemes were tested inside the numerical feature vectors. The description of these follows now.

The simplest of them is the co-occurrence frequency of the (word, feature) pairs (freq). In this case, the importance of a feature is based on the number of times it co-occurred with the word.

The second weighting is a slightly modified version of the first one. Instead of the simple frequencies of the (word, feature) pairs, the logarithm of this frequency is stored, with a smoothing parameter of 1 (logfreq). This way, (word, feature) pairs with very high frequency cannot be overweighted.

The problem of the first two weighting methods is that they assign an overly high importance to those features that occur very frequently in any context, such as the features (subject-of, be) or (object-of, have). It would be better to assign a low importance to features like these, since they do not tell much information about the words they are connected to, and assign high importance to those features instead that are specific to the words. One such measure is the pointwise mutual information (pmi) [3], which measures association strength. This was chosen as the third type of weight. However, since it is unstable for very small counts [3], (word, feature) pairs with a frequency of at most 5 are discarded when using this weighting scheme.

Another way for testing the strength of association is using the log-likelihood ratio [13], which was also employed as a weight in the numerical feature vectors (loglh).

The fifth measure was a combination of two different measures. The first is the logarithm of the frequency of the (word, feature) pairs. But the problem is, as noted before, that it assigns a high value for the most common features that are not specific to any word. This is compensated by the second part, which is the logarithm of the number of words that occur with the given feature. Both logarithms use a smoothing parameter of 1. The combined measure is calculated as the quotient of the two parts:

$$qw(x,y) = \frac{\ln(1+c_{xy})}{\ln(1+f(y))} \tag{1}$$

where $c_{xy}$ is the frequency of the co-occurrence of x and y, and $f(y)$ is the number of words, with which feature y occurs [12].

The sixth weight is also a combined measure. Its first part is again the logarithm of the frequency of the (word, feature) pairs. And its second part, which is the logarithm of the information content of the feature, is again used for compensation. In case of both

logarithms, a smoothing parameter of 1 was used. The two parts are multiplied together to form the combined measure:

$$pw(x,y) = \ln(1 + c_{xy}) \times \ln(1 + I(y)) \quad (2)$$

where $c_{xy}$ is same as before and $I(y)$ is the information content of feature $y$ [12].

The last implemented weighting measure is the entropy-based measured used by Rapp [16].

### 3.4 Similarity Measures for the Numerical Feature Vectors

Two frequently used vector similarity measures were tested in the algorithms with numerical feature vectors. The first was the cosine similarity, which compares two vectors by computing the cosine of the angle between them [13]. The second was a generalization of the Dice coefficient. In its original form, it can only compute the similarity between Boolean vectors. In order to use it for numerical vectors, we used its generalization proposed by Lin [12]. Both similarity measures return a similarity value between -1 and +1 (inclusive).

### 3.5 Determining the Part-of-Speech of the Input Words

There are many words that can take more than one part-of-speech (POS). For example, the words run and bank can be both nouns and verbs. When these types of words are used with different POSs, different features are relevant. Therefore, first the POS of the input words needs to be determined, and the feature vectors can only be created after this. The POS of these controversial words can be inferred from the other words

**Table 1.** Results on the Miller-Charles dataset (Spearman correlation). Notations: bnc/enwiki /web1t5gram denotes the corpus; bagofwords/parsed denotes the used feature types (bag-of-words or grammatical relations); lin/num denotes the method (the one based on Lin [12] or the one using numerical feature vectors); cos/dice denotes the similarity measure; freq/logfreq/pmi/ loglh/qw/pw/rapp denotes the weighting scheme; + denotes the combination of two methods.

| Method | Result |
|---|---|
| bnc-bagofwords-num-cos-qw+enwiki-parsed-num-cos-freq | 0.773 |
| bnc-bagofwords-num-cos-qw+enwiki-parsed-num-cos-qw | 0.750 |
| enwiki-bagofwords-num-cos-pmi+bnc-parsed-num-cos-qw | 0.737 |
| enwiki-bagofwords-num-cos-pmi+enwiki-parsed-num-cos-pmi | 0.729 |
| enwiki-parsed-num-cos-pmi | 0.727 |
| bnc-parsed-num-cos-loglh+enwiki-parsed-num-cos-pmi | 0.712 |
| enwiki-bagofwords-num-cos-pmi | 0.684 |
| enwiki-parsed-num-dice-pmi | 0.661 |
| web1t5gram-parsed-num-cos-loglh | 0.631 |
| enwiki-bagofwords-num-cos-pmi+enwiki-parsed-lin | 0.616 |
| bnc-parsed-lin | 0.417 |

contained in the same question. For our methods, we assumed that each input word is a verb, noun, adjective or adverb and each question contains words of the same POS.

For a question the part-of-speech maximizing the following formula is chosen:

$$pos = \arg\max_{p} \prod_{w \in q} \ln(1.0001 + f_{w,p}) \tag{3}$$

where $p$ can take any of the four possible POSs, $q$ denotes the question, $w$ runs through the words of $q$ and $f_{w,p}$ is the frequency of $w$ having $p$ part-of-speech.

### 3.6 Combination of the Individual Methods

In order to combine the strengths of the different methods and achieve better results, not only the above described methods, but their combinations were also tested. When two methods were combined, the similarity score for each word pair was calculated separately. Afterwards, the logarithm of the scores (with a smoothing parameter of 1) were multiplied together to form the similarity score of the combined method. Taking the logarithm of the scores before multiplying them helps balancing the results: we consider a word pair having two moderate scores better than a word pair having a very low and a very high score.

## 4 Results

All the methods described in the previous section were tested on two different datasets, namely the Miller-Charles word pairs (MC) and the TOEFL synonym questions. Both data sets were widely used in the evaluation of methods computing semantic similarity by others. The first one contains 30 word pairs, for which a similarity score between 0 to 4 was assigned by 38 undergraduate students. Since there were words in 2 word pairs that were not included in previous WordNet versions, in most research these pairs were omitted. Consequently, only the remaining 28 word pairs were used here as well. The other data set contains 80 synonym questions from the TOEFL language exam. In all of the questions, a question word is given with 4 alternatives, and the task is to determine the most similar word to the question word.

In case of the MC dataset, the average scores of the 38 students were used, and the evaluation was done by computing the Spearman correlation of these scores and the scores returned by our methods. When testing with the TOEFL questions, the evaluation measure was the percentage of the correct answers given.

The results of some selected methods are presented in Tables 1 and 2. It can be seen that the best performance was 0.773 on the MC dataset and 88.75% on the TOEFL questions, both achieved by a combined method. The best results of individual methods (without combination) were 0.727 and 83.75%, respectively. The scores of the methods using numerical feature vectors were mostly higher than the scores of the approach based on Lin [12], and still better results were achieved by combining the different methods. The methods that achieved best performance considering both datasets were the *bnc-parsed-num-cos-loglh+enwiki-parsed-num-cos-pmi* (MC: 0.712, TOEFL: 88.75%), the *enwiki-bagofwords-num-cos-pmi+enwiki-parsed-num-cos-pmi* (MC:

**Table 2.** Results on the TOEFL questions (percent of correct answers)

| Method | Result |
|---|---|
| bnc-parsed-num-cos-loglh+enwiki-parsed-num-cos-pmi | 88.75% |
| enwiki-bagofwords-num-cos-pmi+enwiki-parsed-num-cos-pmi | 87.50% |
| enwiki-bagofwords-num-cos-pmi+bnc-parsed-num-cos-qw | 86.25% |
| enwiki-bagofwords-num-cos-pmi | 83.75% |
| enwiki-parsed-num-cos-pmi | 82.50% |
| enwiki-bagofwords-num-cos-pmi+enwiki-parsed-lin | 81.25% |
| enwiki-parsed-num-dice-pmi | 78,75% |
| bnc-bagofwords-num-cos-qw+enwiki-parsed-num-cos-qw | 77.50% |
| bnc-bagofwords-num-cos-qw+enwiki-parsed-num-cos-freq | 72.50% |
| bnc-parsed-lin | 68.75% |
| web1t5gram-parsed-num-cos-loglh | 60.00% |

**Table 3.** Comparison with other results on the Miller-Charles dataset (Spearman correlation)

| Method | Result | Used data |
|---|---|---|
| Human upper bound [17] | 0.934 | |
| Agirre et al. [1] | 0.92 | WordNet, corpus |
| Patwardhan and Pedersen [15] | 0.91 | WordNet |
| Jarmasz and Szpakowicz [8] | 0.87 | Roget's Thesaurus |
| Tsatsaronis et al. [19] | 0.856 | WordNet |
| Kulkarni and Caragea [10] | 0.835 | Web search |
| Lin [12] | 0.82 | WordNet, corpus |
| Resnik [17] | 0.81 | WordNet, corpus |
| bnc-bagofwords-num-cos-qw+ enwiki-parsed-num-cos-freq | 0.773 | corpus |
| bnc-bagofwords-num-cos-qw+ enwiki-parsed-num-cos-qw | 0.750 | corpus |
| enwiki-bagofwords-num-cos-pmi+ bnc-parsed-num-cos-qw | 0.737 | corpus |
| enwiki-bagofwords-num-cos-pmi+ enwiki-parsed-num-cos-pmi | 0.729 | corpus |
| Gabrilovich and Markovitch [6] | 0.72 | corpus |
| bnc-parsed-num-cos-loglh+ enwiki-parsed-num-cos-pmi | 0.712 | corpus |
| Milne and Witten [14] | 0.70 | Wikipedia links, Web search |
| Sahami and Heilman [18] | 0.618 | Web search |

0.729, TOEFL: 87.50%) and the *enwiki-bagofwords-num-cos-pmi+bnc-parsed-num-cos-qw* (MC: 0.737, TOEFL: 86.25%). Comparison with other methods is shown in Tables 3 and 4, which shows that our methods had an average performance on the MC dataset, while one of our methods achieving the 3rd best score on the TOEFL questions. Most importantly, however, if we only take those methods into account that solely use static corpora, our methods perform 1st and 2nd best on the two datasets, respectively.

## 5 Conclusion and Future Work

In this article we have demonstrated methods computing the semantic similarity of words that compare favorably to other measures. They use statistical co-occurrence data extracted from static corpora to create feature vectors for the input words, and then define the similarity of the words as the similarity of their vectors. Several variations were created, using different features, vector types, weights and vector comparison measures, and combinations of our individual methods were also examined.

All these methods were tested on two datasets, namely the Miller-Charles dataset (MC) and the TOEFL synonym questions. On the MC dataset our best method had an average performance (0.773), with many others achieving better results. On the other hand, our best accuracy of 88.75% on the TOEFL questions is 3rd best overall and is much higher than the score achieved by an average non-English US college applicant. When comparing our methods with only those methods that solely rely on static corpora, according to our best knowledge they reach 1st and 2nd place on the two datasets, respectively.

**Table 4.** Comparison with other results on the TOEFL questions (percent of correct answers)

| Method | Result | Used data |
|---|---|---|
| Turney et al. [20] | 97.5% | Web search, thesaurus |
| Rapp [16] | 92.5% | corpus |
| bnc-parsed-num-cos-loglh+ enwiki-parsed-num-cos-pmi | 88.75% | corpus |
| enwiki-bagofwords-num-cos-pmi+ enwiki-parsed-num-cos-pmi | 87.50% | corpus |
| Tsatsaronis et al. [19] | 87.5% | WordNet |
| enwiki-bagofwords-num-cos-pmi+ bnc-parsed-num-cos-qw | 86.25% | corpus |
| enwiki-bagofwords-num-cos-pmi | 83.75% | corpus |
| Higgins [7] | 81.3% | Web search |
| Jarmasz and Szpakowicz [8] | 78.7% | Roget's Thesaurus |
| Average non-English US college applicant [11] | 64.5% | |
| Landauer and Dumais [11] | 64.3% | corpus |
| Lin [12] | 24.0% | WordNet, corpus |
| Resnik [17] | 20.3% | WordNet, corpus |

Based on that, we think that our best methods could be successfully used for solving real-life problems too. The fact that they perform better on the TOEFL questions than on the MC dataset indicates that they are more suitable for selecting the most similar word for an input word from a list of candidates than giving an exact similarity value for a pair of words.

In the future, it would be worthy to test our methods with even larger corpora, as more data can result in better accuracy (for example, Agirre et al. [1] use a corpus of 1.6 Terawords and run their algorithm on 2000 CPU cores). As any corpus can be used

to extract co-occurrence information, our methods could easily be adapted to different languages (especially in case of the bag-of-words approach). Therefore, we would like to try our algorithms with languages other than English, too. Furthermore, as described in Section 2.4, methods that are a combination of different types of methods can combine the advantages of those methods combined. We therefore think that by creating a combined method whose one method is ours and the other method(s) is (are) using web search engines or large lexical databases, our results could be further improved.

## References


1. Agirre, E., Alfonseca, E., Hall, K., Kravalova, J., Paşca, M., Soroa, A.: A study on similarity and relatedness using distributional and WordNet-based approaches. In: 10th Annual Conference of the North American Chapter of the Association for Computational Linguistics - Human Language Technologies, pp. 19–27. Association for Computational Linguistics, Stroudsburg (2009)
2. Clark, S., Curran, J.R.: Parsing the WSJ using CCG and log-linear models. In: 42nd Annual Meeting on Association for Computational Linguistics, pp. 103–110. Association for Computational Linguistics, Stroudsburg (2004)
3. Church, K.W., Hanks, P.: Word association norms, mutual information, and lexicography. Computational Linguistics 16, 22–29 (1989)
4. Dobó, A.: Angol szavak szinonimáinak automatikus keresése. In: National Scientific Conference of Students (OTDK), OTDT, Budapest (2011)
5. Dobó, A., Pulman, S.G.: Interpreting noun compounds using paraphrases. Procesamiento del Lenguaje Natural 46, 59–66 (2011)
6. Gabrilovich, E., Markovitch, S.: Computing Semantic Relatedness using Wikipedia-based Explicit Semantic Analysis. In: 20th International Joint Conference on Artificial Intelligence, pp. 1606–1611. Morgan Kaufmann Publishers Inc., San Francisco (2007)
7. Higgins, D.: Which Statistics Reflect Semantics? Rethinking Synonymy and Word Similarity. In: Kepser, S., Reis, M. (eds.) Linguistic Evidence: Empirical, Theoretical and Computational Perspectives, pp. 265–284. Mouton de Gruyter, Berlin (2005)
8. Jarmasz, M., Szpakowicz, S.: Roget's Thesaurus and Semantic Similarity. In: 4th Conference on Recent Advances in Natural Language Processing, pp. 212–219. John Benjamins Publishers, Amsterdam (2003)
9. Kilgarriff, A.: Googleology is bad science. Computational Linguistics 33, 147–151 (2007)
10. Kulkarni, S., Caragea, D.: Computation of the Semantic Relatedness between Words using Concept Clouds. In: International Conference on Knowledge Discovery and Information Retrieval, pp. 183–188. INSTICC Press, Setubal (2009)
11. Landauer, T.K., Dumais, S.T.: A solution to Plato's problem: The latent semantic analysis theory of acquisition, induction and representation of knowledge. Psychological Review 104, 211–240 (1997)
12. Lin, D.: An information-theoretic definition of similarity. In: 15th International Conference on Machine Learning, pp. 296–304. Morgan Kaufmann Publishers Inc., San Francisco (1998)
13. Manning, C., Schütze, H.: Foundations of statistical natural language processing. MIT Press, Cambridge (2000)
14. Milne, D., Witten, I.H.: An Effective, Low-Cost Measure of Semantic Relatedness Obtained from Wikipedia Links. In: 23rd AAAI Conference on Artificial Intelligence, pp. 25–30. AAAI Press, Menlo Park (2008)

15. Patwardhan, S., Pedersen, T.: Using WordNet-based Context Vectors to Estimate the Semantic Relatedness of Concepts. In: 11th Conference of the European Chapter of the Association for Computational Linguistics, pp. 1–8. Association for Computational Linguistics, Stroudsburg (2006)
16. Rapp, R.: Word Sense Discovery Based on Sense Descriptor Dissimilarity. In: 9th Machine Translation Summit, pp. 315–322. Association for Machine Translation in the Americas, Stroudsburg (2003)
17. Resnik, P.: Using Information Content to Evaluate Semantic Similarity in a Taxonomy. In: 14th International Joint Conference on Artificial Intelligence, pp. 448–453. Morgan Kaufmann Publishers Inc., San Francisco (1995)
18. Sahami, M., Heilman, T.D.: A web-based kernel function for measuring the similarity of short text snippets. In: 15th International Conference on World Wide Web, pp. 377–386. ACM Press, New York (2006)
19. Tsatsaronis, G., Varlamis, I., Vazirgiannis, M.: Text Relatedness Based on a Word Thesaurus. Journal of Artificial Intelligence Research 37, 1–39 (2010)
20. Turney, P.D., Littman, M.L., Bigham, J., Shnayder, V.: Combining Independent Modules to Solve Multiple-choice Synonym and Analogy Problems. In: 4th Conference on Recent Advances in Natural Language Processing, pp. 482–489. John Benjamins Publishers, Amsterdam (2003)

# The Orchestra of Multiple Model Repositories

Sergejs Kozlovics

Institute of Mathematics and Computer Science, University of Latvia
Raina blvd. 29, LV-1459, Riga, Latvia
sergejs.kozlovics@lumii.lv

**Abstract.** This paper motivates and encourages the simultaneous use of multiple model repositories in model-driven software. A multi-repository mechanism is proposed for that. With this mechanism, multiple model repositories residing in the same or different technical spaces can be mounted into a single repository (in the same sense as file systems are mounted in UNIX) and accessed in a uniform way. Relations (including generalizations) between elements from the mounted repositories are supported. Some of the repositories may be "computable" (virtual), which leads to the on-the-fly model transformation concept.

**Keywords:** models, model repositories, technical spaces, on-the-fly transformation.

*I always imagined that to bring an orchestra to play together is not enough for a conductor.*
— Kurt Masur, a German musician

## 1 Introduction

*Meta*-s can be found far and wide — in Linguistics, Mathematics, art, and music [29]. In Computing, we also have our own *meta*-s. While data representing some system are called a model (Level 1), data about models are called a meta-model (Level 2), data about meta-models are called a meta-metamodel (Level 3), and so on.

In 2002 I. Kurtev, J. Bézivin and M. Aksit have made an observation that there are many widely-accepted technologies that organize their artefacts in such three levels, where a meta-metamodel is usually able to describe itself. Such an organization of artefacts is called the three-level conjecture (this does not necessary restrict the number of possible levels to three, however). This led to the concept of Technical Space (TS), which is an abstraction of all such technologies [34,25].

Table 1 lists several technical spaces and mentions which data stores (hereinafter called *repositories*) they use. We utilize the term *models*, when referring to data stored in those repositories.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 503–514, 2013.

**Table 1.** Technical spaces (TS's) and their repositories

| | |
|---|---|
| MOF TS | *Characteristics:* Models are graphs with attributed nodes and labelled edges. Inspired by CDIF [27] and IRDS [7], the MOF standard [37] by OMG is central in this TS. MOF consists of the two main variants: Essential MOF (EMOF) and Complete MOF (CMOF). OMG is working also on Semantic MOF (SMOF), which will support certain features borrowed from the RDF/OWL TS (see below) [36]. The *de facto* standard in MOF TS, however, is ECore from EMF [4,41]. ECore implements MOF concepts in Java. There is also the KM3 language for defining MOF-like metamodels. We combine all these standards into a single TS called MOF TS.<br>*Meta-levels:* model --→ metamodel --→ meta-metamodel (MOF, ECore, KM3)<br>*Repositories:* EMF/Ecore [4,41]; Enhanced Model Repository [5]; NetBeans MDR [35]; MOF 2 for Java [1]; MetaMart Metadata Repository [18]; CDO [3]; JR [38] |
| XML TS | *Characteristics:* Models are trees with attributed nodes.<br>*Meta-levels:* XML-file --→XML schema --→ XML meta-schema (XSD.xsd)<br>*Repositories:* XML files (there are numerous libraries for parsing/saving XML files) |
| Microsoft DSL Tools TS | *Characteristics:* Similar to MOF, but classes have to be arranged into a tree by means of compositions. Relationships may act as classes, and it is possible to define inheritances between relationships.<br>*Meta-levels:* model --→domain model (metamodel) --→ implicit meta-metamodel, which can be reified [25]<br>*Repositories:* *In-Memory Store* (models are serialized as customizable XMLs) [26, p. 89] |
| Grammarware TS | *Characteristics:* A model is a string, which can be parsed into an abstract syntax tree.<br>*Meta-levels:* text/string --→ grammar --→ EBNF (or similar meta-grammar)<br>*Repositories:* usually text files (tools such as *bison* or *javacc* can be used to generate parsers) |
| GOPPRR (MetaEdit+) TS | *Characteristics:* Similar to MOF. N-ary relationships between concepts are possible. Relationships and their ends (roles) may have properties associated with them.<br>*Meta-levels:* model --→ metamodel --→ GOPPRR (Graph-Object-Property-Port-Role-Relationship)<br>*Repositories:* a proprietary GOPPRR repository [9,30] |
| RDF/OWL TS | *Characteristics:* This TS consists of knowledge representation systems, where all data are encoded in triples (subject, predicate, object). These triples form a graph (subjects and objects are nodes, while predicates are edges). Elements are identified by URIs (uniform resource identifiers).<br>*Meta-levels:* RDF/OWL individuals[16,14,12]--→ RDF vocabulary/OWL ontology--→ RDFS[1]/some OWL variant[2]<br>*Repositories:* Sesame [17]; Virtuoso [20]; OWLIM [15]; JR [38] |

[1] RDF Schema, a language that extends RDF and permits describing taxonomies of classes as RDF vocabularies.

[2] There are the following OWL variants: OWL Lite, OWL DL, OWL Full, OWL 2 (direct semantics and RDF-based semantics), OWL 2 EL, OWL 2 QL, OWL 2 RL (EL/QL/RL can be combined). They differ by expressive power, decidability, and computational complexity for decidable variants [13]. Besides, pure RDF and OWL Full permit having multiple meta-levels and mixing them.

**Table 1.** *Continued*

| | |
|---|---|
| Relational Database TS | *Characteristics:* A classical way to encode entities and relationships by means of tables. No support for generalizations (although they can be simulated).<br>*Meta-levels:* database rows --→ database schema (ER-model) --→ system tables for storing database schemas<br>*Repositories:* numerous database management systems from SQLite to ORACLE |
| Typed Attributed Graphs TS | *Characteristics:* Typed graphs, where nodes and edges may have attributes.<br>*Meta-levels:* graph --→ graph schema --→ typed attributed graph definition<br>*Repositories:* JGraLab [8] and others |
| *Note.* The grouping of different technologies into technical spaces is not strict. | |

The motivation to investigate different technical spaces (TS's) and to support them is driven by the following considerations:

- One TS can be more suitable for the given purpose and more convenient than another. That resembles how one programming language can be more suitable for certain applications than another.
- A person can be more familiar with (i.e., have skills and knowledge in) one TS than with (in) another. If the efforts to study a new TS are big enough, it may be reasonable to stay in a more familiar TS.
- A capability not available in a desired TS can be borrowed from another TS that implements that capability. This encourages "more cooperation than competition among alternative technologies" [25].

However, different TS's store models differently. To ensure interoperability, Bézivin et al. suggest using projectors and extractors — offline transformations of models between TS's. In this paper we propose a mechanism that allows working with multiple repositories (which may belong to the same or different TS's) online. The idea is to mount several repositories into packages in the same way as file systems are mounted into directories in UNIX. Unlike projectors and extractors, our approach uses on-the-fly transformations, thus, deep copying of the data is not required, and all the changes in models become visible immediately.

While mounted repositories can be accessed from their packages (with the possibility to create relations between their elements), certain manipulations with packages can be performed as well (for instance, two packages may be merged). Mounting into packages and manipulating the packages — these are the two pillars of the proposed multi-repository mechanism.

## 2 The Main Idea and Design Choices

### 2.1 The Structure of the Multi-repository Mechanism

The multi-repository mechanism is implemented in a module, which we call the *kernel* in this paper. The kernel represents multiple repositories as a single repository to the *clients* (model transformation and other modules, which work with the models stored in repositories). One of the repositories (we call it the

*pivot repository*) is used to store the information about inter-repository relations. The pivot repository acts as a fully fledged repository as well.

Since different repositories have different APIs,[3] some common abstraction layer is needed. We call it Repository Access API, RAAPI (discussed more in detail in Subsection 3.3). RAAPI consists of primitive low-level operations on model elements. The modules that implement RAAPI for different types of repositories are called *repository adapters*. The same RAAPI is used by the kernel to present multiple repositories to the clients. If it is more convenient for the clients to work with a different API, a wrapper over RAAPI can be created. Figure 1(a) summarizes that.

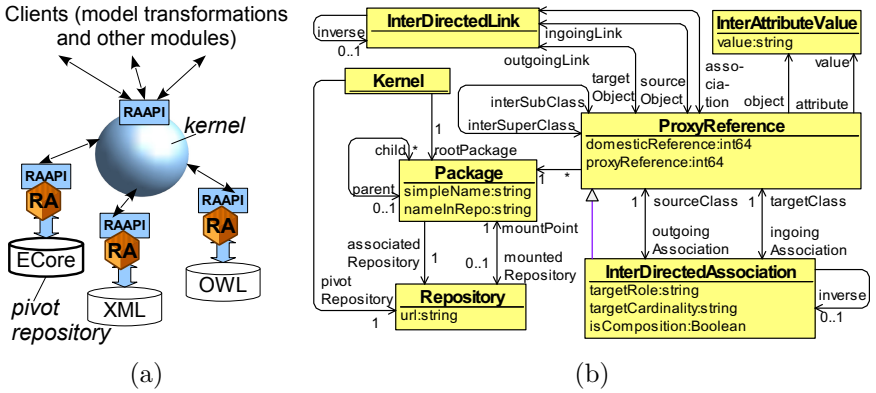



**Fig. 1.** (a) The structure of the multi-repository mechanism. (b) Kernel Metamodel.

This design is based on the following considerations:

- The clients can access multiple repositories in a uniform way using a single API (RAAPI or its wrapper) without the need to be aware of different APIs to access different repositories.
- The clients do not need to switch between different repositories or to specify the desired repository as an argument for each operation on a model. For example, to create a link between two objects from different repositories, a client may assume that the objects are in the same repository: it is the responsibility of the kernel to store and handle this inter-repository link correctly.

### 2.2 Packages as Mount Points

By *package* we mean a group of model elements similar to UML package.[4] We assume that in any repository all the packages form a tree. This resembles how directories are usually organized in a file system as trees.

[3] Application programming interfaces.

[4] If packages are not supported by a particular repository, they can be encoded directly in class names (e.g., "`Package::SubPackage::Class`").

The kernel maintains a rooted tree of packages called the *kernel package tree*. Each kernel package is associated with a package in some repository. The simple (unqualified) names of kernel packages are usually equal to the simple names of the corresponding repository packages, but can be changed at runtime. The names of kernel packages are used by the clients: to refer to a package a client specifies its fully qualified name consisting of simple names starting from the root kernel package.

Initially, the kernel package tree corresponds to the package structure of the pivot repository. When an additional repository is mounted, a new package (a *mount point*) is added to the kernel package tree. The content of the repository will be available via this mount point. For instance, a class can be accessed by concatenating a fully qualified mount point name with the fully qualified class name in the mounted repository.

If a repository can store several models, there are two options:

- treat each model as a separate repository and mount it into a separate package;
- mount the whole repository with all its models at once, but treat each model as a package inside that repository. While it may seem more convenient, there are two shortcomings. First, the repository adapter becomes more complex since it has to perform all necessary actions to represent models as packages. Second, if the repository does not support relations between models, the kernel will not support them as well.

### 2.3 The Role of the Kernel

The kernel deals with elements from different repositories, but needs to represent them as if they were in a single (multi-packaged) repository. This can be performed by introducing *proxy references* to elements. In RAAPI, model elements are referenced by 64-bit integers, which may represent indexes or pointers to elements. Proxy references are also 64-bit RAAPI references, but the kernel ensures they are unique among all the repositories. The kernel maps each proxy reference to the corresponding repository and to the corresponding reference in that repository (proper repository references are called *domestic references*). However, the mapping to the repository is not direct. Instead, each proxy reference maps to a package in the kernel package tree, and each package maps to a repository. Such a design permits changing the repository associated with a package at runtime.

When a particular repository returns a domestic reference, the kernel either creates a new proxy reference, or returns a previously-created one. For this, the kernel maintains a reverse map, which maps pairs <package, domestic reference> to proxy references. Translation between proxy and domestic references is performed by the kernel automatically, thus, repositories do not need to be aware of proxy references (although, they can, if they need to).

The clients may assume that there is only one (multi-packaged) repository like UNIX programs assume there is only one file system. When the kernel processes

an RAAPI operation, the following simple algorithm is used to determine, in which repository the changes have to be stored:

- if all the elements involved are from the same repository, the kernel forwards the call to that repository through its adapter;
- if the elements involved are from different repositories (e.g., an association between two classes from different repositories is being created), the kernel treats it as an inter-repository change and stores it in the pivot repository.

The kernel stores information about the kernel package tree, mounted repositories and inter-repository relations according to Kernel Metamodel depicted in Figure 1(b). The kernel performs only minimal constraint checking for inter-repository relations. More sophisticated constraint checking can be performed by introducing an additional layer over RAAPI, or by means of external model transformations.

## 3 Repository Access API (RAAPI)

Subsections 3.1 and 3.2 introduce notions that are essential for RAAPI. RAAPI itself is explained in Subsection 3.3.

### 3.1 Linguistic vs. Ontological Meta-levels

Atkinson and Kühne [21,32,31] noticed that actually there are two types of meta-levels — linguistic and ontological. All the three levels M1-M3 from the three-level conjecture are **linguistic meta-levels**, since the meta-metamodel from Level M3 can be considered a language for specifying M2 metamodels, and an M2 metamodel can be considered a language for specifying M1 models. However, RDF and OWL Full meta-metamodel from RDF/OWL TS as well as the meta-metamodel of the JR repository permit describing classes and their instances (individuals) at the same linguistic level M2. Moreover, a class may act also as an instance. This makes it possible to create multiple **ontological meta-levels** within M2 according to the class-instance relationship defined at M3. In this case M1 is not needed. If a repository supports that, elements from different ontological meta-levels can be mixed, e.g., an individual can be linked to a class.

### 3.2 Šostaks' Conjecture

While working on a generator for higher order model transformations[5], where multiple meta-levels are involved, A.Šostaks came to the following conclusion (we refer to it as **Šostaks' conjecture**[6]):

[5] These are model transformations that generate other model transformations.
[6] Personal communication with Agris Šostaks, June 2012.

> *It is difficult for a human to think at more than two meta-levels at a time. Still, it is pretty easy for a human to focus on any two adjacent meta-levels.*

This conjecture also reveals itself in Java. All Java classes along with their instances (objects) span two meta-levels. Java has the *Object* class, which is a common superclass for all other classes — an easy-to-understand OOP construction, which remains within the two meta-levels. At the same time, Java has the reflection mechanism that permits considering Java classes as objects. The class named *Class* has been introduced for that. Actually, it is a meta-class (i.e., it brings an additional meta-level to Java), but represented as an ordinary Java class. At this point those, who are studying Java, encounter difficulties, since three meta-levels are involved.[7]

### 3.3 RAAPI Explained

Taking into a consideration Šostaks' conjecture, RAAPI consists of functions, which can access two adjacent meta-levels (either linguistic, or ontological). Switching between meta-levels can be performed by passing a reference to an element at one meta-level to an RAAPI function, which expects a reference to an element at another meta-level. For instance, a reference to a class may be passed to an RAAPI operation, which expects a reference to an object. This trick can also be used to mix multiple meta-levels (where the underlying repository supports that).

Although the detailed description of RAAPI goes beyond the scope of this paper (the current version of RAAPI can be found at `http://tda.lumii.lv/raapi.html`), below we provide some points shedding the light on RAAPI.

- RAAPI is able to read at least ECore and MOF meta-metamodels and their linguistic instances. This is sufficient to access also all other layers (including ontological) by means of the meta-level switching trick from above.
- RAAPI supports multiple classification and dynamic reclassification (i.e., one object can belong to many classes). Multiple inheritance is supported as well.
- Every repository adapter has to support at least two meta-levels: if ontological meta-levels are supported, then these are two ontological meta-levels within M2; otherwise, these are linguistic meta-levels M2 and M1. In case of RDF and OWL Full, the adapter should support multiple ontological meta-levels. Support for a linguistic meta-metamodel (Level M3) is not required, unless there is a need to access technical-space-specific features defined at M3.
- To create an adapter implementing RAAPI for a particular repository, only essential functions have to be implemented. For instance, the given repository may be able to iterate either through proper attributes of a class, or trough all attributes (including derived). An adapter may implement only one of these

[7] Personal communication with Edgars Celms, June 2012.

cases, and the kernel will implement the other. Also, if the given repository does not support some capability (e.g., support for enumerations or support for multiple classification), the adapter may either simulate it, or to discard it. The latter case resembles how certain file system operations are discarded for some file systems in UNIX (e.g., when UNIX-style file permissions are not supported on a mounted file system).
– RAAPI is technically simple, thus, it can be easily adapted to and used from different platforms and programming languages, facilitating the integration with different technical spaces.

## 4 Manipulating the Packages

Instead of forwarding RAAPI calls to a particular physical repository, a repository adapter can also represent a *computable* (or *virtual*) *repository*, which does not exist physically, but relies on the data from other repositories. When a virtual repository is mounted into some kernel package, its repository adapter transforms RAAPI operations to operations on other kernel packages on-the-fly. We can say that such a repository adapter implements an *on-the-fly model transformation*. On-the-fly model transformations can be used to implement certain manipulations with packages. Here are some examples.

*Virtual Copy.* Having an existing package $P$, the "virtual copy" manipulation creates a virtual package $P'$, which acts as a copy of $P$. However, in reality, no data is copied! This manipulation can be implemented by mounting two special virtual repositories into $P$ and $P'$, but keeping pointer to the old package $P$. When the old data is accessed, both repositories forward the call to the old $P$. At the same time, both virtual repositories record changes in new $P$ and $P'$, and provide an illusion that $P$ and $P'$ are being modified independently. One of the use cases for "virtual copy" is implementing transactions. First, a virtual copy of a package is created to fix its state without deep copying of the data. Then, some transactional changes are performed, and, finally, these changes are either discarded, or stored in the old $P$. Another use case is providing the space for semantic reasoning. If a package contains an ontology, its virtual copy can be created, and a semantic reasoner can be launched on that copy. When the reasoner finishes, the original package and its copy will contain the data before and the data after the reasoning, respectively.

*Virtual Merge.* A virtual repository can be mounted into an existing package $P$, keeping a pointer to the old $P$ and a pointer to some other package $Q$. The repository provides an illusion that $P$ and $Q$ are merged (in the sense of UML package merge).

*Introducing derived (calculated) properties and relations.* To create a calculated relation between classes $A$ and $B$ from a package $P$, another virtual repository is introduced. This repository also contains classes $A$ and $B$, but also adds the required calculated association, which is computed on-the-fly when this virtual repository is accessed via RAAPI. To complete the picture, the repository has

to be mounted into some package $Q$, and then $P$ and $Q$ have to be virtually merged.

*Symbolic links.* Although the kernel package tree is a tree, packages can be organized into a graph-like structure by means of UNIX-style symbolic links. A symbolic link $L$ on package $P$ may be implemented by introducing a virtual repository, which is mounted into $L$, but forwards all RAAPI calls to $P$.

*Using volatile temporary data in models.* A volatile temporary repository, whose content is lost on exit, may be introduced. Temporary data can be transparently combined with the persistent data by means of "virtual merge".

*Indexing of model elements.* Certain calculated relations can just reorder the elements from the relations they rely on. Thus, when traversing the corresponding calculated element list, elements will appear in the desired order. To implement this behaviour, the virtual repository can use indexes internally.

*Views on metamodels.* One metamodel (say, a complex one) can be represented as another metamodel (e.g., simpler) by means of views. Like file systems can be read-write and read-only (e.g., CD-ROM), views can be read-write (implementing bi-directional on-the-fly transformations) and read-only (implementing unidirectional on-the-fly transformations). Read-only views simply discard modificating RAAPI operations.

## 5 Related Work

The need for a common API for accessing different types of repositories has already been realized by some teams. For example, ATL Virtual Machine [2], Epsilon Model Connectivity level (EML) [33,6], and the CDO [3] repository use some kind of common API, which plays the same role as RAAPI in the proposed multi-repository mechanism. In contrast to ATL and EML (which use a few API functions being able to work with lists) as well as ECore (which uses object-oriented API), RAAPI is procedural and uses only primitive data types. Also, RAAPI was designed to support SCMOF capabilities.

The megamodel concept (a terminal model, which stores information about other models) introduced by the ATLAS/AMMA team resembles how the kernel stores information about the mounted repositories and the relations between them in the pivot repository [24,23].

Repository adapters used by the kernel resemble how ModelBus uses tool adapters to connect multiple modelling tools [28,10]. In ModelBus, adapters are mainly used to access the data according to the check-in/check-out principle, while RAAPI adapters are intended to perform their functions on-the-fly (especially, when virtual repositories are used). To implement repository adapters for certain technical spaces, numerous existing technologies, such as ORM-technologies (Java Persistence API, .NET Persistence API), D2RQ [22], object-oriented databases [11], etc. can be utilized. If on-the-fly data access is impossible, the check-in/check-out principle can be used as well.

Certain research on model merge is being performed, but in a different context than the proposed "virtual merge" operation. For instance, Epsilon Merge Language is an excellent language intended for describing merge-like operations on models, which are then later executed in an offline mode (not on-the-fly) [33,6]. MOF 2 for Java implements the merge capability, but the goal was MOF 2 compliance, not merging different repositories [1].

The live model transformation framework proposed by the VIATRA team treats complex model changes as elementary changes [40,19]. This resembles on-the-fly transformations, which could be considered split into a set of elementary RAAPI operations.

The "virtual copy" operation is based on the concept of worlds, which is a way to control side effects arising of using the same data from different parts of the program [42].

Interesting ideas about read-only views have been presented by E. Rencis [39]. His mechanism modifies the code of a model transformation in such a way that the view is executed on-the-fly. On-the-fly transformations mentioned in this paper are intended to perform the same job, but without modifying client code (thus, on-the-fly transformations are not tied to a particular transformation language, but only to RAAPI). Ideas and code fragments provided by E. Rencis can be adapted to generate code for virtual repositories implementing views through RAAPI.

## 6 Conclusion

The implementation of the proposed multi-repository mechanism is in progress. While the development of repository adapters is straightforward, the development of the multi-repository mechanism itself is technically difficult: we need to consider numerous details (e.g., distinguish between ordinary and inter-repository links), while keeping the overhead low. For further development of the multi-repository mechanism, refer to the web-page `http://tda.lumii.lv`.

We strongly believe that like in a symphony orchestra different musical instruments, each with its timbre and specific ways to get certain tones and overtones, playing together, are able to perform a rich sounding symphony, multiple model repositories, each with its specific features and application domains, can enrich the capabilities of model-driven software by being used simultaneously and in harmony with each other.

**Acknowledgements.** I gratefully acknowledge E. Kalniņa and S. Rikačovs for the ability to discuss with them certain repositories mentioned in Table 1. The work has been partially supported by Latvian National Research Programme №2 "Development of Innovative Multifunctional Materials, Signal Processing and Information Technologies for Competitive Science Intensive Products" (Project №5 "New Information Technologies Based on Ontologies and Model Transformations"). Certain results of this work have been obtained with the support of the European Social Fund within the project "Support for Doctoral Studies at University of Latvia".

## References


1. A Meta-Modelling Technology for CMOF-based Models, http://www.webgambas.com/metabubble/amof.html
2. ATL – a model transformation technology, an Eclipse Model-to-Model Transformation project, http://www.eclipse.org/atl/
3. The CDO model repository, http://www.eclipse.org/cdo/
4. Eclipse Modeling Framework (EMF, Eclipse Modeling subproject), http://www.eclipse.org/emf
5. Enhanced Model Repository, http://modelbased.net/aif/solutions/singular_solutions/solution_mof_repository.html
6. Epsilon, an Eclipse Model-to-Model Transformation project, http://www.eclipse.org/epsilon/
7. Information technology – Information Resource Dictionary System (IRDS) Services Interface (iso/iec 10728:1993 standard), http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=18821
8. The Java Graph Laboratory (JGraLab) project, http://www.ohloh.net/p/jgralab
9. MetaEdit+, http://www.metacase.com
10. ModelBus homepage, http://www.modelbus.org/modelbus/
11. Object Database Management Systems, http://www.odbms.org/
12. OWL 2 Web Ontology Language document overview, http://www.w3.org/TR/owl2-overview/
13. OWL 2 Web Ontology Language profiles, http://www.w3.org/TR/owl2-profiles/
14. OWL Web Ontology Language reference, http://www.w3.org/TR/owl-ref/
15. OWLIM semantic repository, http://www.ontotext.com/owlim/
16. Resource Description Framework (a suite of W3C Recommendations), http://www.w3.org/RDF/
17. Sesame home page, http://www.openrdf.org/
18. The InfoLibrarian Universal MetaMart Metadata Repository, http://infolibcorp.com/Metadata%20Repository.html
19. VIATRA2 (VIsual Automated model TRAnsformations) framework, an Eclipse Model-to-Model Transformation project, http://www.eclipse.org/gmt/VIATRA2/
20. Virtuoso open-source edition, http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/
21. Atkinson, C., Kühne, T.: Model-Driven Development: A metamodeling foundation. IEEE Software 20(5), 36–41 (2003)
22. Bizer, C., Seaborne, A.: D2RQ – treating non-RDF databases as virtual RDF graphs. In: Proceedings of the 3rd International Semantic Web Conference, ISWC 2004 (2004)
23. Bézivin, J., Jouault, F., Rosenthal, P., Valduriez, P.: The AMMA platform support for modeling in the large and modeling in the small, Tech. rep., LINA, Universite de Nantes (2005)
24. Bézivin, J., Jouault, F., Valduriez, P.: On the Need for Megamodels. In: Proc. of Workshop on Best Practices for Model-Driven Software Development at the 19th OOPSLA, Vancouver, British Columbia, Canada (2004)
25. Bézivin, J., Kurtev, I.: Model-based technology integration with the technical space concept. In: Proceedings of the Metainformatics Symposium (2005)

26. Cook, S., Jones, G., Kent, S., Wills, A.: Domain-Specific Development with Visual Studio DSL Tools. Addison-Wesley (2007)
27. Flatscher, R.G.: An overview of the architecture of EIA's CASE Data Interchange Format (CDIF), http://wi.wu-wien.ac.at/rgf/9606mobi.html
28. Hein, C., Ritter, T., Wagner, M.: Model-driven tool integration with ModelBus. In: Proceedings of Future Trends of Model-Driven Development Workshop (2009)
29. Hofstadter, D.: Gödel, Escher, Bach: an eternal golden braid. Harvester Press Ltd. (1979)
30. Kelly, S.: The model repository: More than just XML under version control. In: OOPSLA DSM Workshop 2008 Keynote (2008)
31. Kühne, T.: Clarifying matters of (meta-) modeling: an author's reply. Software and Systems Modeling 5, 395–401 (2006)
32. Kühne, T.: Matters of (meta-) modeling. Software and Systems Modeling 5, 369–385 (2006)
33. Kolovos, D., Rose, L., Paige, R.: The Epsilon Book, http://www.eclipse.org/epsilon/doc/book/
34. Kurtev, I., Bézivin, J., Aksit, M.: Technological spaces: An initial appraisal. In: CoopIS, DOA 2002 Federated Conferences, Industrial track (2002)
35. Matula, M.: NetBeans Metadata Repository, https://netbeans-uml-extender-plugin.googlecode.com/files/MDR-whitepaper.pdf
36. Object Management Group: MOF Support For Semantic Structures (SMOF), "in process" version of SMOF, http://www.omg.org/spec/SMOF/ (available for OMG members)
37. Object Management Group: OMG Meta Object Facility (MOF) Core Specification Version 2.4.1, formal/2011-08-07
38. Opmanis, M., Čerāns, K.: Multilevel data repository for ontological and meta-modeling. In: Databases and Information Systems VI - Selected Papers from the Ninth International Baltic Conference, DB&IS 2010 (2011)
39. Rencis, E.: On views on metamodels. In: Databases and Information Systems VI - Selected Papers from the Ninth International Baltic Conference, DB&IS 2010 (2011)
40. Ráth, I., Bergmann, G., Ökrös, A., Varró, D.: Live Model Transformations Driven by Incremental Pattern Matching. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) ICMT 2008. LNCS, vol. 5063, pp. 107–121. Springer, Heidelberg (2008)
41. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework, 2nd edn. Addison-Wesley (2008)
42. Warth, A., Ohshima, Y., Kaehler, T., Kay, A.: Worlds: Controlling the scope of side effects. Tech. rep., Viewpoints Research Institute (2010)

# An Ontology-Driven Fuzzy Workflow System

Václav Slavíček

University of Hradec Kralové, Faculty of Informatics and Management,
Rokitanského 62, 500 03 Hradec Králové, Czech Republic
vaclav.slavicek@uhk.cz

**Abstract.** In this paper, we present a methodology for integrating workflow-driven information system with an ontology-based knowledge repository, where both the above specified systems are built upon the fuzzy set theory, and hence capable of dealing with vagueness and uncertainty. Furthermore, we introduce some workflow activities that use fuzzy logic for human-friendly communication with business process participants. We also present a technique of workflow execution log analysis, and subsequent fuzzy ontology reasoning to provide an automated feedback to the workflow, and thus to achieve its evolution. We also report on prototypical implementations, namely a component for hybrid integration of fuzzy ontology with the information system, and a component for building fuzzy workflows in a workflow designer.

**Keywords:** workflow, ontology, fuzzy, knowledge.

## 1 Introduction

With the breakthrough of information technology into industry and commerce, we can notice the trend of supporting various business processes by underlying information systems. This support is referred to as workflow management, and with no doubt makes the processes more efficient. However, knowledge-intensive or unstructured processes are sometimes hard to be supported by standard workflow management techniques. In the past, there have been numerous proposals to handle these challenges with goal-driven, flexible, evolutionary, or fuzzy workflows, as well as by close integration of the workflow management with organizational knowledge. Yet, these approaches have not been widely adopted by the IT industry because of their overall complexity, commercial impracticability, or just partial nature of the solution. By other words, there is a lack of easy-to-implement, holistic solution which would combine a flexible workflow management system with a knowledge-representation capable of handling imprecision and uncertainty similar to human way of thinking and reasoning.

In this work, we present a methodology which integrates an ordinary information system with a workflow management system as well as with ontology, introducing vagueness in all these elements by means of fuzzy logic. More precisely, we propose to extend the information system with a software component facilitating operations over fuzzy sets and relations. The same component can be used in the workflow man-agement system to enable fuzzy conditions in the flow control. We claim that the communication between workflow management system and human participants can be

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 515–527, 2013.

supported by fuzzy set theory, and thus reflect the imprecise nature of human reasoning and decision-making.

This paper recalls background knowledge on object-oriented programming, workflow management, fuzzy logic, description logic, and ontology language OWL2. The rest of this paper is organized as follows: Section 2 outlines the background and reports on related work. In Section 3, we introduce the methodology which integrates workflow and ontology modelling with respect to their vague nature. Section 4 reports on prototypical implementations of some components required for evalution of the methodology in practice. Section 5 sets out conclusions and ideas for future research.

## 2 Background and Related Work

### 2.1 Workflow Management Systems

*Workflow* is the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another to perform a specific action, according to a set of procedural rules [1]. A workflow can (but may not be) part of a business process. *Workflow Management System* (WfMS) is a set of tools providing support for definition, enactment, administration and monitoring of workflows [1]. The enactment involves both direct invocation of computer systems as well as indirect interaction with human participants via a task distribution and assignment. There exist several ways of how an ordinary information system can be enriched with some workflow management functionalities, ranging from dedicated information systems with workflow management functionality to versatile business process management suites.

Workflow flexibility is defined as an ability of the workflow system to quickly react to: i) incorrect or imprecise model, ii) exceptional or rare situation, iii) alteration of organizational structure as well as on any other change in data structure, iv) continuously changing process goals [2]. A continuous change of workflow definition during its execution is also referred to as workflow evolution [3,4], and is currently sup-ported by vast majority of commercial workflow systems. One example of area re-quiring flexible workflows are knowledge-intensive processes, which are goal-driven, less structured, with less predictable transitions, and in fact contradictory to classical, rigidly recurring process flows.

### 2.2 Enhancing Workflow Management with Knowledge

While process maps represent a sort of externalized knowledge, another kind of knowledge is exposed outside the boundaries of the WfMS. It is sometimes codified in standard information systems. On other occasions, it is present only in minds of the employees. According to [5], examples of information desirable during workflow execution are flexible models of organizational structure (people, roles, skills, teams, tasks), patterns of interaction between team members, and models of general tasks with assigned document templates.

Audit trails of workflow enactment, sometimes referred to as workflow execution logs, are also part of organizational memory and may help in organizational learning

[6]. Organizational memory has the potential to enhance decision-making within workflow junctions. A concept of WfMS and Organizational Memory Information System (OMIS) integration described in [7] requires interlaced workflow design and enactment. Organizational learning begins at the workflow execution phase. The execution outcome is repeatedly stored in OMIS to improve quality and effectiveness of the workflow model.

### 2.3 Ontology Modeling

One way of capturing and storing knowledge in a structured and machine-interpretable way is the use of ontology engineering. Ontology is a concrete form of a conceptualisation of communitys knowledge in a specific domain. Ontology includes a vocabulary of terms together with a specification of their meaning. An inference mechanism implemented in reasoners allows eliciting additional implicit knowledge from the explicitly declared facts. OWL 2 is nowadays a popular language for authoring ontologies. There are numerous OWL-supporting ontology editors available, such as open-source Protg, as well as various reasoners like FaCT++, HermiT or Pellet. Formal semantics of OWL 2 is based on Description Logic (DL), a subset of First Order Logic, of type *SROIQ(D)*. OWL comes with three variants (sublanguages), ordered by increasing expressiveness as OWL Lite, OWL DL and OWL Full.

### 2.4 Integration of Object, Workflow, and Ontology Model

A connection between workflow model and object model has so far received de-tailed study, and is thoroughly described in a literature [8,9].

Speaking about workflow-ontology integration, term *workflow ontology* denotes ontology in service of WfMS. Such ontology is designed to store the following information: i) presupposition rules to traverse unavailable information, ii) consistency rules to assess the above presuppositions iii) semantic proximity properties to help finding alternative subworkflows, resources or users if the defaults are not available, iv) semantic rules to evaluate the proximity properties in a particular execution con-text [10].

Regarding the object-ontology integration, a comparison of ontology and object-oriented modeling reveals handful of similarities. Indeed, integration of ontology with mainstream object-oriented programming languages (OOPL) like C# or Java, sug-gests itself quite obviously. There are, however, also several differences, such as an absence of multiple inheritance, strict object conformance and type-safety in OOPLs, and, on the other hand, static nature of ontologies. In general, there are two types of object-oriented modeling: i) direct (traditional) models use classes and instances to represent concepts and individuals of the real world, respectively. ii) Indirect models keep both concepts and individuals in OOPL instances, whereas OOPL classes represent real world meta-concepts [11]. Similar to these models, one can speak about direct or indirect OOPL-ontology integration, depending if ontology concepts map OOPL classes or OOPL instances, respectively. There is also a third option: hybrid integration as a combination of direct and indirect integration [12].

### 2.5 Fuzzy Logic

Fuzzy set theory [13] was introduced by L.A. Zadeh in the 1960s to represent and manipulate vague, unclear, ambiguous, imprecise, noisy, or missing data. Unlike approaches based on the first order logic and probability theory which do not provide framework for dealing with imprecise and non-categorial knowledge, fuzzy logic facilitates approximate human reasoning analogous to human cognitive processes. In classical set theory, the membership of an element in a set is strictly described by $true(1)$ or $false(0)$. Fuzzy set theory moreover accepts other values between 0 and 1. Furthermore, it is possible to assign linguistic variables together with linguistic terms to a specific fuzzy membership functions to help formulate logical statements in natural language. Alike the traditional, two-valued logic, fuzzy logic defines several operators on fuzzy sets.

Fuzzy logic has definitely found its application in various knowledge based sys-tems in business, finance, and management, yet with a very limited adoption in the field of workflow modeling and management. Still, business processes are often steered by soft human preferences, which are way from being formal, structured or determinable. Hence WfMS can certainly benefit from fuzzy conditions when split-ting the flow into several branches based on a set of procedural rules.

### 2.6 Application of Fuzzy Set Theory in the above Discussed Domains

**Uncertainty and Imprecision in Workflow Management.** In the design phase, it sometimes proved hard for workflow designers to define a promising workflow model. Hence, it could be convenient to design the processes during its execution or even ex post. Concept of *weak workflows* [14] allows building the workflow design simultaneously with its enactment. With this concept, it is possible to model processes with incomplete information. Weak workflows provide for interlaced design and enactment - initial abstract model is being gradually refined. Examples of implementations are projects FRODO [15] or DeFlex [16].

Aslo workflows based on *progressive model* [17] do not need any detailed workflow model in advance. It uses the enactment phase to record data inserted step-by-step by users, as well as additional information about these steps. This information is then available as a template for analogous processes in the future. Other similar approaches are Retroactive workflows [18], and Teleo-Reactive workflows [19].

The *fuzzy business process management* approach [20][21] translates the vague-ness and ambiguity of human thinking into workflow automation by extending Event-Driven Process Chain (EPC) modeling notation with fuzzy rule sets. These rule sets are afterwards used in the decisions about future process-flow. The discussed approach can even be enhanced with a sort of organizational learning, provided that the fuzzy rules are generated automatically out from statistical evaluations [22]. Analysis of workflow audit trail with the help of fuzzy logic is also presented in [23].

In [24], *Autonomic Object (AO) Intelligence Algorithm* based on *Extensive Mamdani Fuzzy Reasoning System* is used to enhance workflow flexibility by employment of fuzzy reasoning. AO intelligence and autonomic computing is used for the workflow model updates.

**Fuzzy Logic in Ontology Modeling.** Classical ontology languages are not appropriate for dealing with imprecision or vagueness in knowledge. Therefore, Description Logics (DL) have been enhanced by various approaches to handle probabilistic uncertainty, possibilistic uncertainty, and vagueness.

Regarding handling of vagueness, popular is the fuzzy generalization of *SHOIN(D)*, so called fuzzy *SHOIN(D)*. With this generalization, concrete domains (data types) are represented by fuzzy sets. It also introduces fuzzy modifiers, fuzzy axioms, fuzzy *RBoxes*, fuzzy *TBoxes*, and fuzzy *ABoxes*. Besides the research on the theoretical framework, number of fuzzy DL reasoners has also been implemented. Examples are *FuzzyDL* [25] [26] and *DeLorean* [27].

## 3 Ontology-Driven Fuzzy Workflows

We propose a methodology which integrates fuzzy workflows with fuzzy ontology, to design a WfMS that meets the following requirements:

- is goal-driven rather than process driven,
- is flexible, i.e. can divert from a predefined workflow
- capitalizes on the knowledge of the organization,
- deals with imprecise and vague information,
- boosts initiative, and cooperation of human participants,
- provides feedback and so contributes to the workflow evolution and improvement.

The overall methodology can be decomposed into the following steps, as also illustrated by Figure 1:

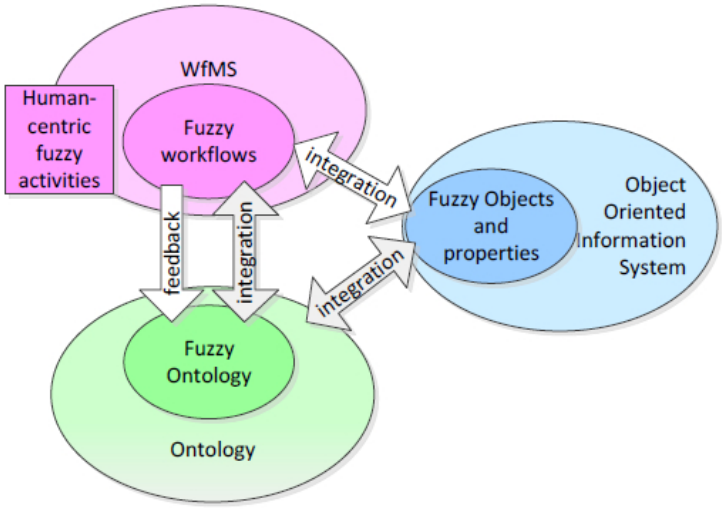



**Fig. 1.** Schema of Ontology-Based Fuzzy Workflows

– Introduction of **fuzzy logic** support to an information system (IS), including its WfMS component, as well as to the ontology-based knowledge repository,
– introduction of **human-centric workflow activities** (i.e. basic workflow building blocks) that are aware of human-nature of the business process participants,
– **integration of the WfMS with the IS and with the fuzzy ontology**, so that the workflow conditions can be evaluated both against the IS, and the ontology-based knowledge,
– provision of **feedback from the workflow execution log** back to the ontology.

### 3.1 Fuzzy WfMS

The application of fuzzy logic in WfMS brings the following advantages:

- Classical, crisp set theory is a special case of fuzzy theory. Thanks to this fact, even a fuzzy workflow can be subject of *strict limits and boundaries*. On the other hand, fuzzy workflow can flexibly react to the *vague reality*. Example is an internal regulation to process all tasks within 24 hrs. Still, it often does not mean that the customer would be totally satisfied with 23hrs, and disappointed after 25hrs duration. Hence the resolution time is a suitable candidate for being represented by a fuzzy number.
- In accordance to the existing concept of weak and retroactive workflows, unclear workflow transitions can be left unspecified in the design phase. Taking it to the extreme, any workflow can be modelled as a complete graph of activities at the first stage, where every pair of activites is connected by an edge, with its subsequent reduction.

### 3.2 Human-Centric Workflow Activities

With the help of fuzzy logic, it is feasible to design workflow activities that are aware of human-nature of the process participants. Both for the information flow from a human to WfMS (E. g. processing of human decisions, authorizing approvals) and backwards (E. g. task distribution, message announcement). For the task assignment, for example, we propose to enhance the ordinary workflow task distribution list with a rich formatting. As shown in Figure 2, tasks are distinguished by various font colours and sizes.

| TaskDesc | CaseType | CaseDesc | InitDate |
|---|---|---|---|
| Solve the incident | Incident | User cannot logon | 22/08/2011 14:22 |
| Configure new port in HAM | MR | CZ ACM Add Port | 23/08/2011 08:23 |
| Configure new virt. item in HAM | MR | CZ ACM New Config | 24/08/2011 16:07 |
| Unlock user account in HAM | CAR | User Acc. Unlock | 24/08/2011 16:33 |
| Prepare custom report | CAR | Report | 24/08/2011 17:17 |

**Fig. 2.** Example of fuzzy task distribution list

This way, we can represent their membership degree in two distinct fuzzy sets. The formatting can reflect task semantic proximity, urgency, impact, etc.

### 3.3 Integration of Object-Oriented, Workflow and Ontological Model

We eventually deal with three different paradigms. Table 1 compares various features of ontological modelling, object-oriented modelling, and workflow modelling.

**Table 1.** Comparison of ontology, object-oriented and workflow modeling

| Feature / Modelling | Static Structures (concepts, individuals, data and object properties, roles) | Dynamic Behaviour (functions and methods, dynamically changing static structures) | System-User interaction (dynamic behaviour in long time transactions) |
|---|---|---|---|
| **Ontology** | **Yes** | **No** | **No** |
| **Object-oriented** | **Yes** | **Yes** | **Limited** (hard to design and maintain, requires persistent storage like DBMS) |
| **Workflows** | **Yes** (assuming access to business or ontology layer) | **Yes** (WF activities per se, or by OO methods invocation) | **Yes** (easy to manage, built-in persistent storage) |

**OOPL-Ontology Integration.** As shown in Figure 3, we propose to use of hybrid integration, where some OWL concepts are mapped 1:1 to predefined OOPL classes, wheres others are mapped N:N.

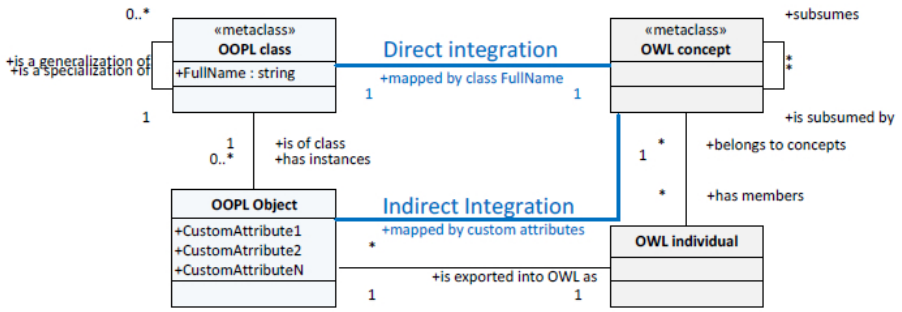



**Fig. 3.** UML metaclass diagram of hybrid OOPL-ontology integration as implemented in Fuzzy Ontology Framework

For direct mapping, we can specify that every OOPL object class C belongs to specific OWL concept by defining the concept as the one whose individuals have property *ClassFullName = C.FullName*. On the other hand, we can have several OOPL classes mapped to a single OWL concept and vice versa. To reveal such a relationship in the IS, we can request a fuzzy set of OWL concepts for specific individual, or the other way around a fuzzy set of individuals for a specific concept.

**WfMS-Ontology Integration.** The main purpose of the WfMS-ontology integration is to evaluate semantic rules defined in the ontology in order to find out proximity of particular individuals towards a specific concept. We typically know that an individual belongs to a basic concept like *Employee or CustomerOrder*. Yet we need to evaluate its proximity to several other sub-concepts like *SalesPerson* or *HighValueCustomerOrder*. This in turn influences the workflow execution.

Lets take an example of OWL concept Employee that is linked to an appropriate C# class and to a database table of the same name. Of course, there can be several subsumed concepts like *CheapSalesPerson*, *CapableSalesPerson*, or *CheapAndCapableSalesPerson*. Axioms defining these subconcepts can alter during the course of time, and so we want to keep the concepts neither hard-coded in the OOPL nor in a relational database. We need more flexible way to define what a low commission or a high sales amount is, as well as how the particular properties interact. This can be achieved by means of the fuzzy ontology, and its hybrid integration with fuzzy IS. For the above purposes, we need to be able to express two kinds of fuzzy axioms in the ontology:

- *Concept subsumption* is a statement type *TBox* of form $C \sqsubseteq D > \alpha$, denoting $inf_{x \in \triangle I} \{C(x) \Rightarrow D(x)\} > \alpha$. Example is a statement stating that $C_s \sqsubseteq C_e > 0.8$, where $C_e$ is a concept representing employees, $C_s$ represents sales persons, and $\triangle I$ is an interpretation domain. It declares that every employee is at least a 0.8 sales person.
- *Instance-concept affiliation* is an *ABox* type statement of form $C(a) > \alpha$, where $C$ denotes a fuzzy concept, $a$ is an individual, and $\alpha$ is a degree of membership. $C(a)$ tells us to what extent the individual a can be considered an element of fuzzy concept $C$. If $C$ stays for a concept representing managers, for example, then every employee $a$ belongs to this concept to degree $\alpha$ from the interval $< 0, 1 >$.

### 3.4 Workflow Audit Trail Analysis and Feedback

The proposed methodology is also provides for a feedback from the workflow execution log back to the ontology. Since the flow control is in turn influenced by knowledge stored in the ontology, this mechanism can contribute to the workflow evolution.

Figure 4 illustrates typical integration of an information system with WfMS. We propose a data transfer from the WfMS execution log to the main relational database (highlighted arrow in Figure 4). More precisely, objects in the main database which have been involved in the processing of workflow instances will be enhanced with statistical information about their activities which took place in past. Example is a number of processed customer orders per employee. Once there are raw statistical data available for data objects, the rest of the analysis can be easily accomplished as a part of fuzzy ontology reasoning. We can easily declare a concept *ExperiencedEmployee as*

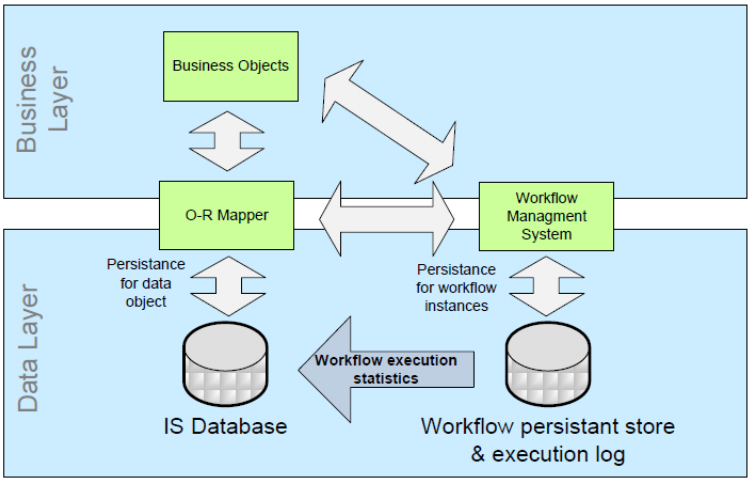



**Fig. 4.** Transfer of workflow execution statistics from WF execution log to the data model of the IS

*Employee and highNumberOfProcessedOrders*, for example, where *highNumberOfProcessedOrders* is a right-shoulder fuzzy set. If there are order processing tasks assigned to concept ExperiencedEmployee in a workflow defi-nition, then the higher number of orders processed by a specific employee in past, the higher degree of assignment of new tasks in his/her task list.

## 4 Prototype

We also report on some specific implementations related to the methodology introduced in the previous section.

### 4.1 Fuzzy Framework

FuzzyFramework[1] is a library facilitating evaluation of fuzzy logic expressions. It is feasible to implement it into arbitrary existing information systems built upon the Microsoft .NET platform. There is a support both for continuous and discrete fuzzy sets. It works with any fuzzy sets as long as we can describe them by a group of polynomial functions. Last but not least, there is no special parser required in the .NET environment, since the library overloads standard .NET operators to work with fuzzy relations. This facilitates an incorporation of the library into Windows Workflow Foundation, a Microsoft .NET based workflow management engine.

As apparent from the example in Figure 5, the prototype also offers a graphical representation of fuzzy sets and hierarchical representation of fuzzy relations. This is intended to help users get quickly familiar with the internal structure of fuzzy expressions they deal with in the workflow conditions.

[1] http://www.codeproject.com/KB/library/Fuzzy-Framework.aspx

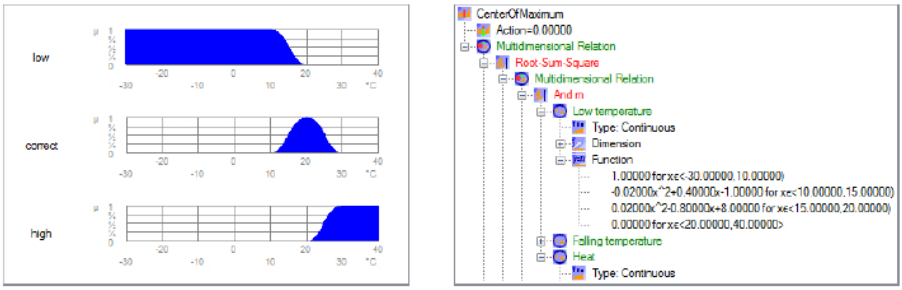



**Fig. 5.** Example of graphical representation of fuzzy sets and relations in Fuzzy Framework

### 4.2 Fuzzy WfMS

We have chosen to use Windows Workflow Foundation (WWF) ver. 4.0 for the concrete prototypical implementation of fuzzy workflows. The main reason is that WWF workflow designer features a native VB.NET parser. The parser is available in all control-flow as well as data manipulation activities, enabling us to define arbitrary fuzzy expressions directly in the designer, including a defuzzification. Figure 6 illustrates a definition of fuzzy expression within an *If-Else* activity condition.

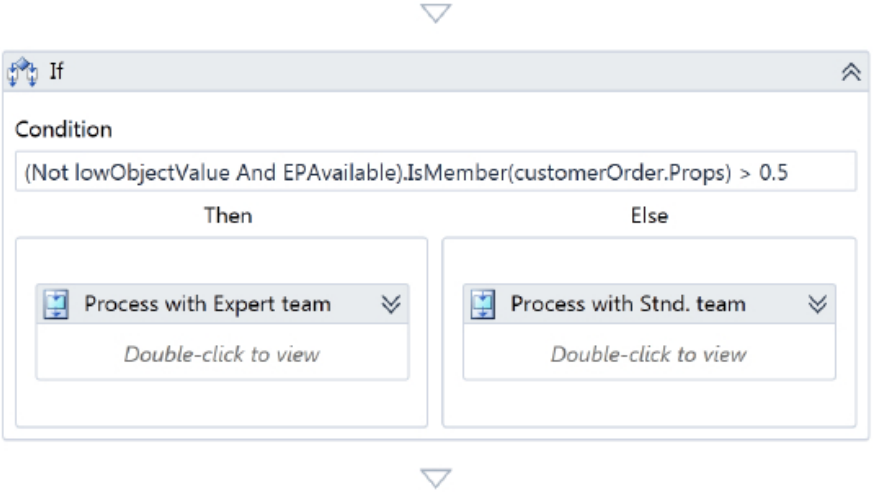



**Fig. 6.** The *If-Else* activity executes one of its children based on the result of fuzzy expression

### 4.3 Fuzzy Ontology Framework

To represent the fuzzy ontology, we have decided to re-use exisitng methodology *Fuzzy-OWL2* introduced in [28]. This approach implements the vagueness into standard OWL2 ontology by means of native OWL2 annotation properties.

Regarding the integration of fuzzy ontology with fuzzy IS, we have proposed our own methodology, and implemented a prototype called *Fuzzy Ontology Framework*[2] on platform Microsoft .NET. In a nuthshell, the component loads a collection of OOPL instances from the business layer, and stores them as individuals in the OWL ontology. Based on the member (i.e. field or property) values of these instances, and the specified ontology, a fuzzy reasoner can subsequently assign the individuals (i.e. instances) to one or more OWL concepts. At present, only reasoner *FuzzyDL* [25] is supported. The concept affiliation is returned back to the business layer, which can in turn use this information at its discretion. It is typically used in workflow procedural rules to determine the future process flow. More precisely, most of instances are internally stored in memory rather than in the ontology. The reason is that several thousand individuals would hamper manipulation with the source ontology file.

## 5 Conclusion and Future Work

In this paper, we proposed a methodology that integrates a fuzzy workflow management system with a fuzzy ontology knowledge representation. Unlike the existing related work, this approach uses a holistic view on fuzzy workflow management within an organization, including its mutual cooperation with other data and knowledge repositories. Furthermore, in accordance to the fuzzy nature of the solution, we propose a number of workflow activities focused on the computer  human interaction.

The solution is easy-to-implement to existing information systems. We claim that an adoption of the proposed methodology by IT industry can positively impact the flexibility of business processes supported by the systems, as well as the motivation of human participants to actively cooperate on the workflow evolution.

Fuzzy logic with its linguistic variables gives the workflow designers a powerful tool for modeling complex flow conditions with ease, still obtaining significant results to assure an effective process execution. On the other hand, the designers have to keep in mind that vague workflow model can potentially hamper the establishment of clear ultimate responsibility for particular workflow tasks.

The natural direction for future work lies in extending the prototype to wider range of workflow management systems, knowledge-representations, and reasoners. This will enable an evaluation of the methodology in several practical use cases. Regarding the methodology itself, it can be further extended with a KPI-based reward system, in order to motivate the human workflow participants to pick the most benefical tasks out of the task list. Not only beneficial from the perspective of the business process, but also to satisfy the participants themselves in a longer perspective.

## References

1. WfMC: Workflow Management Coalition Terminology & Glossary (1999), http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf
2. Narendra, N.C.: Goal-based and risk-based creation of adaptive workflow processes. In: Proceedings of the AAAI 2000 Spring Symposium on Bringing Knowledge to Business Processes, California (2002)

[2] http://www.codeproject.com/Articles/348918/Fuzzy-Ontology-Framework

3. Aalst, W.: Workflow Management: Models, Methods and Systems. MIT Press, London (2002)
4. Fent, A.: Design for Change: Evolving Workflow Specifications in ULTRAflow. University of Passau (2001) (accessed), http://www.im.uni-passau.de/publikationen/FRF02/FRF02.pdf
5. Dustdar, S.: Reconciling Knowledge Management and Workflow Management Systems: The Activity-Based Knowledge Management Approach. Journal of Universal Computer Science 11(4), 589–604 (2005)
6. Zhao, L.: Knowledge Management and Organizational Learning in Workflow Systems, Department of Information and Systems Management. Hong Kong University of Science and Technology (1998), http://chord.nce.ufrj.br/cursos/sctci2004/documentos/ais98-km.pdf
7. Wargitsch, C.: An Organizational-Memory-Based Approach for an Evolutionary Workflow Management System: Concepts and Implementation. In: Proceedings of the 31st Annual Hawaii International Conference on System Sciences, pp. 174–183 (1998)
8. Shukla, D.: Essential Windows Workflow Foundation. Pearson Education, London (2007)
9. Zhu, A.: Microsoft Windows Workflow Foundation 4.0 Cookbook. Packt Publishing, Birmingham (2010)
10. Almeida, T.: Flexible Workflow Execution through an Ontology-based Approach. In: Workshop on Ontologies as Software Engineering Artifacts (OOPSLA) (2004), http://swt-www.informatik.uni-hamburg.de/conferences/oopsla%202004%20-%20accepted/VieiraTati_CasanovaMarco_OOPSLA2004.pdf
11. Puleston, C., Parsia, B., Cunningham, J., Rector, A.: Integrating Object-Oriented and Ontological Representations: A Case Study in Java and OWL. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 130–145. Springer, Heidelberg (2008)
12. Frenzel, C.: Mooop – A Generic Integration of Object-Oriented and Ontological Models. University of Ausburg (2010), http://opus.bibliothek.uni-augsburg.de/volltexte/2011/1695/pdf/TR_2010_14.pdf
13. Zadeh, L.: Fuzzy Sets. Information and Control 8, 338–353 (1965)
14. Elst, L.: Weakly-structured Workflows for Knowledge-intensive Tasks: An Experimental Evaluation. In: Proceedings of the 12th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Washington, DC, USA, p. 340
15. Petcu, A.: FRODO: a FRamework for Open/Distributed Optimization. In: Ecole Polytechnique Fédérale de Lausanne (2006), http://liawww.epfl.ch/frodo/
16. Adsett, C.: Realizing Weak Work Workflow with Declarative Flexible XML Routing in SOAP (DeFleX). National Research Council of Canada (2004), http://www.iit.nrc.gc.ca/iit-publications-iti/docs/NRC-48062.pdf
17. Stavness, N.: Supporting Flexible Business Processes with a Progression Model. University of Saskatchewan (2004), http://www.cs.usask.ca/faculty/kas/papers/MBUI_2004_Progression.pdf
18. Covvey, D.: The Representation of Dynamic, Context-Informed Workflow. BPM and Workflow Handbook: Spotlight on Human Centric BPM (2008)
19. Marionovic, S.: Teleo-Reactive workflows for pervasive healthcare. In: Proceedings of 8th IEEE International Conference on PERCOM, Mannheim, pp. 316–321 (2010)
20. Adam, O.: Fuzzy Workflows-Enhancing Workflow Management with Vagueness. German Research Center for Artificial Intelligence (DFKI) (2003), http://www.tk.uni-linz.ac.at/EUROINFORMS2003_Workflow/rc30_1.pdf
21. Landry, J.-F., Ulmer, C., Gomez, L.: Fuzzy Distributed Workflows for Crisis Management Decision Makers. In: Ortiz-Arroyo, D., Larsen, H.L., Zeng, D.D., Hicks, D., Wagner, G. (eds.) EuroISI 2008. LNCS, vol. 5376, pp. 226–236. Springer, Heidelberg (2008)

22. Zirpins, C.: Flexible Workflow Description with Fuzzy Conditions. Dept. of Computer Science (VSIS), University of Hamburg (2007), http://www.ee.ucl.ac.uk/lcs/papers2002/./LCS037.pdf
23. Quirchmayr, G., List, B., Tjoa, A.M.: Analyzing Workflow Audit Trails in Web-Based Environments with Fuzzy Logic. In: Bauknecht, K., Madria, S.K., Pernul, G. (eds.) EC-Web 2001. LNCS, vol. 2115, pp. 315–325. Springer, Heidelberg (2001)
24. Wang, R.: Flexible Workflow Autonomic Object Intelligence Algorithm Based on Extensible Mamdani Fuzzy Reasoning System. In: Global Design to Gain a Competitive Edge (2008)
25. Bobillo, F.: An Expressive Fuzzy Description Logic Reasoner. In: ISTI-CNR (2008), http://gaia.isti.cnr.it/~straccia/software/fuzzyDL/download/old/documents/fuzzyDL.pdf
26. Straccia, U.: The fuzzy DL System. In: ISTI-CNR (2012), http://gaia.isti.cnr.it/~straccia/software/fuzzyDL/fuzzyDL.html
27. Bobillo, F.: DeLorean. Universidad de Zaragoza (2012), http://webdiis.unizar.es/fbobillo/delorean
28. Bobillo, F.: Fuzzy Ontology Representation using OWL. International Journal of Approximate Reasoning 52(7), 1073–1094 (2011)

# Constructs Replacing and Complexity Downgrading via a Generic OWL Ontology Transformation Framework

Ondřej Šváb-Zamazal[1], Anne Schlicht[2],
Heiner Stuckenschmidt[2], and Vojtěch Svátek[1]

[1] University of Economics, Prague
{ondrej.zamazal,svatek}@vse.cz
[2] University of Mannheim
{anne,heiner}@informatik.uni-mannheim.de

**Abstract.** Many of the tools supporting the OWL ontological language face complexity problems when handling certain constructs of the language. This leads to the requirement of automatically changing the ontology, either by removing a specific type of construct or by adhering (downgrading) the ontology to a predefined OWL2 profile such as OWL2 EL. We present an approach to construct replacing and complexity downgrading that relies on transformation patterns processed by a generic ontology transformation framework. Transformation patterns allow to declaratively formulate and transparently execute axiom replacement operations. This potentially preserves derivations that would otherwise be lost due to simple removal of problematic axioms.

## 1 Introduction

Existing tools operating on ontologies normally support a certain, well defined, set of logical operators. In many cases this set of operators is not sufficient to completely capture the semantics of the OWL language. As a result, these tools cannot be used on certain ontologies or they provide incomplete reasoning results. In both cases, the transformation of the input ontology can improve the situation. In particular, the ontology can be transformed into a version that only uses the supported operators. Doing this outside the tools gives the user more flexibility because (s)he can design a transformation that is not directly hard-coded into the tool.

In our previous work on the PatOMat project[1] we already addressed the general need for 'style' transformation in ontological engineering. In this paper we are concerned with the 'language profiling' scenario of transformation, i.e. replacing certain OWL constructs that could be hard for some tools. This replacement task is supported by a general *ontology transformation framework* [12], a simple *transformation pattern language*, and a set of web-based *services* relying on external tools such as the Ontology Pre-Processor Language (OPPL) and OWL-API, see Section. 2. The current paper extends [12] with a description of the language profiling scenario with its two use cases,

[1] http://patomat.vse.cz/

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 528–539, 2013.

a larger collection of transformation patterns, an experiment, new features of the core implementation and a new web-based application.

The rest of the paper is structured as follows. Section 2 briefly reviews the *PatOMat* framework, transformation language and processing services, in the current form. Section 3 introduces the language profiling scenario with its pipeline. This scenario is then split into two use cases; the first one (Section 4) describes the 'on purpose' construct replacement use case, while the second (Section 5) deals with complexity downgrading. Complexity downgrading is an extension of the first scenario in terms of applying more than one transformation pattern dynamically composed into a sequence according to recommendations from ontology analysis. Furthermore, an experiment is presented in Section 5.2 that deals with the second use case. Finally, Section 6 surveys related work, and Section 7 discusses the benefits of the approach and wraps up the paper.

## 2 *PatOMat* Transformation Framework

The central notion in the *PatOMat* framework[2] is that of *transformation pattern* (TP). A TP contains two *ontology patterns* (source OP and target OP) and the description of the transformation betweem them, called *pattern transformation* (PT). For instance, we can specify a TP such that a subsumption relation (as source, OP1) should be transformed to a SKOS[3] taxonomic relationship (as target, OP2). A schematic description follows.

```
OP1: ?OP1_A subClassOf ?OP1_B
OP2: ?OP2_A skos:broader ?OP2_B
PT:  ?OP1_A~?OP2_A    ?OP1_B~?OP2_B.
```

The representation of OPs is based on the OWL 2 DL profile. However, while an OWL ontology refers to particular entities, e.g. to class `Person`, in the patterns we generally use *placeholders*, e.g. ?*OP*1_*A*. Entities are specified (i.e. placeholders are instantiated) at the time of instantiation of a pattern. An OP consists of *entity declarations* (referring to placeholders or concrete entities), *axioms* and *naming detection patterns*; the last capture the naming aspect of the OP important for its detection.[4] A PT consists of a set of *transformation links* and a set of *naming transformation patterns*. Transformation links are either *logical equivalence relationships* or *extralogical relationships* holding between pairs of entities of different type (such as class vs. individual, as in our example above). Naming transformation patterns serve for generating new names for old or newly created entities. Naming patterns range from *passive naming operations* such as detection of a head noun for a noun phrase to *active naming operations* such as derivation of a verb form of a noun.

For instance, the abovementioned TP would transform the following OWL ontology fragment

```
Paper subClassOf Document. Review subClassOf Document.
ConferencePaper subClassOf Paper. JournalPaper subClassOf Paper.
```

[2] [12] provides more details about the framework, and at `http://owl.vse.cz:8080/tutorial/` there is a fully-fledged tutorial for the current version.

[3] `http://www.w3.org/TR/skos-primer/`

[4] The naming aspect is less important for language profiling than it is for ontology matching or importing (merging).

to the SKOS terminology fragment

```
Paper skos:broader Document. Review skos:broader Document.
ConferencePaper skos:broader Paper. JournalPaper skos:broader Paper.
```

The framework prototype implementation is available either as a *java library* or as three *core services*.[5] The java library is directly used in a web-based application briefly described in Sec. 5. The whole transformation is divided into three steps, which correspond to the three services:

- The *OntologyPatternDetection* service takes the transformation pattern and a particular original ontology on input, and returns the binding of entity placeholders on output, in XML. The structural/logical aspect is captured in the structure of an automatically generated SPARQL query;[6] the naming aspect is dealt with based on its description within the source pattern.
- The *InstructionGenerator* service takes the particular binding of placeholders and the transformation pattern on input, and returns particular transformation instructions on output, also in XML. Transformation instructions are generated according to the transformation pattern and the pattern instance.
- The *OntologyTransformation* service takes the particular transformation instructions and the particular original ontology on input, and returns the transformed ontology on output.

The third service is partly based on OPPL [2] and partly on our specific implementation over OWL-API.[7] Currently we use OPPL for the operations on *axioms* and for *adding entities*, and OWL-API for *re/naming* entities according to naming transformation patterns and for adding *OWL annotations*. As far as detection is concerned, the SELECT part of OPPL could be used to some extent; our naming constraints are however out of the scope of OPPL. Furthermore, in contrast to OPPL, we *decompose* the process of transformation into parts, which enables user intervention within the whole workflow.

The framework has been recently enriched with several advanced features such as *recursive* processing of structures in ontologies in a single step both in the detection phase and in the actual transformation phase. Furthermore, multiple alternative strategies can be applied in handling *additional axioms*, i.e. axioms that are not a literal part of an input pattern but get affected by its transformation; in this case removal can be allowed for both axioms and entities (with additional options that we omit for brevity), axioms only, or none.

The framework was previously explored for two other scenarios. The *ontology matching* scenario, where two ontologies are to be matched, is based on the idea that we can transform the modelling style of one ontology so as to make automated matching to the other ontology easier [12]. Another scenario deals with importing (merging) a best-practice *ontology content pattern* into an existing 'provisional' ontology, which thus needs to be adequately adapted [11].

[5] All accessible via the web interface at `http://owl.vse.cz:8080/`

[6] `http://www.w3.org/TR/rdf-sparql-query/`

[7] `http://owlapi.sourceforge.net/`

## 3 *PatOMat* in Use: Language Profiling Scenario

In comparison with those other scenarios, the language profiling scenario leads to a fully automatic pipeline. First, a source ontology is pre-processed in order to syntactically decompose the constructs that can hinder querying in a unified way. In our case we decompose (→) the following constructs:

- *intersection*: A subClassOf (B and C) → A subClassOf B. A subClassOf C.[8]
- *disjointness*: DisjointClasses(B, C, D) →
  B disjointWith C. C disjointWith D. B disjointWith D.
- and *disjoint union*: DisjointUnion(B, C, D)[9] →
  B equivalentTo C or D. C disjointWith D.

The next step is a detection performed by the *OntologyPatternDetection* service. There is typically more than one pattern instance as a result of the detection step. Furthermore, it is usually precise, because in the case of the language profiling scenario, detection is merely based on structural/logical aspects and naming detection patterns are mostly not needed. The following step amounts to generation of transformation instructions by the *InstructionGenerator* service. Finally, the application of instructions is carried out by the *OntologyTransformation* service according to the selected transformation strategy. By default, it uses the 'progressive' transformation strategy, which enables the removal of axioms but not the removal of entities.

This is the basic pipeline of the language profiling scenario as applied in the first use case (cf. Section 4). In contrast, the complexity downgrading use case (cf. Section 5) slightly modifies the pipeline by adding an analysis of the source ontology to specify which transformation patterns should be applied. Consequently, selected transformation patterns are dynamically composed into a sequence. Finally, a post-processing step is performed for ensuring completness of the process.

The transformation of language profiling constructs can be generally done in three ways: either they can be replaced with an *equivalent different representation*, or they can be replaced with an *approximate different representation*, or they can be *removed*. The first option is obviously the best one. However, it is only rarely possible to find an equivalent representation using other constructs when in need to eliminate a problematic construct during complexity downgrading. The second option is more realistic. However, there is often no (even approximate) alternative way of representation and the problematic construct has to be simply removed.

## 4 Ontology Transformation for Specific Language Construct Replacement

Transformation can be driven by a request for replacing a specific language construct. In this section we provide an example dealing with nominals. Tackling nominals can be

[8] For writing axioms we use the intuitive Manchester syntax available at: http://www.w3.org/TR/owl2-manchester-syntax/

[9] B is the disjoint union of C and D.

problematic for some reasoners. In the following example we will show how nominals can be replaced rather than removed. Let us assume that we have nominals describing continents and the `Continent` class defined as 'one of' those nominals (implicitly assuming their mutual difference):

```
Continent equivalentTo {Africa, America, Antarctica, Asia, Australia, Europe}.
```

Let us further assume that we have the `AfricanRedSlip` class,[10] defined via the `hasContinentOfOrigin` property:

```
AfricanRedSlip subClassOf Ware.
AfricanRedSlip subClassOf (hasContinentOfOrigin value Africa).
```

Nominals could be simply removed; however then we would lose e.g. part of the description of `AfricanRedSlip`. Instead, we can replace a set of nominals by union of helper classes *xxx*`_nc` each one accomodating exactly one original instance of the nominal class:

```
OneOfContinent equivalentTo (Africa_nc or America_nc or
       Antarctica_nc or Asia_nc or Australia_nc or Europe_nc).
Africa a Africa_nc. America a America_nc.
Antarctica a Antarctica_nc. Asia a Asia_nc.
Australia a Australia_nc. Europe a Europe_nc.
```

This transformation is approximate because it is no longer assured that e.g. `Africa_nc` could not have other individuals than Africa. Due to this change we should also modify the description of `AfricanRedSlip`:

```
AfricanRedSlip subClassOf Ware.
AfricanRedSlip subClassOf (hasContinentOfOrigin some Africa_nc).
```

This can be done automatically using our framework with a specific TP.[11] It is worth noting that, historically, nominals used to be represented in this ('transformed') way.

Further off-the-shelf transformation patterns for replacing different OWL constructs are available online.[12] They can be divided into three groups: equivalent, approximate and removed representations (cf. Section 3).

## 5 Ontology Transformation for Complexity Downgrading of an Ontology

The transformation can also be driven by an ontology complexity requirement of some tool. In such a case, transformation usually comprises more than one transformation pattern in order to achieve the required complexity level. In this work we focus on the

[10] African red slip is a kind of ancient pottery, see `http://open.vocab.org/docs/AfricanRedSlip`

[11] The pattern is available from `http://nb.vse.cz/~svabo/patomat/tp/lr/tp_nominals-6a.xml`

[12] `http://nb.vse.cz/~svabo/patomat/tp/lr/`; there is a link to the XML serialization of each pattern, a short description, and an ontology on which the pattern can be tested.

complexity level corresponding to the OWL2EL profile [7], since this profile is supported by many tools, e.g. the ELOG-reasoner [8]. In comparison with the use case from the previous section there are two more steps. Based on a given list of forbidden constructs, *ontology analysis* figures out (by using the OWL-API library) which transformation patterns have to be executed. These patterns are added into a sequence of transformation patterns and then executed by the transformation in a sequential order. Additionally there is a *post-processing* step where the remaining forbidden constructs are removed using the OWL-API library. This step ensures completeness of the downgrading process.

Both use cases from Section 4 and 5 are supported by a *web-based application*[13]. Following the input of the source ontology URI (and selected TP in the first use case), the transformed ontology is displayed (together with a brief transformation log) and a link to its code is also provided.

### 5.1 Transformation Patterns Employed in Downgrading to OWL2EL Profile

There are several OWL 2 constructs that are not supported in the OWL2EL profile [7]: universal quantifications to a class expression, cardinality restrictions, disjunctions, class negations, enumerations involving more than one individual, disjoint properties, irreflexive object properties, inverse object properties, functional and inverse-functional object properties, (a)symmetric object properties.

It is generally difficult to find some replacement of unsupported constructs since their replacement usually leads to using other unsupported constructs, e.g. ObjectMaxCardinality could be replaced by a complemented ObjectMinCardinality restriction, which is equally forbidden in OWL2EL.

In the following, we go through three different language constructs that can be replaced using our transformation patterns (replacement transformation). We describe them briefly and exemplify the preserved implications. However, let us note that different solution as a transformation pattern can be suggested and applied in the framework. Finally, we provide an experiment illustrating the effect of transformation on query answering results.

**Complement of Universal Restriction.** The complement construct is not allowed in OWL2EL at all. We can approximately replace it using existential restriction wrt. the top concept, i.e. instead of having

```
PizzaWithTopping subClassOf (not (hasTopping only Tomato))
```

we will have the following

```
PizzaWithTopping subClassOf (hasTopping some Thing)
```

In order to exemplify the preservation of derivations, let us consider that we also have the following axioms in our TBox:

```
(hasPizzaIngredient some Thing) subClassOf Pizza
hasTopping subPropertyOf hasPizzaIngredient
```

[13] Available from http://owl.vse.cz:8080/Downgrading/

If the problematic axiom were only removed and not replaced the following subsumption could not be inferred:

```
PizzaWithTopping subClassOf Pizza
```

The corresponding transformation patterns described in this paper are available online[14].

**Minimum Cardinality.** Cardinality restrictions are not allowed in OWL2EL. Minimum cardinality of 1 can be *equivalently* replaced by an existential restriction applied on the same filler class. In the case that the minimum cardinality is higher than one, an *approximate* transformation can be applied.

For example, instead of having

```
AcceptedPaper subClassOf (hasDecision min 2 Acceptance)
```

we will have the following

```
AcceptedPaper subClassOf (hasDecision some Acceptance)
```

In order to exemplify the preservation of derivations, let us consider that we also have the following axioms in our TBox:

```
EvaluatedPaper = hasDecision some Decision
Acceptance subClassOf Decision
```

This implies

```
AcceptedPaper subClassOf EvaluatedPaper
```

**Enumerations of More than One Individual.** The OWL2EL profile only permits enumeration of one individual, therefore transformation must be carried out in the cases with higher number of individuals. We suggest the following approximate transformation. Instead of having

```
EurAsia = {europe, asia}
```

we will have the following

```
Europe_nc = { europe }. Asia_nc = { asia }.
Europe_nc subClassOf EurAsia
Asia_nc subClassOf EurAsia
```

We assume that the individuals `Europe` and `Asia` are different. However, in this way we cannot express that every `EurAsia` is either `Europe_nc` or `Asia_nc`.

In order to exemplify the preservation of derivations, let us consider that we also have the following axioms in our TBox:

```
EuropeanWatch = ( hasContinentOfOrigin hasValue europe )
EurAsiaWatch = ( hasContinentOfOrigin some EurAsia )
```

This implies

```
EuropeanWatch subClassOf EurAsiaWatch
```

[14] http://nb.vse.cz/~svabo/SOFSEM2013/

### 5.2 Experiment

We performed an experiment about the effects of *replacement transformation* in comparison with *removal transformation*, which simply removes the axioms involving forbidden constructs by OWL-API. The experiment had three steps:

1. *Ontology collection gathering*. In order to gather collection of ontologies we used the Watson semantic search[15]. We applied four selection criteria for selecting ontologies into the collection: OWL ontology language (target language of the framework), more than 10 classes, more than 5 properties (ontologies should not be too small), and absence of imports (current limitation of the OPPL tool and thus of the framework). This gives us 328 ontologies. Final criterion says that an ontology must have at least one forbidden construct transformable by transformation patterns. This reduced the set of ontologies to 63. However, due to parsing problems (in OWL-API or Jena[16]), other syntactical problems in ontologies and inconsistent ontologies, we had finally 38 ontologies in our experimental collection.
2. *Transformation of ontologies*. We transformed each of these original ontologies (O variant of an ontology; see in the `ontologies` directory[17]) into the OWL2EL profile using *removal transformation* (R variant of an ontology; see in the `ontologiesR` directory), using simple modifications of ontologies such as adding declarations of classes and properties using OWL-API (ST variant of an ontology; see in the`ontologiesST` directory) and using *replacement transformation* with an application of our transformation patterns (T variant of an ontology; see in the `ontologiesT` directory).
3. *Comparison of number of preserved subsumption relations*. Finally, for each transformed version of an ontology we computed the subsumption relations included in the ontology explicitly (using ARQ in Jena) or implicitly (using ARQ in Jena and Pellet reasoner[18]). The generated query was in the following shape: *ASK Class1 rdfs:subClassOf Class2* [19]. Then we automatically compared the preserved subsumption relations in the R variant wrt. subsumption relations in the original ontology, preserved subsumption relations in the ST variant wrt. subsumption relations in the original ontology, and, finally, preserved subsumption relations in the T variant wrt. subsumption relations in the original ontology.

The number of problematic axioms (obtained using OWL-API) ranged from 11 to 2133. Besides the forbidden constructs listed in Section 5.1 there were forbidden datatypes in data range and undeclared classes or properties[20]. The number of minimum cardinality

[15] `http://kmi-web05.open.ac.uk:8080/WatsonWUI/`

[16] `http://jena.apache.org/`

[17] Detail web-page report about the experiment along with downloadable collection of ontologies is at: `http://nb.vse.cz/~svabo/SOFSEM2013/`

[18] `http://clarkparsia.com/pellet`

[19] Class1 and Class2 were iteratively bound with all combinations of named classes from given ontology.

[20] Although a declaration is not matter of logic, an OWL ontology without declarations is incomplete and thus in the OWL Full profile.

**Table 1.** Effects summary

| | number of ontologies |
|---|---|
| no difference between O and R variants | 17 |
| no positive effect wrt. saved subsumption relations | 13 |
| saved subsumptions due to simple modifications | 7 |
| saved subsumptions due to replacement transformations | 1 |

replacement transformations ranged from 1 to 120 applied on all ontologies in the collection and the number of enumeration replacement transformations ranged from 1 to 27 only applied on 8 analysed ontologies.

In total, there were 17 ontologies in which removal transformation had no negative effect on subsumption relations (Table 1), including 3 ontologies which had no subsumption relation in the original ontology at all. Next, for 13 ontologies any kind of transformation did not save subsumption relations. It turns out that simple modifications (ST variant) improved 7 ontologies with regard to lost subsumption relations ranging from 1 to 75 saves. Detailed analysis showed that those seven ontologies missed classes or properties declarations and these were simply added using OWL-API. Without these modifications the removal transformation by OWL-API simply removed all axioms in which the problematic entities were involved. Consequently, this also removed asserted subsumption relations. Finally, there was only one positive effect caused by minimum cardinality replacement transformation, in which the number of missing subsumption relations decreased from 74 to 62.

Let us have a closer look at one example of preserved subsumptions there. The replacement transformation preserved, for instance, the following subsumption relations (an equivalence is decomposed into (1) and (2)):

```
(1) Module subClassOf StructuralElement
(2) StructuralElement subClassOf Module
```

These subsumption relations are derived based on the following axioms:

```
Module subClassOf (element min 1).
StructuralElement subClassOf (element min 1).
element Domain Module.
element Domain StructuralElement.
```

Thanks to replacement transformation in which "Module subClassOf (element min 1)" was replaced by "Module subClassOf (element some Thing )" (analogically for StructuralElement) the (1) and (2) relations were preserved.

This weak overall effect (potentially even intensified considering the 265 ontologies in which no replaceable forbidden constructs were identified[21]) can be explained by the fact that the current replacement transformation patterns cover a small set of all problematic issues only. However, any newly designed transformation pattern can be employed within the process in the future. Next, the replacement trasformation can only have a positive impact (in the setting of our experiment) if there are further axioms due to which subsumption relations can be derived (as demonstrated step-by-step

[21] On the other hand, we did not check how many of them have forbidden constructs.

for each replacement transformation in Section 5.1). Last but not least, we should also consider that this experiment only evaluates the effect of preserved subsumption relations but it does not evaluate the first mentioned use case, which is an "on purpose" construct replacement use case (Section 4). This should be accordingly reflected in a future experiment.

Regarding the time performance, which includes pattern detection, instructions generation and transformation, a cardinality replacement transformation takes approximately ten seconds, while an enumeration replacement transformation takes twenty seconds. In the case of enumeration replacement transformation the time increases with a number of transformations because it is applied iteratively over an ontology, while a cardinality replacement transformation runs only once for all applicable cardinality transformations in an ontology.

## 6 Related Work

Prior research on ontology simplification can be divided into generic approaches and those specifically tailored for a certain (popular) reasoner. Additionally we also consider general approaches to ontology transformation (not confined to simplification). The following three paragraphs reflect this distinction.

[6] aimed at elimination of transitivity axioms from an ontology in order to reduce its expressivity. [1] presented an inference service for approximate translation of a concept from one Description Logic to (typically) less expressive Description Logic. In comparison with our approach, both these approaches center on logical features, while we follow a more engineering-oriented approach, taking into account the view of the human modeller. There is also the approach published in [10], which aims at tractable TBox reasoning over a very expressive Description Logic. They proposed approximate TBox reasoning using EL rules and additional deduction rules. Transformation of badly tractable constructs are realized as additional data structures. In comparison, our approach addresses general transformation and is centered around the idea of transformation patterns as reusable transformation rules, while reasoning as such is left to reasoning tools. Thus, while in our approach a tool obtains a transformed ontology, in the case of the approach in [10] the transformation is used for an approximate reasoning algorithm and there is no transformed version of an ontology on the output. Furthermore, [10] does not consider nominals replacement and does not remove every non-EL axiom.

Regarding the tricky expressions for a particular reasoner, in [4] there has been presented the lint tool Pellint applicable on ontologies incurring reasoning performance problems to the Pellet reasoner.[22] Particularly, Pellint detects problematic modeling constructs as patterns. There are two groups of patterns: axiom-based patterns dealing with a single axiom, and ontology-based patterns dealing with two or more axioms in the whole ontology. These patterns could be captured by means of our transformation patterns to some extent.

The most prominent project in ontology transformation in general (i.e. aside the simplification setting) is probably OPPL [2], which we introduced in Section 2; we directly reuse it in our framework. In [9] the authors consider ontology translation from the

[22] http://clarkparsia.com/pellet/

Model Driven Engineering perspective. The basic shape of our transformation pattern (as described in detail in [12]) is very similar to their meta-model. However, the transformation is considered at the data level rather than at the schema level as (primarily) in our approach. In [5] the authors presented an ontology update framework that can automatically apply change patterns capturing the evolution of a domain of interest. Their approach is however based on the RDF model and SPARQL update language, while our approach is built on the top of the OWL model.

## 7 Conclusions and Future Work

This paper presents an approach to ontology construct replacing and complexity downgrading where pattern-based transformation is applied on the source ontology to derive a target ontology. We explained these two use cases and demonstrated their usefulness on examples. We also performed a tiny experiment from the reasoning perspective; the positive effect of our approach was only weak there, which is attributed to the fact that the current replacement transformation patterns only cover a small subset of problematic issues and the ontologies do not contain additional axioms needed for derivation of subsumption relations with replaceable forbidden constructs. The strong point of the presented approach is however that, in contrast to research focused on solving widely the 'notorious' problems of logical inference, the users can easily design their own transformation patterns[23] to address a certain, specific and unforeseen, construct-replacing use case, such as that specifically dealing with nominals (Section 4) or complexity downgrading for a certain, newly introduced profile (Section 5). If such patterns are shared, other users could easily apply them through the online transformation web services (i.e. without the necessity to install a particular reasoner as in the logic-centric approaches to transformation).

We plan to investigate what other kinds of transformation patterns and use cases and, moreover, other complexity downgrading tasks, could be addressed by the presented framework. Our approach could be further improved e.g. by precomputing the subsumptions of named classes in the source ontology and adding them into the target ontology. Regarding practical implementation, we plan to extend the support of the framework for analogous datatype-related constructs in OWL such as *DataOneOf*.

*This research has been partially supported by the DAAD grant "Pattern-based ontology transformation supporting ontology matching and reasoning tasks" and by CSF grant No. P202/10/1825, "PatOMat – Automation of Ontology Pattern Detection and Exploitation".*

## References

1. Brandt, S., Kuesters, R., Turhan, A.-Y.: Approximation and Difference in Description Logics. In: 8th Conf. Principles of Knowledge Representation and Reasoning, Toulouse (2002)
2. Egaña, M., Stevens, R., Antezana, E.: Transforming the Axiomisation of Ontologies: The Ontology Pre-Processor Language. In: W'shop OWL Experiences and Directions, Washington, DC (2008)

[23] Recently a graphical editor of TP authoring has been released as plug-in for Eclipse: `http://owl.vse.cz:8080/tpe/`

3. Iannone, L., Palmisano, I., Rector, A., Stevens, R.: Assessing the Safety of Knowledge Patterns in OWL Ontologies. In: Aroyo, L., Antoniou, G., Hyvönen, E., ten Teije, A., Stuckenschmidt, H., Cabral, L., Tudorache, T. (eds.) ESWC 2010, Part I. LNCS, vol. 6088, pp. 137–151. Springer, Heidelberg (2010)
4. Lin, H., Sirin, E.: Pellint - A Performance Lint Tool for Pellet. In: W'shop OWL Experiences and Directions, Karlsruhe (2008)
5. Lösch, U., Rudolph, S., Vrandečić, D., Studer, R.: Tempus Fugit - Towards an Ontology Update Language. In: Aroyo, L., Traverso, P., Ciravegna, F., Cimiano, P., Heath, T., Hyvönen, E., Mizoguchi, R., Oren, E., Sabou, M., Simperl, E. (eds.) ESWC 2009. LNCS, vol. 5554, pp. 278–292. Springer, Heidelberg (2009)
6. Motik, B.: Reasoning in Description Logics using Resolution and Deductive Databases. PhD thesis, Univ. Karlsruhe (2006)
7. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web Ontology Language Profiles. W3C Recommendation (2009), http://www.w3.org/TR/owl2-profiles/
8. Noessner, J., Niepert, M.: ELOG: A Probabilistic Reasoner for OWL EL. In: Rudolph, S., Gutierrez, C. (eds.) RR 2011. LNCS, vol. 6902, pp. 281–286. Springer, Heidelberg (2011)
9. Silva Parreiras, F., Staab, S., Schenk, S., Winter, A.: Model Driven Specification of Ontology Translations. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 484–497. Springer, Heidelberg (2008)
10. Ren, Y., Pan, J.Z., Zhao, Y.: Soundness Preserving Approximation for TBox Reasoning. In: AAAI (2010)
11. Svátek, V., Šváb-Zamazal, O., Vacura, M.: Adapting Ontologies to Content Patterns using Transformation Patterns. In: WOP (2010)
12. Šváb-Zamazal, O., Svátek, V., Iannone, L.: Pattern-Based Ontology Transformation Service Exploiting OPPL and OWL-API. In: Cimiano, P., Pinto, H.S. (eds.) EKAW 2010. LNCS, vol. 6317, pp. 105–119. Springer, Heidelberg (2010)

# Tempo Adaptation within Interactive Music Instruments in Mobile Phone

Marek Takáč and Alena Kovárová

Faculty of Informatics and Information Technologies,
Slovak University of Technology in Bratislava, Slovakia
xtakac@gmail.com, kovarova@fiit.stuba.sk

**Abstract.** Nowadays it is not unusual, that applications for mobile devices are using various sensors such as touch screen and accelerometer for their control. We aim to use these mobile sensors for music creation in a way that is intuitive to users. The main contribution of this paper is proposed formula for real-time tempo adaptation for a single user. We also dealt with the control design of musical instruments of different types (to make it similar to their real counterparts) so that users can easily adapt to the device. To test it, we have implemented an application simulating three musical instruments: piano, flute and drums ex-tended by a metronome, editable rhythms and other configuration settings in-cluding tempo adaptation.

**Keywords:** mobile device sensors, music, tempo adaptation.

## 1 Introduction

Mobile devices are undoubtedly one of the phenomena of our time. Mobile development is accessible to almost everybody, not only in developed countries, as almost everybody can now own some type of mobile device. As of 2012, it is relatively easy to obtain a device with a large screen, GPS navigation and with multiple integrated sensors. This technological shift significantly increases the range of applications on mobiles in terms of computing, but also in ways that users are interacting with the device.

All of these devices can be used for interactive music creation. Creating sound output using mobile device sensors is relatively young and, therefore, studied only in the last few years. This area of research can be further developed and that is the aim of this work.

Our first challenge was to create musical instruments with controls similar to those on real instruments so that the interface is intuitive for users. There are different types of musical instruments; we focus on the following three classes – keyboard, wind and percussions. We have explored the potential of today's mobile devices (in way of sensors combination) to achieve intuitive control, however the available computing power, device memory and sensors offer space to create additional features that a user might like. For our second challenge we decided to look at metronome, rhythms and automatic tempo adjustment.

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 540–551, 2013.

## 2 Related Work

### 2.1 Sensors for Mobile Music Performance

Modern mobile devices contain variety of sensors that can be used to generate music in very different ways. Michael Rohs and Georg Essl described this in their publication named Interactivity for Mobile Music-Making [1]. This work contains classification of available sensors in mobile devices that are relevant to making music according to different dimensions. A simplified matrix of the classified sensors is presented in Figure 1.

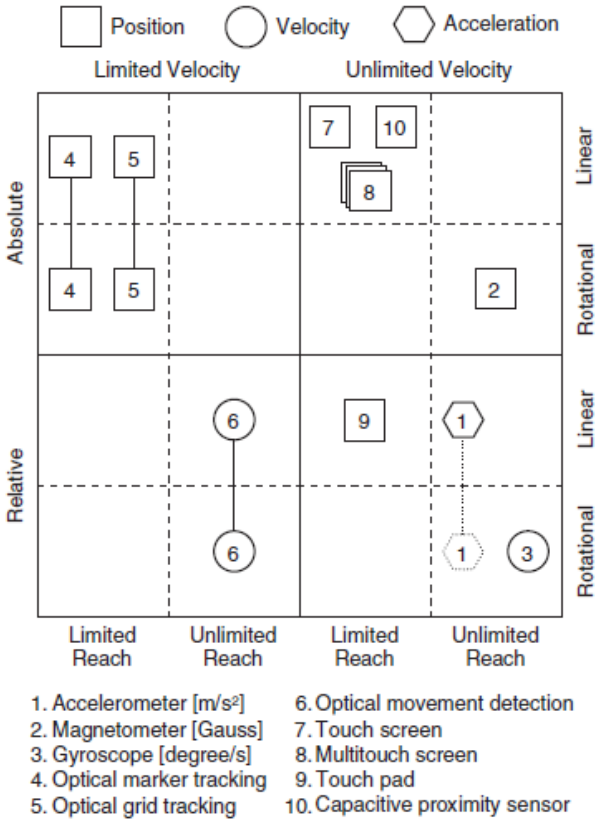



**Fig. 1.** Design space of sensors for mobile music performance [1]

In general, sensors can measure linear or rotational movement. Additionally, there are sensors that can measure both linear and rotational movement; these are represented as connections between cells in the matrix.

In context of mobile music creation it is also very important how particular sensors are utilized. Many sensors are usable only if they work within the available constraints of the user. This means that in some cases users are limited by sensor (e.g. in

optical sensing at low frame rates), which is shown in the figure by dimension limited/unlimited velocity. Some musical phrases, which rely on speeds faster than the sensor technology can reliably track, may be unavailable to the user. Sensors placed in unlimited velocity dimension are able to detect rapid movements so this limitation does not apply.

Dimension limited/unlimited reach means that sensor is limited (or not) by physical interaction space. For example, touch screen is limited by its size, or proximity sensor is limited by maximum sensing distance. In musical context this means that mapping specific sounds to interaction is more difficult on sensors with limited reach.

Data provided by sensors can be absolute or relative. Absolute data sensors are generally easier to work with in musical context because they can be easily mapped to particular tone pitch or volume. The same is possible with relative data, but it is necessary to propose more complex transformation of data to get specific sounds [1].

When we looked at existing applications, several interactive applications aimed on creating musical tones directly on mobile devices have been created. They mainly represent an individual musical instrument, like piano, guitar, acoustic drums, ocarina. In next lines we mention applications, which we consider as interesting (simply by using them) from their group of type. All of these applications are for iPhone OS, but similar application with the same control principles can be found also for other operating systems.

***Keyboard musical instruments***

- Piano+ ([1]. . . /piano+/id430119524)
- Virtuoso ([1]. . . /virtuoso-piano-free-2-hd/id304075989)
- WaveSynth ([1]. . . /wavesynth/id310846058
- Gyro Piano ([1]. . . /gyro-piano/id382551316

All of these applications are controlled mostly by a *touch-screen*. They usually include a simple graphical interface showing one or two octaves of piano keyboard on the screen. Some of them support multi-touch. In piano it is crucial to change between octaves – it is implemented either as a manual switch (touch the button) or, e.g. in Gyro Piano, it uses the *accelerometer* and *gyroscope* – to move across the piano keys the device has to be moved in the x-axis.

***Wind musical instruments***

- Smule's iPhone Ocarina ([1]. . . /ocarina/id293053479

This Ocarina [2] is probably the most famous application of this kind. It is controlled via the *touch screen* (multi-touch, by touching the screen at those places where are the holes and this contact will "cover" them) and blowing into a *microphone*. It also uses *accelerometer* to simulate vibrato of played tone.

***Percussions***

- Amazing Drums ([1]. . . /amazing-drums-lite/id424438240)
- Drums Deluxe Light ([1]. . . /drums-deluxe-light/id399326714))

[1] `http://itunes.apple.com/us/app/...`

- Gyro Air Drums ([1]. . . /gyro-air-drums/id383027843 or http://www.gyroairdrums.com/)
- vDrummer ([1]. . . /drums/id311549739 or http://www.v-drummer.com)
- Drum Meister ([1]. . . /drum-meister-pro-lite/id384869832)

Most of these applications play the drums and percussion through interaction with multiple sensors of the device. In the first case, the *touch screen* with displayed image of drum set. This allows a user to create sound by touching respective drum. The second and much more interesting way of sound creation is if a user can "play" on the instrument by inclination and "hitting" in space. For this purpose is used the built-in *gyroscope* that makes it possible to determine to which particular part of the instrument was "hitting" and also to recognize gestures representing the strike, but there is a drawback – there is not entirely accurate detection of beats of drums.

### 2.2 Tempo Adaptation Based on User's Interaction

There are several types of algorithms that detects tempo of the song. Tempo is measured in units called BPM (Beats per Minute), which determines the number of "beats" (or more precisely – quarter notes) per minute. The process which is behind the detection of beats (hence the tempo) is an audio signal analysis, including finding the positions with greater amplitude, which usually represents snare drum hits in tempo. Song can be split to parts and then the process of finding regional maxima can separate "real" beats in tempo from soft dynamics hits [3].

There are several algorithms that are applicable to tempo detection [4,5] but these algorithms are used to find out tempo of existing track and not during its formation as in our case. To detect tempo we are using our suggested algorithm (see Section 3.3), inspired by Goto and Muraoka [4]. This algorithm is based on regular analysis of IBI (inter-beat interval - interval between two successive strokes in rhythm). The tempo is not changed if the last interval is a multiple of the current interval. It is evident that human is not capable of playing tones accurately to milliseconds in the tempo so in this algorithm we use an error tolerance value. Tempo is changed if the IBI maintains approximately the same value specified number of times in a row, and is different from the current tempo.

### 2.3 Audio Output Playing

To have a sound (music instrument) we had to deal also with audio output. In principle, audio output can be played via MIDI interface on external device, operating as a sequencer or by direct playback of prepared audio samples stored on device. The first approach was used in application Camus [6]. However, this application is not designed to generate music, but only to modify existing songs with musical effects. But our application should use only the iPhone audio playback capabilities, therefore we have decided to use the sound samples. They are stored in memory of device as files containing samples of tones. The great advantage of this approach is that it possible to use high-quality music samples, which could significantly improve the quality of audio output. However, to minimize memory requirements (our average sample file has

84 kilobytes), application contains only a small number of samples (e.g. virtual piano contains only 20 samples out of 88). Then we can increase or decrease the frequency (pitch) of these samples by so-called pitch shifting algorithm. This algorithm adjusts pitch of any tone in playback in real time by a selected interval. Tone frequency change is also related the change of song duration. Pitch shifting algorithms are designed to adjust only frequency and sound duration is maintained (the reverse process of this algorithm is known as time stretching).

## 3 Interaction Improvements and Algorithms

### 3.1 Sensors for Mobile Music Performance

Today's mobile phones contain these useful sensors: touch screen, accelerometer, gyroscope, camera and microphone. We propose the following improvements:

***Keyboard musical instruments:*** To move along the keyboard, the user can either use *swipe gesture* or shift the keyboard (detected by gyroscope) by changing the *tilt* of device which can be detected by 3-axis *accelerometer*.

***Wind musical instruments:*** Our plan was to use a *camera* of device as a sensor which will detect "covering" of flute's thumbhole by real-time analysis of captured images. But in most of devices, the camera is in opposite position to microphone, therefore it would be very hard for user to control it. Moreover this type of instruments is used to have many holes. This is also a problem, because operating systems allows detecting only five touches at one time.

***Percussions:*** Enhance it by *shaking* which can be detected by *accelerometer* – user's shaking gestures with device can play sounds of e.g. shaker or tambourine.

### 3.2 Sound Playback Algorithm

In addition to standard instruments mentioned in previous section (we call them "interactive"), there are also others that the user does not control directly – "non-interactive", and these instruments are used to create a sound playback, for example metronome or different rhythms. They play pre-defined sounds without user's interaction. During the playback the algorithm has to determine the time when concrete sound should be played. This can be done by simple formula:

$$time_n = time_{n-1} + \frac{240}{tempo} duration_{n-1} \tag{1}$$

where $time_n$ is $n$-th tone, $tempo$ is tempo of non-interactive instrument $duration_{n-1}$ is duration of $n$-1 tone. Using this recursive formula we can calculate the specific time when should be $n$-th tone played. This time is represented in seconds and so the numerator has to be 240, which is the multiplication of seconds in a minute with number four, because in music theory tempo is defined as number of quarter tones played in one minute. Our equation is similar to Reidsma et al. [7] equation 1 with conductor, but in our case the system follows user's tempo and does not try to lead him/her.

### 3.3 Automatic Tempo Adjustment Algorithm

Tempo of selected rhythm adapts during user interaction with one of three virtual instruments that results in a different tempo. As mentioned earlier, if new tempo is in same ratio (usually multiple of powers of two, but it is not a rule) to the original value, tempo change is considered natural and therefore it will not be changed. So the situation can be divided into two cases. In first case, if new tempo is in natural ratio to original it should not be changed and in second case, if is it not tempo is changed. To find this information was used and experimental fine-tuned following formula:

$$changetempo = \begin{cases} 1 \text{ if } \forall i \left| \frac{\text{newIBI}}{\text{oldIBI}} - multi_i \right| > \frac{\text{error}}{\text{multi}_\text{i}} \\ 0 \text{ else} \end{cases} \quad (2)$$

where *changetempo* is information whether is necessary or not to change the tempo, *newIBI* is time interval between the last played tones, *oldIBI* is time interval between the tones in the original tempo. Variable $multi_i$ stores *i*-th multiple of old tempo which can be considered in natural ratio with new tempo and error is time error value which can be tolerated musical timing of user. The result says that the tempo change is necessary if in tempo variable is number 1; otherwise new tempo is in natural ratio to original. In application we are using five multiples stored in variables multi1-multi5, namely the numbers: 1/4, 1/2, 1, 2 and 4. We also use value 0.2 stored in variable *error*, because during the tests it gives us the best and most natural results for tempo adaptation. From implementation point of view automatic tempo adjustment algorithm is based on observer design patterns.

## 4 Evaluation

To test our ideas we implemented three interactive musical instruments - piano, flute with five holes, drums and two non-interactive – metronome and drum rhythm with

**Fig. 2.** The basic screenshots of our application, from left: piano, flute, drums, settings of non-interactive instruments and automatic tempo

ability to adapt to user's tempo (see Figure 2). We implemented it for iPhone and we have used a number of frameworks (Core Motion, Core Media, AVFoundation, UIKit etc.) and for audio playback was used OpenAL API.

We conducted three types of testing – quantitative, qualitative and comparative. The number of testers was 10 – 3 women and 7 men, most of them of age 23 (the youngest was 19 and the oldest 48 years old). One of them had high IT skills and had experiences with iPhone. Other testers usually at least had experience with the touch screen but not with iPhone. Two testers did not have experiences with touch screen at all. We did not compare the adaptation, because there is no other application with such feature.

### 4.1 Quantitative Testing

The first test we conducted was aimed at tracking the time respondents needed to perform specified tasks. The results of this testing should point out how easy and intuitive interface our applications can be controlled. The average length of individual tasks reveals if there are some the poorly designed parts. The tasks were designed to cover most areas application functions (see Table 1).

**Table 1.** Tasks and their performance times realized by 10 testers

| *Task / Tester* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* |
|---|---|---|---|---|---|---|---|---|---|---|
| 1. Push „c3" key on the piano | 8.9 | 20.6 | 11.6 | 7.4 | 8.2 | 8.2 | 7.5 | 13.7 | 5.9 | 6.8 |
| 2. Play a tone on the flute | 7.9 | 10.2 | 40.5 | 9.5 | 5.6 | 35.4 | 6.7 | 106 | 14.7 | 19.6 |
| 3. Play a sound of shaker from percussion set | 37.3 | 100 | 71.7 | 81.7 | 72.1 | 46.2 | 41.3 | 17.9 | 18.7 | 138 |
| 4. Record sound of drums in file "drums" | 33.7 | 92.1 | 39.1 | 27.1 | 47.8 | 60 | 39.2 | 36.5 | 42.8 | 34.9 |
| 5. Play the file | 7.0 | 15.4 | 6.3 | 3.5 | 4.1 | 43.2 | 3.9 | 4.1 | 38.2 | 5.5 |
| 6. Create a new rhythm using editor | 45.4 | 94.9 | 90.7 | 58.1 | 48.8 | 72.9 | 45.1 | 79.1 | 71.5 | 82.4 |
| 7. Set this rhythm to the piano | 46 | 55.7 | 45.3 | 8.1 | 63.7 | 34.5 | 13.7 | 130 | 53.1 | 56.1 |
| 8. Open the flute help screen | 6.2 | 8.3 | 5.6 | 3.4 | 4.1 | 5.9 | 3.5 | 6.6 | 8.2 | 3.3 |

This test had to run first, because otherwise these people would already be familiar with graphical interface and control and thus would have been significantly distorted by time. Before carrying out a specific task, it was explained and then a tester was asked to realize it. The following diagram (Figure 3) shows the minimum, maximum and average time of testers' performance for each of the eight tasks.

As can be seen from the results, all respondents were able to perform each task under 140 seconds. The tasks also differed in the number of steps that had to be done to properly complete the task. It can also be seen that in addition to the task number three (play a sound of shaker on percussion set) and number six (create a new rhythm using editor) respondents were able to complete each task on average in under one minute. It also can be said that this was a task that most showed deficiencies in the GUI design, because testers usually looked for given functionality on another place or expected more highlighted settings.

The left side of Figure 3 shows which tasks had the biggest difference in the ratio between the minimum and maximum time. Generally speaking, this kind of tasks feel

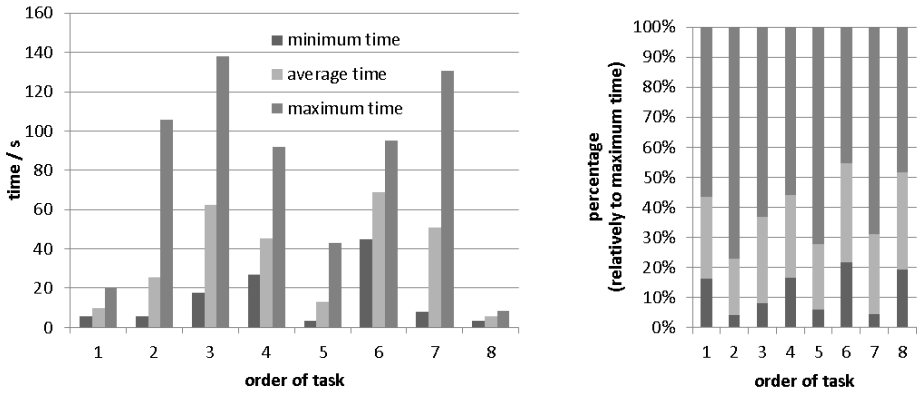



**Fig. 3.** Minimum, maximum and average time of testers' performance

completely natural to certain users (they have either prior experience or good intuition), however other users have considerable problems solving it. An example of such task is task number two, where a tester was asked to play a flute tone. Some respondents did not expect the tool to be controlled by blowing into the microphone, which revealed additional problems of application. Although the testers had no problem to check it in Help, they did not bother to do it. We solved this problem by adding a blowing icon on a flute screen.

Our qualitative test results are relatively good because the measured times showed that the application is fairly intuitive and easy to operate.

### 4.2 Qualitative Testing

The qualitative testing allowed testers to go through the whole application and then they were asked to fill out our questionnaire. Questions were asked about every major applications features (see Table 2).

Before each survey question we first presented to the tester all the (feature-) given functionality and how it can be controlled. Then the tester was free to test the intended functionality, which then he had to evaluate from 1 (worst) to 10 (best). On the next picture you can see a graph (Figure 4) showing the average grade for each question within a questionnaire. In addition, it shows the difference between the highest and lowest grade of the matter.

As can be seen, the average of all questions is located around the values of 7 and 8. This also means that on average respondents were very satisfied with each feature of the application. In this graph we also show the difference between the highest and lowest grades for testing. This difference indicates which questions were more and which less tester-subjective (dependent on the individual user's taste or subjective opinion). The biggest difference was the value 5 in next three questions: evaluation of piano control, flute control and the way of rhythm creation using the editor. On the other side, the lowest difference in combination with high average probably shows the best part of the system (see e.g. question. 9 - graphical design of applications).

**Table 2.** Questions and tester's grades of qualitative testing

| *Question / Tester* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* |
|---|---|---|---|---|---|---|---|---|---|---|
| 1. Piano GUI | 7 | 7 | 7 | 8 | 7 | 8 | 7 | 8 | 8 | 9 |
| 2. Flute GUI | 7 | 9 | 7 | 6 | 6 | 6 | 7 | 8 | 6 | 7 |
| 3. Drums GUI | 8 | 8 | 8 | 7 | 7 | 7 | 9 | 7 | 7 | 9 |
| 4. Piano control | 8 | 8 | 8 | 8 | 9 | 5 | 8 | 9 | 10 | 8 |
| 5. Flute control | 6 | 8 | 9 | 6 | 7 | 9 | 4 | 8 | 8 | 8 |
| 6. Drums control | 9 | 7 | 7 | 8 | 9 | 7 | 8 | 8 | 9 | 10 |
| 7. The way of rhythm creation using editor | 6 | 9 | 7 | 7 | 7 | 7 | 4 | 7 | 8 | 8 |
| 8. Hierarchy of screens | 8 | 7 | 7 | 8 | 6 | 5 | 6 | 8 | 6 | 6 |
| 9. Graphical design of application | 9 | 8 | 8 | 7 | 8 | 7 | 8 | 9 | 9 | 9 |
| 10. Overall application control | 7 | 7 | 7 | 8 | 7 | 6 | 7 | 8 | 7 | 8 |
| 11. The way of combination of interactive and non-interactive instruments | 7 | 9 | 8 | 8 | 8 | 8 | 7 | 7 | 8 | 7 |
| 12. Realization of rhythm adaptation | 9 | 8 | 8 | 7 | 9 | 7 | 8 | 9 | 8 | 6 |

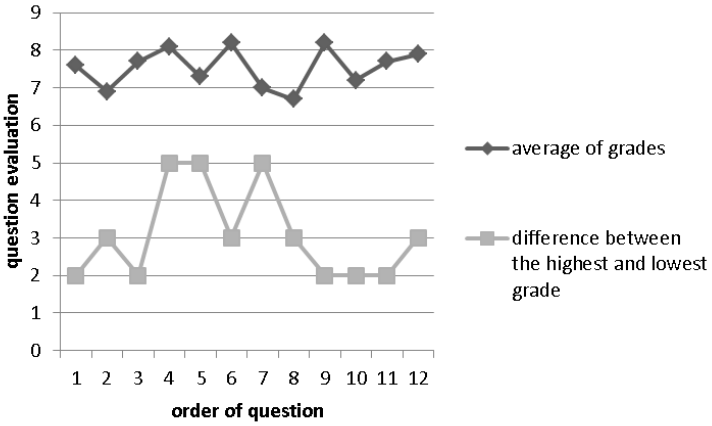



**Fig. 4.** Minimum, maximum and average time of testers' performance

These results indicate that the application can be considered as very good quality including GUI design, control and other features.

### 4.3 Comparative Testing

The last type of testing was based on a comparison of our application with other applications designed for similar purposes. Comparing functionality, we compared all three of our interactive instruments. Moreover, we let our testers to use two other different pianos and ask them to evaluate GUI, control and sounds.

**Flute:** Our Flute was compared with the Ocarina iPhone application, which utility is much more manageable than it is in our case (therefore no other tests provided), but on the other hand we consider our flute more graphically appealing.

**Drums:** Our drums were compared (obvious pros and cons, no other test provided) with applications Amazing Drums and Drums Deluxe Light, which both offer free versions of just ordinary acoustic set (see Figure 5).

**Fig. 5.** Amazing Drums (left) and Drums Deluxe Light (right)

Application Amazing Drums has very unattractive main screen (picture of drums) and the same goes for the sounds of this program. On the other hand, application Drums Deluxe Light creates sounds very similar to acoustic drums and the main screen also resembles the real drum set. The downside of this application (comparing to ours) is, that it does not show the visual feedback when played on any part of the drum set. But its advantage is that it offers a number of component sounds of the same drum that plays in dependence of touched space (e.g. in the middle of the drum or on the side of it). Both compared application can be controlled only by touch-screen.

**Piano:** Comparison of our piano program was conducted with applications Piano+ and Piano Virtuoso, both of which are available free on the App Store. Both applications provide only the functionality to play this instrument. The visual of their main screens shows Figure 6:


**Fig. 6.** Piano + (left) and Virtuoso (right) application

The comparison of our piano with two other applications, we proceed similarly as in the qualitative tests. Since we were not able to provide the absolute correctness of this test (testers and developer knew personally each other and testers have already completed two previous tests), we tried to at least establish the conditions that would minimize the impact on incorrect evaluation. Therefore we first demonstrated (functionality and control) all three applications to the testers. Then they were asked to play a

few of chords and tones in each application (to make the comparison was the most relevant). When testing the application, they have also highlighted the three main attributes (graphical user interface, control and sounds) that were evaluated in each application by grades from 1 (completely wrong) to 10 (excellent).

The testers' grades and questions are in Table 3. The average evaluation of these attributes for all three applications can be seen on Figure 7.

**Table 3.** Questions and tester's grades of qualitative testing

| *Question / Tester* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* |
|---|---|---|---|---|---|---|---|---|---|---|
| Our piano GUI | 7 | 7 | 7 | 8 | 7 | 8 | 7 | 7 | 8 | 9 |
| Piano+ GUI | 5 | 9 | 4 | 5 | 3 | 5 | 3 | 7 | 3 | 6 |
| Virtuoso GIO | 8 | 6 | 8 | 6 | 7 | 3 | 8 | 9 | 6 | 7 |
| Our piano control | 8 | 8 | 8 | 8 | 9 | 5 | 8 | 9 | 10 | 8 |
| Piano+ control | 6 | 5 | 4 | 5 | 5 | 7 | 3 | 3 | 2 | 4 |
| Virtuoso control | 8 | 6 | 6 | 7 | 6 | 5 | 8 | 6 | 7 | 6 |
| Our piano sounds | 9 | 8 | 8 | 8 | 9 | 8 | 7 | 7 | 7 | 8 |
| Piano+ sounds | 3 | 6 | 5 | 6 | 5 | 2 | 3 | 4 | 3 | 6 |
| Virtuoso sounds | 7 | 6 | 8 | 7 | 7 | 5 | 7 | 8 | 8 | 8 |

Figure 7 shows average grade for each of the tested feature in each of tested piano. As can be seen, our piano received the highest grades (in all three features) from our testers. In contrast, Piano+ with a relatively great distance determined in all features for the worst. Virtuoso received relatively similar grades as our piano; the noticeable difference is only in control.

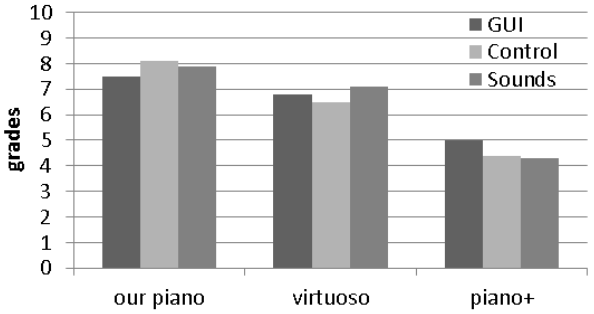



**Fig. 7.** Average evaluations of GUI, control and sounds in our piano, Piano+ a Virtuoso

These test results show that our instruments were created in a similar or better quality as well as in compared applications. The main difference still remains – our application combine more musical instruments and offer even rhythm adaptation.

## 5 Conclusions

In this paper we described our research which consists of study how mobile device can be used as a smart virtual music instrument. We proposed several improvements for three different types of musical instruments and implemented them (each is controlled by variety of sensors), with which we have conducted experiments. In addition, we focused on design and implementation of non-interactive instruments in which we had successfully implemented our formula for automatic tempo adjustment.

Our application was tested by ten users and three types of tests. From quantitative test (time of tasks performance) we realized some design errors but they were not critical (to improve intuitiveness, highlighting of some of the graphic elements or their move them to other place is needed). The qualitative test consists of a questionnaire covering all important parts of the system features. These were in average graded to be very good. At the end of testing, we asked testers to compare our piano with two other applications. Here our piano seems to have the best GUI, control and sounds. This is a very good result considering that those other two applications were designed only to play this one instrument.

It is crucial to note that during the test, the parts of the system that we have not found in similar types of applications were graded as excellent e.g., rate adaptation, a combination of two types of instruments or multiple music instruments within one application. Therefore, the quality of our application can be considered as equivalent to other existing applications or even comparably better thanks to its unique features including tempo adaptation.

This application could be used for entertainment or also as a simple educational musical instrument application.

**Acknowledgement.** This work was partially supported by the grants VG1/0971/11 and APVV 0208-10.

## References

1. Essl, G., Rohs, M.: Interactivity for Mobile Music-Making. Organised Sound 14(2), 197–207 (2009)
2. Wang, G.: Designing Smule's iPhone Ocarina. In: Proceedings of the 2009 International Conference on New Interfaces for Musical Expression, Pittsburgh, Pennsylvania (2009)
3. Sauer, D., Yang, Y.-H.: Music-Driven Character Animation. ACM Transactions on Multimedia Computing, Communications, and Applications 5(4), 27:1–27:16 (2009)
4. Goto, M., Muraoka, Y.: Real-time beat tracking for drumless audio signals: Chord change detection for musical decisions. Speech Commun. 27(3-4), 311–335 (1999)
5. Tzanetakis, G., Essl, G., Cook, P.: Audio Analysis using the Discrete Wavelet Transform. In: Proceeding Conference in Acoustics and Music Theory Applications, AMTA 2001, Kiathos, Greece, pp. 318–323. Priceton University Press (2001)
6. Rohs, M., Essl, G., Roth, M.: CaMus: Live Music Performance using Camera Phones and Visual Grid Tracking. In: Proceedings of the 2006 International Conference on New Interfaces for Musical Expression, NIME 2006, IRCAM - Centre Pompidou, pp. 31–36 (2006)
7. Reidsma, D., Nijholt, A., Bos, P.: Temporal interaction between an artificial orchestra conductor and human musicians. Comput. Entertain. 6(4), Article 53, 22 p. (2008)

# Dynamic Voting Interface in Social Media: Does it Affect Individual Votes?

Michail Tsikerdekis

Masaryk University, Faculty of Informatics, Brno, Czech Republic
tsikerdekis@gmail.com

**Abstract.** The rise in popularity of social media along with new web technologies has presented designers and developers with tremendous new interface opportunities for evaluating user-generated content. One of these new interface designs found in social media today, is the dynamic voting interface; voting results are public from the initiation of an evaluation procedure and are constantly being updated. However, it is currently unclear on whether these interfaces affect the outcome of a voting process and to what degree. This study employed a mixed methods survey as an attempt to try and answer these questions and provide exploratory evidence for the effects of dynamic voting interfaces on social media communities. Findings coming from this study are able to provide support for the "no effect" hypothesis.

**Keywords:** dynamic, voting, interface, social, media, design.

## 1 Introduction

For every group that is collaborating online, the time to decide and reach consensus may be the most crucial step in a decision-making scenario. In turn, this helps to finalize a group's position on a topic. Software designers and developers that produce Group Decision Support Systems (GDSSs) should always take into account the different voting mechanisms that can be developed for a community. The implications of voting mechanisms in GDSSs are numerous and each has a potential to affect the outcome of a decision making process [1]. A trend that was seen in recent years with the development of social media is voting while results remain publicly available and are interactively evolving as new votes come in. This feature of a voting design can be defined as a *dynamic voting interface*. This is something that became popular along with freshly designed mechanisms such as the Social Decision Support Systems [2], or Delphi voting structures [3] that became more possible due to the nature of computer-mediated communication systems. However, there have been no studies that investigated what effect a dynamic voting interface has had to an outcome of a voting process.

At its simplest form today, this voting process takes a form of social buttons in social media which in turn play a decisive role in what information is shared. These features essentially created a like economy on the Web [4]. While arguably, these voting processes are not contributing to critical decisions as the ones made in more business-oriented GDSSs, the underlying processes remain the same. This makes them particularly interesting to study because of the fact that at their purest and simplest form, one

P. van Emde Boas et al. (Eds.): SOFSEM 2013, LNCS 7741, pp. 552–563, 2013.

can observe if they indeed can affect voting opinions. In addition, one can also investigate what is the power of the effect. If they do affect individual votes, then this may be in clear violation of one of the voting standards that requires people to freely express their true voting preference [5].

This study attempted to assert if dynamic voting interfaces in social media can affect the outcome of a voting process. Having this knowledge, software engineers should pay close attention when implementing voting mechanisms. In addition, I follow a different approach in evaluating results from HCI research by shifting away from traditional null-hypothesis statistical testing (NHST), and apply a Bayesian hypothesis testing analysis instead. Bayesian methods were preferred over NHST since this study did not necessarily want to assert that there is a dependency between experimental conditions and the variables involved. As such, while in NHST "p-values are incapable of providing support for the null hypothesis" [6], Bayesian hypothesis testing does quantify the probability that the null hypothesis is true [6, 7]. Further, Bayesian statistical analysis "enables us to distinguish between cases where the data is inconclusive [. . . ] and cases where there is strong evidence regarding the null hypothesis" such as when sample size is inadequate [7]. Additional qualitative data was obtained in order to increase internal validity [8].

## 2 Theoretical Background

Social media software includes a variety of classifications such as blogs, collaborative projects, social networking sites, virtual social worlds, virtual game worlds, content communities and even micro blogging [9, 10]. Some of the collaborative features have connections with GDSSs which cover a broad spectrum of tools and are generally defined as collaboration technologies, designed to support meetings and group work [11]. There has been an extensive amount of research on GDSSs [12–17]. There are many gains for online collaborators such as; more precise communication, members are empowered to build on the ideas of others, and a more objective evaluation of ideas [18]. In addition, distance stops becoming a factor which in turn could give rise for a new GDSSs type, distributed by group support systems [19]. The outcome from an online collaborating process is usually extracted by a final decision-making voting process.

Voting mechanisms have been described in literature before the advent of information technologies [20, 21]. However, GDSSs provided brand new opportunities and unknown implications in the design and effects of the new voting mechanisms [1]. A number of factors can affect the outcome of a voting process such as the decision on one of the numerous voting criteria [5, 22] or even the option of anonymity that may or may not be provided to individuals [23]. Studies have also been developed for integrating voting techniques in GDSSs [1]. Most of the studies found online revolved around internet voting. However, research on internet voting for GDSSs research has been limited [5].

Internet voting comes with benefits and disadvantages. Apparent convenience, 24-hour availability over several days, and the ability of Internet voting to be unaffected by traffic and weather issues make this to be tremendously advantageous over traditional voting methods [24]. On the other hand, there are a couple of potential problems that

rise with internet voting such as ”the identification of users, the security and the reliability of the voting system, the accessibility of the Internet, privacy issues, and voter training” [5]. Regardless, some argue that with internet voting, “the majority of citizens will eventually start voting in the way that they book their holidays nowadays.” [25].

Today, online voting is not just restricted for elections but for evaluation of information such as user-generated content. Rated content along with social bookmarking features can also be extremely helpful for search engines for finding content that is currently unreachable [26]. Additionally, modern techniques that allow for dynamic voting status interfaces are also becoming popular with a suspicion that this may influence users to vote [27]. As such, it’s not hard to imagine that it may as well, have the power to influence individual votes.

## 3 Hypotheses

One of the most popular websites today, YouTube, uses a dynamic voting interface in order to evaluate the content being uploaded by users, which is currently at an uploading rate of 60 hours’ worth of video, per every one minute [28]. Video can also be evaluated by users who are responsible for the 4 billion views of videos every day. This makes YouTube a website that puts the design of a dynamic voting interface to the test every day. In addition, the simplistic nature of a vote, liking or disliking a video, is ideal for discovering if such a voting design has any influence on how many individuals vote.

YouTube’s design uses visuals in order to show whether most votes were in favor of a video or against as well as text. However, it is not clear if users are actually using both the visual and textual information when voting, or if even they use any of the information at all. In literature, it is known that people in general do not read but scan pages [29] but this was never studied in terms of a voting process. As such it was an additional opportunity to determine on whether visuals or text have power (if any) on the way individuals vote. Two hypotheses were formulated.

**$H_1$:** There is a dependency between the votes presented in text by a dynamic voting interface and the way individuals vote.

**$H_2$:** There is a dependency between the votes presented in text and graphically by a dynamic voting interface and the way individuals vote.

However, as this study used a Bayesian framework to evaluate results, corresponding null hypotheses that claim independence are just as likely.

## 4 Research Design

### 4.1 Survey

In order to evaluate the likelihood of the above hypotheses, I created and administered a survey. This method provides standardization for collecting similar data from groups [30]. The type of survey was a randomized controlled trial “in which participants are allocated truly randomly to an experimental group and a control group” which strengthens the claims for internal validity [31].

For this study, two videos that already existed and had already been evaluated on YouTube by users were selected in order to form two survey parts. This established a baseline for the videos, namely if people perceived them as likeable or not. One video was likeable by most of those that watched it and rated it, and the second was disliked by the majority of the people that rated it.


**Fig. 1.** Dynamic voting interface used for the two treatments for the second video. On the left is treatment A and on the right treatment B.

However, these two videos alone would not provide sufficient evidence for the influence of a dynamic voting interface. Two variations of these videos were produced, with everything remaining identical except the results of the vote indicator. These were reversed for each treatment. Hence, an individual may have viewed the likeable first video while the voting results favored the video, while another may have viewed the video but the voting indicator did not favor the video. The same was applied for the second video. Examples of these two treatments for the second video are shown on Figure 1 and a summary of the treatments of the experimental design can be seen on Table 1. The extreme proportions between likes and dislikes were established in order to investigate the maximum potential for influencing votes. Participants were allocated in each treatment randomly based on automated algorithms.

**Table 1.** Treatments formed by the survey's experimental design

| **Experimental Condition** | **First Video (Originally favored on YouTube)** | **Second Video (Originally not favored on YouTube)** |
|---|---|---|
| Treatment A | Votes favor video both graphically and in text | Votes favor video both graphically and in text |
| Treatment B | Votes favor video only graphically, text does not favor video | Votes do not favor video graphically or in text |

An additional alteration was made in the survey for the first video. While the text that showed the results based on the likes and dislikes was reversed between treatments, the color indicator on top of the text always remained green, as opposed to changing to red which would indicate more negative votes. This was made in order to assert the

$H_1$ hypothesis. If most individuals scanned the indicator and never read the text, it was expected that the two treatments for the first video would have produced virtually identical results. On the other hand, the second video was expected to produce significant differences between the two treatments if indeed dynamic voting interface had an effect on individual voting. Figure 2 depicts the two treatments for the first video.

Moreover, further demographic information was obtained by the participants with an additional question that asked them if they have ever watched a video on Youtube. This would help assert users were familiar with the process. Finally, a qualitative question was added in order to determine the motives for people's votes ("Why did you like or dislike the video? Please elaborate.").

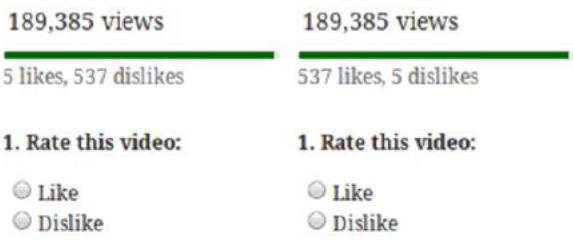



**Fig. 2.** Dynamic voting interface used in the two treatments for the first video. On the left is treatment B and on the right treatment A.

### 4.2 Context

The survey was administered through a Facebook application, and invitations for the survey were circulated in several academic and non-academic groups and fan pages on the social networking service. In addition, a snowball sampling method was employed through allowing users to invite their friends in order to reach farther users within the network. This method of using a combination of a volunteer and snowball sampling technique has proven to be relatively successful in a previous study [32] and in theory, as long as people are willing to invite others and participate in the survey, one can reach any node within a social network based on the theory of six degrees of separation [29, 33]. Moreover, the survey was developed in English but also translated further into other languages such as Greek and Spanish, and the language was detected automatically and adapted to the survey depending on the user's profile settings. The survey took place during the period of February 15th to March 19th, 2012.

## 5 Results

In total, 123 people visited the application during the survey. Out of the 123 that visited the application, 89 went through the whole survey. Most of the participants who did not complete the survey abandoned it in the first page. I then decided that the 89 participants will effectively become the sample that will be used for further analysis since the individual responses to the survey contained complete results without any missing data.

The sample's age groups were: 13-17 (2.2%), 18-25 (52.8%), 26-39 (33.7%), and 40-59 (11.2%). No respondents reported that their age group was 60 and above. The sample contained an almost equal amount of male (50.6%) and female (48.3%) respondents while 1.1% of the sample refused to report their sex. The majority of respondents came from Europe (75.3%), followed by South America (12.4%), North America (7.9%), Africa (2.2%) and Asia (1.1%). One point one percent (1.1%) of respondents chose not to disclose their location. In addition, all participants in the survey stated that they had watched a video on YouTube at some point in the past. This was an indication that the participants were familiar with the service.

### 5.1 Bayesian Hypothesis Testing

While the purpose of this paper is not to educate on Bayesian methods, a brief introduction is given here since it is rarely seen in HCI research [7].

At the core of all Bayesian statistics lies the idea of *Bayes' rule*. Having a prior belief and given a set of data, one can obtain a posterior belief [34]. This idea can be expanded further by establishing various models that produce different posterior probabilities for our data.

Having multiple models or hypotheses is particularly useful when one wants to be able to instantly state which model is more probable given the evidence. To do this, the ratio of marginal likelihoods or else the *Bayes factor* is used [35].

Before seeing the data one has a ratio of prior beliefs about the two models which are usually given an equal probability [36]. Priors should always be chosen in Bayesian statistics, and their value depends on two point of views, namely subjectivist and objectivist, with the latter having the need to satisfy requirements of rationality and consistency [37, 38]. An uninformative prior favors objectivity by asserting that no model is perceived as better than the other. On the other hand, an informative prior values the researcher's prior knowledge.

After seeing the data, posterior odds for the two models are obtained. As a result, the ratio of those posterior and prior odds for the two models produces the odds of one model being more likely to occur than the other.

There is some resemblance between a Bayes Factor and the significance value in NHST. However, since this represents a ratio it becomes the odds of one model being likely against another; a claim that cannot be made for the p-value obtained by NHST. Finally, by containing the odds for a likelihood of a model, Bayes Factor is perceived as the weight of evidence coming from the data [39].

### 5.2 Bayesian Analysis

The experimental treatments formed two two-way contingency tables. These are presented on Tables 2 and 3.

One approach for conducting Bayesian hypothesis testing for this data is by creating a model in software and then directly sample from the posterior distribution of interest using algorithms based on the computer-driven sampling methodology known as Markov chain Monte Carlo. To do this, a combination of the statistical computing

**Table 2.** Contingency table for the first video

| Treatment/Response | Like | Dislike | Total |
|---|---|---|---|
| Like treatment | 38 | 2 | 40 |
| -percentage within treatment | 95% | 5% | 100% |
| -percentage within response | 45.8% | 33.3% | 44.9% |
| Dislike treatment | 45 | 4 | 49 |
| -percentage within treatment | 91.8% | 8.2% | 100% |
| -percentage within response | 54.2% | 66.7% | 55.1% |
| Total | 83 | 6 | 89 |
| -percentage within treatment | 93.3% | 6.7% | 100% |
| -percentage within response | 100% | 100% | 100% |

**Table 3.** Contingency table for the second video

| Treatment/Response | Like | Dislike | Total |
|---|---|---|---|
| Like treatment | 6 | 40 | 46 |
| -percentage within treatment | 13% | 87% | 100% |
| -percentage within response | 40% | 54.1% | 51.7% |
| Dislike treatment | 9 | 34 | 43 |
| -percentage within treatment | 20.9% | 79.1% | 100% |
| -percentage within response | 60% | 45.9% | 48.3% |
| Total | 15 | 74 | 89 |
| -percentage within treatment | 16.9% | 83.1% | 100% |
| -percentage within response | 100% | 100% | 100% |

software R and OpenBUGS was used for this study. OpenBUGS is an open source software with similar functionality to WinBUGS, a general-purpose program that facilitates Bayesian analysis for statistical models.

For the purposes of this study, an already established model for binomial hypothesis testing was used [40]. In short, each treatment is perceived as a binomial distribution and a model is built in order to establish if there is an equality of proportions. Each binomial distribution is being affected by the rate parameters $\theta_1$ and $\theta_2$. Their difference $\delta$, and its posterior distribution provides information which helps in obtaining the Bayes factor at the point $\delta$=0. This is the point where the differences between the two treatments eliminate each other. Put simply, the point at which the proportions of treatments A and B are absolutely equal. Additionally, the prior chosen for both $\theta$ parameters is uninformative. This was selected in order to remove subjectivity in the overall process when selecting an otherwise ”informative” prior.

Using R and OpenBUGS, sampling procedures were employed with 50,000 iterations and a burnin of 5,000 samples while thinning was set to 1. A good description

of the background process and algorithms can be found in most-recent Bayesian books [34,36,40,41]. The outcome of the prior and posterior distribution for δ can be seen on Figure 3. The dots mark the prior and posterior points at $\delta = 0$, the point of the null hypothesis where the differences become 0. It is visually evident that at the specific point where $\delta = 0$, the belief is reinforced. Put simply, the posterior belief is higher than the prior belief. The Bayes Factor in support of the null hypothesis $BF_{01}$ is showing that given the data, the null hypothesis of independence ($H_0$) is 6.51 times more likely than the dependence hypothesis ($H_1$ where $\delta \neq 0$). This provides a substantial strength of evidence in favor of the null hypothesis [6, 35].

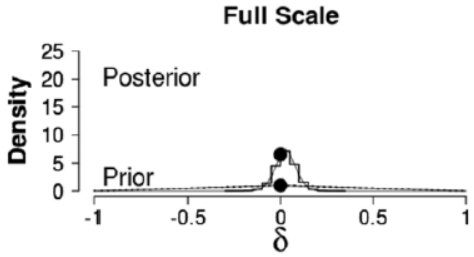



**Fig. 3.** Prior and posterior distribution for the first video based on the Bayesian Binomial model

Applying the same procedure for the second video is also illuminating. The Bayes factor in support of the null hypothesis $BF_{02}$ is showing that given the data the null hypothesis of independence ($H_0$) is 3.20 times more likely than the dependence hypothesis ($H_2$ where $\delta \neq 0$). In terms of strength of evidence, this is still substantial [6, 35]. Prior and posterior distribution plots are also shown for this analysis on Figure 4.

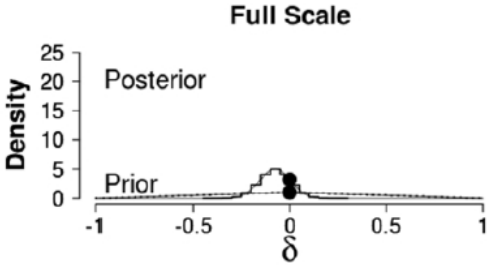



**Fig. 4.** Prior and posterior distribution for the second video based on the Bayesian Binomial model

### 5.3 Qualitative Data

Qualitative questions for both videos shed a light over the behaviors of the respondents seen in the quantitative analysis. All responses within the sample had an emotional or

logical basis. For example, a respondent for the first video, under treatment B voted against it and stated that it "Shows aggression and I dont [sic] like that." Coincidentally, the same participant also voted against the second video when under treatment A, and the statement was similar along the lines that it "shows aggression also and I dont [sic] like." This demonstrates that the respondent demonstrated a consistency in his or her opinion and in the way that he or she voted regardless of the treatment.

Similarly, two respondents under treatment A for the first video voted against it stating that "There was no meaning for me" and that it was "pointless." Others rationalized their choice to an impressive degree. The following respondent voted in favor of the first video under treatment A: "1- It contained Ice [sic] and water and this is something i don't see too much 2- It talked about travel and this is something I like 3- It show that a smart idea , which is unity is power." Finally, one respondent also managed to surprise by his or her power of observation. Under treatment B for the first video while voting favorably he or she wrote: "Funny commercial. I don't get why it got so many dislikes though. People must hate the bus." Against all odds, while the expectation of this study was that people scan pages without reading text, this respondent noticed that text results were not favoring the video. While the results produced for the first video between treatments were nearly identical, this qualitative find may suggest that some individuals do pay attention to textual indications for votes but still vote independently of what others voted.

Patterns similar to the ones found for the first video also exist for the second video. "i [sic] hate these orange clips" stated one respondent under treatment A while voting against the video. Another respondent stated that "It didn't have any useful information, material nor artistic value in it :) [sic]." However, cases also existed where people voted in favor of the second video although under treatment B. One respondent stated "it [sic] was okay. i like attempts at freakiness, so it worked for me." The detailed response dismisses any considerations that treatment B had any effect on the favorable vote of this individual.

## 6 Discussion

Quantitative results coming from the first video show that users are substantially ignoring textual information when voting. Evidence was against the $H_1$ which seems to indicate that users seem to rely more on graphical representations of votes instead of text. This may be positive evidence for demonstrating that people scan pages rather than reading them. Moreover, the outcome from the voting for both treatments show that the majority of respondents agreed with the actual original estimate that exists on YouTube for the first video.

The second video showed a similar support for the null hypothesis. Findings showed no effect between treatments and no change in the overall outcome of the voting process. Most users in the survey regardless of treatment found the video unlikeable which is similar to the original estimate from the YouTube community.

Finally, qualitative finds seem to be in agreement with quantitative results. Put simply, people acted in the way they thought best and provided justifications that were in accordance with their decisions. While this factor still cannot possibly assert that there

is not an underlying process in which judgment is distorted, the body of work coming from this study seems to point to a clear answer; dynamic voting interface as seen on YouTube does not affect the evaluation of user generated content. In terms of YouTube, it means that when a video has many negative or positive votes, one can be fairly certain, these votes where independent of the dynamic voting interface being present in its design.

### 6.1 Recommendations

Based on the evidence provided by the study, it is recommended for software developers and designers to consider their choices of using a dynamic voting interface. There are positive indications that a voting indicator would not affect the outcome of a voting process. However, while it is rendered safe, more evidence needs to be brought to light in order to rule out any definitive effect on the majority outcome or even individual votes.

Additionally, this study has also found a strong connection between graphical indicators and individual votes. A safer approach for a dynamic voting interface may be to include just textual information on the votes instead of graphical. However, qualitative finds seem to indicate that some individuals may still pay strong attention to other individual votes.

### 6.2 Limitations

This study investigated a voting process taken from a real social media implementation of a dynamic voting interface. However, results may have been different in a more complicated voting process where the choices may have been more. In addition, experimental design surveys are not as efficient in demonstrating impulsive action in voting procedures that may occur in real life scenarios.

## 7 Conclusion

Human-computer interaction research has investigated many aspects of online collaboration but research into online voting processes is still limited. In addition, most of the evaluations rely on traditional statistical procedures, which arguably have several weaknesses compared to Bayesian statistics. This study not only demonstrated that Bayesian hypothesis testing can be easily applied in HCI research, but has also established this fact by conducting novel research in a previously unexplored aspect of online collaboration. In addition, these results paint a promising picture for users who collaborate online and for software engineers who consider using an dynamic voting interfaces for their communities.

## References


1. Gavish, B., Gerdes, J.H.: Voting mechanisms and their implications in a GDSS environment. Annals of Operations Research 71, 41–74 (1997)
2. Turoff, M., Hiltz, S.R., Cho, H., Li, Z., Wang, Y.: Social Decision Support Systems (SDSS). In: Proceedings of the 35th Hawaii International Conference on System Sciences, pp. 1–10 (2002)
3. Hiltz, S.R., Turoff, M.: Structuring computer-mediated communication systems to avoid information overload. Communications of the ACM 28, 680–689 (1985)
4. Gerlitz, C., Helmond, A.: Hit, Link, Like and Share. Organizing the social and the fabric of the web in a Like economy. In: DMI Mini Conference University of Amsterdam, pp. 24–25 (2011)
5. Li, Z.: Design and evaluation of a voting tool in a collaborative environment (2003)
6. Wagenmakers, E.-J., Lee, M.D., Lodewyckx, T., Iverson, G.: Bayesian versus frequentist inference. In: Hoijtink, H., Klugkist, I., Boelen, P.A. (eds.) Bayesian Evaluation of Informative Hypotheses in Psychology, pp. 181–207. Springer, New York (2008)
7. Kaptein, M., Robertson, J.: Rethinking statistical analysis methods for CHI. In: Proceedings of the 2012 ACM Annual Conference on Human Factors in Computing Systems, CHI 2012, p. 1105 (2012)
8. Bryman, A.: Social Research Methods. Oxford University Press, Oxford (2012)
9. Kaplan, A.M., Haenlein, M.: Users of the world, unite! The challenges and opportunities of Social Media. Business Horizons 53, 59–68 (2010)
10. Kaplan, A.M., Haenlein, M.: The early bird catches the news: Nine things you should know about micro-blogging. Business Horizons 54, 105–113 (2011)
11. Dennis, A.R., George, J.F., Jessup, L.M., Nunamaker, J.F., Vogel, D.R.: Information technology to support electronic meetings. MIS Quarterly 12, 591–624 (1988)
12. Dufner, D., Hiltz, S.R., Turoff, M.: Distributed group support: a preliminary analysis of the effects of the use of voting tools and sequential procedures. In: Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences. Information Systems: Collaboration Technology Organizational Systems and Technology, vol. IV, pp. 114–123. IEEE (1994)
13. Fjermestad, J.: An Integrated Framework for Group Support Systems. Journal of Organizational Computing and Electronic Commerce 8, 83–107 (1998)
14. Ki Jeong, C., Hung Kook, P.: Examining the conflicting results of GDSS research. Information Management 33, 313–325 (1998)
15. Li, Z., Cheng, K.-E., Wang, Y., Hiltz, S.R.: Thurstone's Law of Comparative Judgment for Group Support. In: AMCIS 2001 Proceedings, Paper 48 (2001)
16. Pervan, G.P., Atkinson, D.J.: GDSS research: An overview and historical analysis. Group Decision and Negotiation 4, 475–483 (1995)
17. Seibold, D.R., Heller, M.A., Contractor, N.S.: Group decision support systems (GDSS): Review, taxonomy, and research agenda. In: Kovačič, B. (ed.) New Approaches to Organizational Communication, pp. 143–168. SUNY Press (1994)
18. Nunamaker, J.F., Dennis, A.R., Valacich, J.S., Vogel, D., George, J.F.: Electronic meeting systems. Communications of the ACM 34, 40–61 (1991)
19. Turoff, M., Hiltz, S.R., Bahgat, A.N.F., Rana, A.R.: Distributed Group Support Systems. MIS Quarterly 17, 399–417 (1993)
20. Dummett, M.A.E.: Voting Procedures. Clarendon Press (1984)
21. Felsenthal, D.S.: Topics in social choice: sophisticated voting, efficacy, and proportional representation. Praeger (1990)

22. Brams, S.J.: Alternative Voting Systems. In: Maisel, L.S. (ed.) Political Pareties and Elections in the United States: An Encyclopedia, pp. 23–31. Garland, New York (1991)
23. Hiltz, S.R., Turoff, M., Johnson, K.: Experiments in group decision making, 3: disinhibition, deindividuation, and group process in pen name and real name computer conferences. Decision Support Systems 5, 217–232 (1989)
24. Hoffman, L.J.: Internet voting: will it spur or corrupt democracy? In: Proceedings of the Tenth Conference on Computers, Freedom and Privacy: Challenging the Assumptions, pp. 219–223. ACM, New York (2000)
25. Ladner, A., Pianzola, J.: Do Voting Advice Applications Have an Effect on Electoral Participation and Voter Turnout? Evidence from the 2007 Swiss Federal Elections. In: Tambouris, E., Macintosh, A., Glassey, O. (eds.) ePart 2010. LNCS, vol. 6229, pp. 211–224. Springer, Heidelberg (2010)
26. Heymann, P., Koutrika, G., Garcia-Molina, H.: Can social bookmarking improve web search? In: Proceedings of the International Conference on Web Search and Web Data Mining, WSDM 2008, vol. 301, p. 195 (2008)
27. Turnbull, D.: Rating, voting & ranking: designing for collaboration & consensus. In: CHI 2007 Extended Abstracts on Human Factors in Computing Systems, pp. 2705–2710 (2007)
28. Youtube: Statistics, http://www.youtube.com/static?template=pressstatistics
29. Zhang, L., Tu, W.: Six Degrees of Separation in Online Society. Distribution 3, 1–5 (2006)
30. Wright, J.D., Marsden, P.V.: Survey Research and Social Science: History, Current Practice, and Future Prospects. In: Wright, J.D., Marsden, P.V. (eds.) Handbook of Survey Research, pp. 3–26. Emerald Group Publishing, Bingley (2010)
31. Lewin, C.: Understanding and Describing Quantitative Data. In: Somekh, B., Lewin, C. (eds.) Research Methods in the Social Sciences, pp. 220–230. Sage, London (2005)
32. Tsikerdekis, M.: The choice of complete anonymity versus pseudonymity for aggression online. eMinds International Journal on Human-Computer Interaction 2, 35–57 (2012)
33. Watts, D.J., Dodds, P.S., Newman, M.E.J.: Identity and Search in Social Networks. Science 296, 1302–1305 (2002)
34. Jackman, S.: Bayesian Analysis for the Social Sciences. Wiley, Hoboken (2009)
35. Jeffreys, H.: Theory of Probability. Clarendon Press (1998)
36. Kruschke, J.K.: Doing Bayesian Data Analysis: A Tutorial with R and BUGS. Academic Press (2010)
37. Cox, R.T.: The Algebra of Probable Inference. American Journal of Physics 31, 66 (1963)
38. Jaynes, E.T.: Bayesian methods: general background. In: Justice, J.H. (ed.) Maximum Entropy and Bayesian Methods in Applied Statistics, pp. 1–25. Cambridge University Press (1986)
39. Good, I.J.: Weight of Evidence: A Brief Survey. In: Bernardo, J.M., DeGroot, M.H., Lindley, D.V., Smith, A.F.M. (eds.) Bayesian Statistics, pp. 249–269. Elsevier, New York (1985)
40. Lee, M.D., Wagenmakers, E.-J.: A Course in Bayesian Graphical Modeling for Cognitive Science. Convergence 7 (2010)
41. Albert, J.: Bayesian Computation with R. Springer, New York (2009)

# Author Index