LNCS 6543

Ivana Černá Tibor Gyimóthy
Juraj Hromkovič Keith Jeffery
Rastislav Královič Marko Vukolić
Stefan Wolf (Eds.)

SOFSEM 2011: Theory and Practice of Computer Science

37th Conference on Current Trends in Theory and Practice of Computer Science
Nový Smokovec, Slovakia, January 2011, Proceedings

Springer

Ivana Černá   Tibor Gyimóthy
Juraj Hromkovič   Keith Jeffery
Rastislav Královič   Marko Vukolić
Stefan Wolf (Eds.)

SOFSEM 2011: Theory and Practice of Computer Science

37th Conference on Current Trends
in Theory and Practice of Computer Science
Nový Smokovec, Slovakia, January 22-28, 2011
Proceedings

Springer

Volume Editors

Ivana Černá
Masaryk University, Faculty of Informatics, Department of Computer Science
Botanicka 68 a, 602 00 Brno, Czech Republic, E-mail: cerna@fi.muni.cz

Tibor Gyimóthy
University of Szeged, Department of Software Engineering
Árpád tér 2., 6720 Szeged, Hungary, E-mail: gyimothy@inf.u-szeged.hu

Juraj Hromkovič
ETH Zürich, Informationstechnologie und Ausbildung, CAB F 16, F 13.1
Universitätstr. 6, 8092 Zürich, Switzerland, E-mail: juraj.hromkovic@inf.ethz.ch

Keith Jeffery
Science and Technology Facilities Council, Rutherford Appleton Laboratory
Harwell Science and Innovation Campus, Didcot, OXON OX11 0QX, UK
E-mail: keith.jeffery@stfc.ac.uk

Rastislav Královič
Comenius University, Department of Computer Science
84248 Bratislava, Slovakia, E-mail: kralovic@dcs.fmph.uniba.sk

Marko Vukolić
EURECOM, 2229 Route des Crêtes, BP 193, 06904 Sophia Antipolis cedex, France
E-mail: Marko.Vukolic@eurecom.fr

Stefan Wolf
ETH Zürich, Institute of Theoretical Computer Science, IFW E49.1
Haldeneggsteig 4, 8092 Zürich, Switzerland, E-mail: wolfst@inf.ethz.ch

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-18380-5 e-ISBN 978-3-642-18381-2
DOI 10.1007/978-3-642-18381-2
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2010942599

CR Subject Classification (1998): F.2, F.1, D.2, H.2-3, C.2, H.4, D.4, G.2.1

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

The 37th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2011, was held during January 22–28, 2011 in the Hotel Atrium, Nový Smokovec, of the Tatra Mountains of Slovakia. This volume contains 5 of the invited lectures and 41 contributed papers selected for presentation at the conference. The contributed papers were selected by the Program Committee out of a total of 122 submissions.

SOFSEM (originally SOFtware SEMinar) is an annual international winter conference devoted to the theory and practice of computer science. Its aim is to present the latest developments in research for professionals from academia and industry, working in leading areas of computer science. As a well-established and fully international conference, SOFSEM maintains the best of its original Winter School aspects, such as a high number of invited talks, in-depth coverage of selected research areas, and ample opportunities to discuss and exchange new ideas. SOFSEM 2011 was organized around the following four tracks:

– Foundations of Computer Science (Chair Ivana Černá)
– Software, Systems, and Services (Chair Tibor Gyimothy)
– Processing Large Datasets (Chair Keith Jeffery)
– Cryptography, Security and Trust (Chairs Stefan Wolf and Marko Vukolic)

An integral part of SOFSEM 2011 was the traditional Student Research Forum (Chair Mária Bieliková) organized with the aim to give students feedback on both the originality of their scientific results and on their work in progress. The papers presented at the Student Research Forum were published in local proceedings.

SOFSEM 2011 was organized by the Faculty of Mathematics, Physics and Informatics of the Comenius University in Bratislava, and the Slovak Society for Computer Science with support from the Czech Society of Cybernetics and Informatics.

SOFSEM 2011 added a new page to the tradition of SOFSEM dating back to 1974, which was possible due to the effort of many people. As editors of these proceedings, we are grateful to everyone who contributed to the scientific program of the conference. We would like to thank the invited speakers Christian Cachin, Markus Gross, Juerg Gutknecht, Tony Hey, Rainer Koschke, Ulrich Rührmair, Vitaly Shmatikov, Jiri Srba, and Tomas Vojnar for presenting their work to the audience of SOFSEM 2011. We thank all authors who submitted their papers for consideration. Many thanks go to the Program Committee, and to all external referees, for their hard work in evaluating the papers. The work of the Program Committee was carried out using the EasyChair system, and we gratefully acknowledge this contribution. Special thanks are due to Mária Bieliková, of the Slovak University of Technology in Bratislava, for her expert preparation and handling of the Student Research Forum, and to the SOFSEM Steering Committee headed by Július Štuller, of the Institute of Computer Science in Prague, for its excellent support throughout the preparation of the conference.

We are also indebted to the Organizing Committee led by Vanda Hambálková and Dana Pardubská.

November 2010

Ivana Černá
Tibor Gyimothy
Juraj Hromkovič
Keith Jeffery
Rastislav Královič
Marko Vukolic
Stefan Wolf

Conference Organization

Program Chairs

Ivana Černá
Tibor Gyimothy
Juraj Hromkovič
Keith Jeffery
Rastislav Královič
Marko Vukolic
Stefan Wolf

Program Committee

Paolo Atzeni
Gildas Avoine
Juan Bicarregui
Michael Brodie
Manfred Broy
Jan Camenisch
Andrea De Lucia
Yvo Desmedt
Massimiliano Di Penta
Matthias Fitzi
Peter Fritzson
Vashti Galpin
Ganesh Gopalakrishnan
Helena Handschuh
Mark Harman
Keijo Heljanko
Mira Kajko-Mattson
Ruediger Kapitza
Engin Kirda
Ralf Laemmel
Corrado Leita
Martin Leucker
Markus Lumpe
Gerald Lüttgen
Radu Marinescu
Eric Mercer
Michele Missikoff
Kirill Morozov
Pavol Navrat
Phong Nguyen
JesperBuus Nielsen
Krzysztof Pietrzak
Dimitris Plexousakis
Jaroslav Pokorny
AhmadReza Sadeghi
David Safranek
Wilhelm Schaefer
Lutz Schubert
Marco Serafini
Arno Siebes
Jiri Sima
Miroslava Sotakova
Jiri Srba
Julius Stuller
Tarja Systa
Patrick Valduriez
Daniel Varro
Paulo Verissimo
Tomas Vojnar
Andreas Winter
Verena Wolf
Juerg Wullschleger
Arie van Deursen
Jaco van de Pol

External Reviewers

Aldeida Aleti
Ernst Althaus
Andreas Artmeier
Gabriele Bavota
Nikola Benes
Marc Benkert
Philip Bille
Markus Bläser
Bruno Blanchet
Hans-Joachim Boeckenhauer
Hans L. Bodlaender
Benedikt Bollig
Jewgenij Botaschanjan
Ulrik Brandes
Xavier Carpent
Jakub Chaloupka
Taolue Chen
Arthur Choi
Allan Clark
Páidí Creed
Morten Dahl
Emilio Di Giacomo
Alexander Ditter
Stefan Dobrev
Wei Dong
Laurent Doyen
Jana Fabrikova
Uli Fahrenberg
Mohamed Faouzi Atig
Henning Fernau
Topher Fischer
Anna Formica
Xiang Gan
Robert Ganian
Massimiliano Goldwurm
Petr Gregor
Lars Grunske
Alexander Harhurin
Julian Haselmayr
Refael Hassin
Chong Hee Kim
Danny Hermelin
Petr Hlineny
Ákos Horváth
Leong Hou U
Yu Huang
David N. Jansen
Michael Jones
Line Juhl
Howard Karloff
Petteri Kaski
Sanjeev Khanna
Daniel Kirsten
Leonid Kof
Dennis Komm
Vaclav Koubek
Michal Koucky
Daniel Kral
Richard Kralovic
Jan Krcal
Bohuslav Krena
Jan Kretinsky
Rom Langerak
Tuomas Launiainen
Matej Lexa
Kamal Lodaya
Sylvain Lombardy
Bodo Manthey
Tania Martin
Alexander Meduna
Everett Morse
Mohammadreza Mousavi
Haiko Muller
Anderson Nascimento
Daniel Neider
Uwe Nestmann
Christian M. Neumann
Gethin Norman
Jan Obdrzalek
Mads Chr. Olesen
M. Sohel Rahman
Andras Salamon
Arnaud Sangnier
Petr Savicky
Giuseppe Scanniello
Andre Schumacher
Stefan Schwoon
Yannick Seurin

Jiří Sgall
Michael Smith
Monika Steinová
Volker Stolz
Jukka Suomela
Till Tantau
Claus Thrane
Jana Tumova
Marc Uetz
Yannis Velegrakis
Walter Vogler
Nicholas Vrvilo
Saint Wesonga
David White
Siert Wieringa
Clinton Woodward
Stanislav Zak

Table of Contents

Integrity and Consistency for Untrusted Services
(Extended Abstract)

Christian Cachin

IBM Research - Zurich
CH-8803 Rüschlikon, Switzerland
cca@zurich.ibm.com

Abstract. A group of mutually trusting clients outsources an arbitrary computation service to a remote provider, which they do not fully trust and that may be subject to attacks. The clients do not communicate with each other and would like to verify the integrity of the stored data, the correctness of the remote computation process, and the consistency of the provider's responses.

We present a novel protocol that guarantees atomic operations to all clients when the provider is correct and fork-linearizable semantics when it is faulty; this means that all clients which observe each other's operations are consistent, in the sense that their own operations, plus those operations whose effects they see, have occurred atomically in same sequence. This protocol generalizes previous approaches that provided such guarantees only for outsourced storage services.

Keywords: cloud computing, fork-linearizability, data integrity, computation integrity, authenticated data structure, Byzantine emulation.

1 Introduction

Today many users outsource generic computing services to large-scale remote service providers and no longer run them locally. Commonly called the *cloud computing model*, this approach carries inherent risks concerning data security and service integrity.

Whereas data can be stored confidentially by encrypting it, ensuring the *integrity* of remote data and outsourced computations is a much harder problem. A subtle change in the remote computation, whether caused inadvertently by a bug or deliberately by a malicious adversary, may result in wrong responses to the clients. Such deviations from a correct specification can be very difficult to spot manually.

Suppose a group of clients, whose members trust each other, relies on an untrusted remote server for a collaboration task. For instance, the group stores its project data on a cloud service and accesses it for coordination and document exchange. Although the server is usually correct and responds properly, it might become corrupted some day and respond wrongly. This work aims at discovering such misbehavior, in order for the clients to take some compensation action.

When the service provides data storage (read and write operations only), some well-known methods guarantee data integrity. With only one client, a *memory checker* [1] ensures that a read operation always returns the most recently written value. If multiple clients access the remote storage, they can combine a memory checker with an external trusted infrastructure (like a directory service or a key manager in a cryptographic file system), and achieve the same guarantees for many clients.

But in the asynchronous network model without client-to-client communication considered here, nothing prevents the server from mounting a *forking attack*, whereby it simply omits the operations of one client in its responses to other clients. Mazières and Shasha [15] put forward the notion of *fork-linearizability*, which captures the optimal achievable consistency guarantee in this setting. It ensures that whenever the server's responses to a client A have ignored a write operation executed by a client B, then A can never again read a value written by B afterwards and vice versa. With this notion, the clients detect server misbehavior from a single inconsistent operation — this is much easier than comparing the effects of *all* past operations one-by-one.

This paper makes the first step toward ensuring integrity and consistency for *arbitrary computing services* running on an untrusted server. It does so by extending untrusted storage protocols providing fork-linearizability to a generic service protocol with fork-linearizable semantics. Previous work in this model only addressed integrity for a storage service, but could not check the consistency of more general computations by the server.

Similar to the case of a storage service, the server can readily mount a forking attack by splitting the group of clients into subgroups and responding consistently within each subgroup, but not making operations from one subgroup visible to others. Because the protocol presented here ensures fork-linearizability, however, such violations become easy to discover. The method therefore protects the integrity of arbitrary services in an end-to-end way, as opposed to existing techniques that aim at ensuring the integrity of a computing platform (e.g., the *trusted computing* paradigm).

Our approach requires that (at least part of) the service implementation is known to the clients, because they need to double-check crucial steps of an algorithm locally. In this sense, the notion of fork-linearizable service integrity, as considered here, means that the clients have collaboratively verified every single operation of the service. This strictly generalizes the established notion of fork-linearizable storage integrity. A related notion for databases is ensured by the Blind Stone Tablet protocol [20].

1.1 Contributions

We present the first precise model for a group of mutually trusting clients to execute an *arbitrary service* on an untrusted server S, with the following characteristics. It guarantees *atomic operations* to all clients when S is correct and *fork-linearizability* when S is faulty; this means that all clients which observe each other's operations are *consistent*, in the sense that their own operations, plus those operations whose effects they see, have occurred atomically in same sequence.

Furthermore, we generalize the concept of *authenticated data structures* [16] toward executing arbitrary services in an authenticated manner with multiple clients. We present a protocol for consistent service execution on an untrusted server, which adds $O(n)$ communication overhead for a group of n clients; it generalizes existing protocols that have addressed only the special case of storage on an untrusted server.

1.2 Related Work

Ensuring integrity and consistency for services outsourced to third parties is a very important problem, particularly regarding security in cloud computing [8].

A common approach for tolerating faults, including adversarial actions by malicious, so-called Byzantine servers, relies on replication [5]. All such methods, however, break down as soon as a majority of servers becomes faulty. We are interested in consistency for only one server, which is potentially Byzantine.

Our approach directly builds on *authenticated data structures* [16, 14, 19]; they generalize Merkle hash trees for memory checking [1] to arbitrary search structures on general data sets. Authenticated data structures consist of communication-efficient methods for authenticating database queries answered by an untrusted provider. In contrast to our setting, the two- and three-party models of authenticated data structures allow only one client as a writer to modify the content. Our model allows any client to issue arbitrary operations, including updates.

Previous work on untrusted storage has addressed the multi-writer model. Mazières and Shasha [15] introduce untrusted storage protocols and the notion of fork-linearizability (under the name of *fork consistency*), and demonstrate them with the SUNDR storage system [12]. Subsequent work of Cachin et al. [4] improves the efficiency of untrusted storage protocols. A related work demonstrates how the operations of a revision control system can be mapped to an untrusted storage primitive, such that the resulting system protects integrity and consistency for revision control [2].

FAUST [3] and Venus [18] extend the model beyond the one considered here and let the clients occasionally exchange messages among themselves. This allows FAUST and Venus to obtain stronger semantics, in the sense that they eventually reach consistency (in the sense of linearizability) or detect server misbehavior. In our model without client-to-client communication, fork-linearizability, or one of the related "forking" consistency notions [3], is the best that can be achieved [15].

Several recent cloud-security mechanisms aim at a similar level of service consistency as guaranteed by our protocol. They include the Blind Stone Tablet [20] for consistent and private database execution using untrusted servers, the SPORC framework [9] for securing group collaboration tasks executed by untrusted servers, and the Depot [13] storage system.

Orthogonal approaches impose correct behavior on a remote service indirectly, for instance through accountability in a storage service [21] or distributed systems [10]. Yet other work relies on trusted hardware modules at all parties [6, 7].

1.3 Organization

Section 2 describes the model and recalls fork-linearizability and other consistency notions. In Section 3 the notion of *authenticated service execution* is introduced, which plays the main role for formalizing arbitrary services so that their responses can be verified. Section 4 presents the fork-linearizable service execution protocol. The detailed analysis and generalizations are omitted from this extended abstract.

2 System Model

2.1 System

We consider an asynchronous distributed system consisting of n clients $C_1, \ldots, C_n$ and a server S. Every client is connected to S through an asynchronous reliable channel that delivers messages in first-in/first-out (FIFO) order. The clients and the server together are called *parties*. A *protocol* P specifies the behaviors of all parties. An *execution* of P is a sequence of alternating states and state transitions, called *events*, which occur according to the specification of the system components.

All clients follow the protocol; in particular, they do not crash. Every client has some small local trusted memory, which serves to store keys and authentication values. The server might be faulty and deviate arbitrarily from the protocol; such behavior is also called *Byzantine*. A party that does not fail in an execution is *correct*.

2.2 Functionality

We consider a *deterministic state machine*, which is modeled by a *functionality* F as follows. It maintains a *state* $s \in \mathcal{S}$, repeatedly takes some *operation* $o \in \mathcal{O}$ as input (o may contain arguments), and outputs a *response* $r \in \mathcal{R}$ and a new state s'. The initial state is denoted by s_{F0}. Formally, a step of F is written as

$$(s', r) \;\leftarrow\; F(s, o).$$

Because operations are executed one after another, this gives the *sequential specification* of F. We discuss the concurrent invocation of multiple operations later.

We extend this notation for executing multiple operations $o_1, \ldots, o_m$ in sequence, starting from an initial state s_0, and write

$$(s', r) \;=\; F(s_0, [o_1, \ldots, o_m])$$

for $(s_i, r_i) = F(s_{i-1}, o_i)$ with $i = 1, \ldots, m$ and $(s', r) = (s_m, r_m)$.

We define the *space complexity* of F, denoted by *SPACE*$_F$, to be the number of bits required to store the largest of its states, i.e.,

$$\mathit{SPACE}_F \;=\; \max_{s \in \mathcal{S}} |s|.$$

The space complexity determines the amount of local storage necessary to execute F.

2.3 Operations and Histories

Our goal is to emulate F to the clients with the help of server S. The clients invoke the operations of F; every operation is represented by two events occurring at the client, an *invocation* and a *response*. A *history* of an execution σ consists of the sequence of invocations and responses of F occurring in σ. An operation is *complete* in a history if it has a matching response. For a sequence of events σ, *complete*(σ) is the maximal subsequence of σ consisting only of complete operations.

An operation o *precedes* another operation o' in a sequence of events σ, denoted $o <_\sigma o'$, whenever o completes before o' is invoked in σ. A sequence of events π *preserves the real-time order* of a history σ if for every two operations o and o' in π, if $o <_\sigma o'$ then $o <_\pi o'$. Two operations are *concurrent* if neither one of them precedes the other. A sequence of events is *sequential* if it does not contain concurrent operations. For a sequence of events σ, the subsequence of σ consisting only of events occurring at client C_i is denoted by $\sigma|_{C_i}$ (we use the symbol $|$ as a projection operator). For some operation o, the prefix of σ that ends with the last event of o is denoted by $\sigma|^o$.

An execution is *well-formed* if the sequence of events at each client consists of alternating invocations and matching responses, starting with an invocation. An execution is *fair*, informally, if it does not halt prematurely when there are still steps to be taken or messages to be delivered.

2.4 Consistency Conditions

We now describe the formal consistency notions required from an untrusted service, formulated in terms of the possible views of a client. A sequence of events π is called a *view* of a history σ at a client C_i w.r.t. a functionality F if σ can be extended (by appending zero or more responses) to a history σ' such that:

1. π is a sequential permutation of some subsequence of *complete*(σ');
2. $\pi|_{C_i} =$ *complete*$(\sigma')|_{C_i}$; and
3. π satisfies the sequential specification of F.

Intuitively, a view π of σ at C_i contains at least all those operations that either occur at C_i or are apparent from to C_i from its interaction with F.

One of the most important consistency conditions for concurrent operations is linearizability, which guarantees that all operations occur atomically.

Definition 1 (Linearizability [11]). *A history σ is* linearizable *w.r.t. a functionality F if there exists a sequence of events π such that:*

1. *π is a view of σ at all clients w.r.t. F; and*
2. *π preserves the real-time order of σ.*

The notion of fork-linearizability [15] (originally called *fork consistency*) requires that when an operation is observed by multiple clients, the history of events occurring before the operation is the same. For instance, when a client reads a value written by another client from a storage service, the reader is assured to be consistent with the writer up to the write operation.

Definition 2 (Fork-linearizability). *A history σ is* fork-linearizable *w.r.t. a functionality F if for each client C_i there exists a sequence of events π_i such that:*

1. *π_i is a view of σ at C_i w.r.t. F;*
2. *π_i preserves the real-time order of σ;*
3. *(*No-join*) For every client C_j and every operation $o \in \pi_i \cap \pi_j$, it holds that $\pi_i|^o = \pi_j|^o$.*

We now recall the concept of a *fork-linearizable Byzantine emulation* [4]. It summarizes the requirements put on our service emulation protocol, which runs between the clients and an untrusted server. This notion means that when the server is correct, the service should guarantee the standard notion of linearizability; otherwise, it should ensure fork-linearizability.

Definition 3 (Fork-linearizable Byzantine emulation). *A protocol P* emulates *a functionality F* on a Byzantine server S with fork-linearizability *whenever the following conditions hold:*

1. *If S is correct, the history of every fair and well-formed execution of P is linearizable w.r.t. F; and*
2. *The history of every fair and well-formed execution of P is fork-linearizable w.r.t. F.*

2.5 Cryptographic Primitives

Our implementation uses *hash functions*, *digital signatures*, and *symmetric-key encryption*. We model them as ideal functionalities here. But all notions can be made formal in the model of modern cryptography.

A hash function H maps a bit string x of arbitrary length to a short, unique representation of fixed length. It is assumed to be collision-free, that is, no party can produce two different inputs x and x' such that $H(x) = H(x')$.

A digital signature scheme provides two operations, *sign* and *verify*. The invocation of *sign* takes an index $i \in \{1, \ldots, n\}$ and a bit string m as parameters and returns a signature ϕ with the response. The *verify* operation takes the index i of a client, a string m, and a putative signature ϕ as parameters and returns a Boolean value $b \in \{\text{FALSE}, \text{TRUE}\}$ with the response. It satisfies that $\mathit{verify}(i, m, \phi) = \text{TRUE}$ for all i and m if and only if C_i has executed $\mathit{sign}(i, m) = \phi$ before. Only C_i may invoke $\mathit{sign}(i, \cdot)$ and S cannot invoke *sign*. Every party may invoke *verify*.

A symmetric encryption scheme consists of a key generation algorithm, an encryption algorithm *encrypt* and a decryption algorithm *decrypt*. Initially a trusted entity runs the key generator and obtains a key $k \in \mathcal{K}$. Algorithm *encrypt* takes k and a message m as inputs and returns a ciphertext c. Algorithm *decrypt* takes k and a ciphertext c as inputs and returns a message m. For any k and m, it is required that $\mathit{decrypt}(k, \mathit{encrypt}(k, m)) = m$. Furthermore, any party that obtains $c = \mathit{encrypt}(k, m)$ but has no access to k obtains no useful information about m.

3 Service Execution and Authentication

This section first introduces a model for executing the service F on server S such that operations are invoked by the clients. The primary task of S is to maintain the global state s of F; we intend this model for coordination services, shared collaboration spaces, lightweight databases, storage applications and so on, with small computational expense for every operation, but high demand on maintaining a consistent state.

Given this setting, the clients could simply send their operations to S and, since F is deterministic, S could execute them and return the responses. But we are interested in a model where the clients execute the bulk of every operation, so as to reduce the load on S. This assumption also helps preparing the ground for authenticating the responses of S.

In the second part of this section, we introduce a model for *authenticating* the execution of a sequence of operations issued by a single client (imagine for a moment there is only one client; we extend this to multiple clients later). The client uses its local trusted memory to maintain some authentication data, from which it verifies the responses of F sent by S. This model closely resembles the established concept of authenticated data structures.

3.1 Separated Execution

We model the execution of operations of F in a *separated way*, such that the clients do most of the work. Not all functionalities encountered require that every operation accesses the complete state s. An operation o can be executed in a separated way when it uses only a part s_o of the *global state* s of the functionality; this part may depend on the operation. If o modifies the global state, then the separated execution will also generate an updated state s'_o, which must be reconciled with s to maintain the correct semantics of F.

More formally, we say a functionality F allows *separated execution* when there exist three deterministic algorithms $\mathit{extract}_F$, exec_F, and $\mathit{reconcile}_F$ as follows. Algorithm $\mathit{extract}_F$ produces a *partial state* s_o from a global state s and an operation o,

$$s_o \;\leftarrow\; \mathit{extract}_F(s, o);$$

algorithm exec_F executes o on the partial state s_o to produce a response r and a *partial updated state* s'_o,

$$(s'_o, r) \;\leftarrow\; \mathit{exec}_F(s_o, o);$$

finally, algorithm $\mathit{reconcile}_F$ takes s'_o and o, together with the old global state s and outputs the new global state

$$s \;\leftarrow\; \mathit{reconcile}_F(s, s'_o, o).$$

The algorithms satisfy that for any $s \in \mathcal{S}$ and $o \in \mathcal{O}$, and for any s', r with $(s', r) = F(s, o)$, there exists a partial state $s_o = \mathit{extract}_F(s, o)$ and a partial updated state s'_o such that

$$(s'_o, r) = \mathit{exec}_F(s_o, o) \;\wedge\; s' = \mathit{reconcile}_F(s, s'_o, o)$$

and

$$|s_o| \ll |s| \;\wedge\; |s'_o| \ll |s'|.$$

In other words, the algorithms for the separated execution of F produce the same response and new state as the original F, but there exist intermediate states for the operation (s_o and s'_o), which are much smaller than the full state(s). The latter requirement should be understood qualitatively and is not quantified; but it is crucial for enabling efficient separated execution between a client and a server.

The *communication complexity* of some F with separated execution measures the size of the messages that must be communicated for separated execution. It is denoted by COMM_F and defined as the number of bits required to store the largest partial state s_o, partial updated state s'_o, together with a description of the operation o itself, for executing any operation on any state. That is,

$$\begin{aligned}\mathit{COMM}_F \;=\; \max\bigl\{ |s_o| + |s'_o| + |o| \;\big|\; & \\ s \in \mathcal{S}, o \in \mathcal{O}, s_o = \mathit{extract}_F(s,o), (s'_o, r) = \mathit{exec}_F(s_o, o)\bigr\}.\end{aligned}$$

3.2 Authenticated Separated Execution

When only a single client engages in separated execution of operations on the server, well-known methods allow the client to verify the correctness of the responses. These methods protect the client from a faulty server that tries to forge wrong responses. Known generally as *authenticated data structures* [16, 14], they apply to a broad class of information retrieval services, such as reading an item from a memory, hash tables, or search queries to a structured data type. Such service authentication schemes rely on a small *authenticator* value maintained by the client in its local trusted memory. The client can verify the response of an operation o in such a way that it recognizes when the response differs from the correct response r, resulting from applying o to the current state s of the service. That is, state s is obtained by applying all past operations of the client to F in order and the correct response is determined by $(s', r) = F(s, o)$. We model this concept as an extension of separated execution.

We say a functionality F allows *authenticated separated execution* when there exist three deterministic algorithms $\mathit{authextract}_F$, $\mathit{authexec}_F$, and $\mathit{authreconcile}_F$ as follows. Algorithm $\mathit{authextract}_F$ produces a *partial state* s_o from a global state s and an operation o,

$$s_o \;\leftarrow\; \mathit{authextract}_F(s, o).$$

The client maintains an *authenticator* denoted by a, which is initialized to a default value a_{F0}. Algorithm $\mathit{authexec}_F$ takes a, s_o, and o as inputs and produces an *updated authenticator* a', a *partial updated state* s'_o, and a response r. In the course of executing o, the algorithm also *verifies* its inputs with respect to a and may output the special symbol $\perp$ as response, indicating that the verification failed. In other words,

$$(a', s'_o, r) \;\leftarrow\; \mathit{authexec}_F(a, s_o, o),$$

with $r = \perp$ if and only if verification failed. Finally, algorithm $\mathit{authreconcile}_F$ takes s'_o and o, together with the old global state s and outputs the new global state

$$s \;\leftarrow\; \mathit{authreconcile}_F(s, s'_o, o).$$

Its role is exactly the same as in separated execution.

A *proper authenticated execution* of the operation sequence $o_1, \ldots, o_m$ proceeds as follows. Starting with the initial authenticator $a_0 = a_{F0}$ and state $s_0 = s_{F0}$, it computes

$$\begin{aligned}(s_i, r_i) &\leftarrow F(s_{i-1}, o_i)\\ s_{o_i} &\leftarrow \mathit{authextract}_F(s_i, o)\\ (a_i, s'_{o_i}, r_i) &\leftarrow \mathit{authexec}_F(a_{i-1}, s_{o_i}, o_i),\end{aligned}$$

for $i = 1, \dots, m$ and outputs the triple (a_m, s_m, r_m) containing an authenticator a_m, state s_m, and response r_m.

Consider now the proper authenticated execution of an arbitrary operation sequence and the resulting authenticator a and state s. The following conditions must hold:

Correctness: For any $o \in \mathcal{O}$ and $(s', r) = F(s, o)$, there exist $s_o = \mathit{authextract}_F(s, o)$ and a', s'_o, and $r \neq \perp$ such that

$$(a', s'_o, r) = \mathit{authexec}_F(a, s_o, o) \;\wedge\; s' = \mathit{authreconcile}_F(s, s'_o, o).$$

and

$$|a'| \ll |s| \;\wedge\; |s_o| \ll |s| \;\wedge\; |s'_o| \ll |s'|.$$

Security: For any $o \in \mathcal{O}$ and any adversary that outputs some $\tilde{s}_o$, suppose that there exist a' and s'_o such that $(a', s'_o, \tilde{r}) = \mathit{authexec}_F(a, \tilde{s}_o, o)$ with $\tilde{r} \neq \perp$; then $\tilde{r} = r$.

The *correctness* property is simply reformulated from the unauthenticated scheme for separated execution. It states that for any authenticator and state s resulting from a proper authenticated execution, applying separated execution of o yields a response $r \neq \perp$ such that verification succeeds and, moreover, the resulting updated state s' together with r satisfies $(s', r) = F(s, o)$.

The *security* property considers a faulty S as an adversary, which tries to forge some partial state $\tilde{s}_o$ that causes the client to produce a wrong response $\tilde{r}$. But in an authenticated separated execution scheme, algorithm $\mathit{authexec}_F$ either outputs the correct response ($\tilde{r} = r$), or it recognizes the forgery and the verification fails ($\tilde{r} = \perp$).

The *communication complexity* of some F with authenticated separated execution is defined in the same way as for separated execution and measures how much data must be communicated between C and S.

The notion of authenticated data structures [14] differs from a service with authenticated separated execution in that the former does not contain a partial updated state and the reconciliation step. In fact, the server could equally well execute the whole operation on the state that it maintains. But in practice, many algorithms execute update operations more efficiently when the client computes the updated parts of the state and the server merely stores them in its memory.

3.3 Examples

The literature contains many examples of data structures that can be formulated as functionalities with authenticated separated execution. They are interesting because their communication complexity for separated execution is much smaller their space complexity. For instance, hash trees can be used to check the correctness of individual entries in a memory with N elements [1] with complexity $O(\log N)$, a generalization of hash trees can authenticate responses produced by any DAG-structured query evaluation algorithm with logarithmic overhead [14], and cryptographic methods based on accumulators can maintain authenticated hash tables with constant communication for query operations and sub-linear cost for updates [17].

As a concrete example, consider a functionality *MEM* whose state consists of N storage locations denoted by $\mathit{MEM}[1], \dots, \mathit{MEM}[N]$. *MEM* supports two operations:

read(j), which returns *MEM*$[j]$, and *write*$((j,x))$, which assigns *MEM*$[j] \leftarrow x$ and returns nothing. Note that for $N = n$ and when C_i may only write to *MEM*$[i]$, we obtain the functionality that was considered in most previous work on untrusted storage (e.g., [4]).

A standard hash tree computed over *MEM*$[1], \ldots,$ *MEM*$[N]$ gives an authenticated separated execution scheme, where the internal nodes of the tree are also stored in the state of *MEM*. The authenticator is the root node of the hash tree, which commits all entries in *MEM*. Algorithm *authextract*$_{MEM}$ for an operation that concerns entry j always returns the internal tree nodes along the path from the root to the leaf node j and all their siblings, which are needed for recomputing the root hash in order to authenticate leaf node j [1]. Verification succeeds if the recomputed root hash matches the authenticator. For a write operation, the nodes on the path from *MEM*$[j]$ to the root are updated and included in the partial updated state s'_o. The server extracts them from s'_o and stores them in the appropriate place during *authreconcile*$_{MEM}$.

The client must explicitly recompute the path in the hash tree also for write operations, in order to verify the sibling nodes along the path from the modified leaf node to the root; these nodes originate from the server and influence the computation of the new root hash. If they are not verified, they might lead to an invalid authenticator. Because the client computes these values anyway, they are contained in the partial updated state, and the server only needs to store them.

In this way, our notion of authenticated separated execution models closely what happens in practical hash tree implementations inside cryptographic storage systems; this is not possible with the notion of an authenticated data structure, where no reconciliation algorithm is foreseen.

4 Fork-Linearizable Execution Protocol

We now introduce a novel untrusted service execution protocol, which emulates an arbitrary F on a Byzantine server with fork-linearizability. The protocol combines elements from existing untrusted storage protocols with an authenticated separated execution scheme for F.

The protocol operates in lock-step mode, similar to the bare-bones storage protocol of SUNDR [15]. This means that the server serializes all operations and does not allow them to execute concurrently. Proceeding in lock-step is for illustration purposes only; extending it to concurrent operations is feasible and discussed at the end of this paper.

At a high level, the protocol operates like this. A client assigns a local *timestamp* to every one of its operations. Every client maintains a timestamp vector T in its trusted memory. At client C_i, entry $T[j]$ is equal to the timestamp of the most recently executed operation by C_j in some view of C_i. To begin executing an operation o, client C_i sends a SUBMIT message with o to S. A correct S responds to this SUBMIT message by invoking the authenticated separated execution scheme, and computes $s_o \leftarrow$ *authextract*$_F(s, o)$ on the current state s.

In addition to s, the server maintains a timestamp vector V, an authenticator a, and a signature φ, which it received in a so-called COMMIT message from the client C_c that executed the last preceding operation at S. The signature was issued by C_c on V and a. The server sends a REPLY message to C_i containing V, a, s_o, c, and φ.

When it receives the REPLY message, the client first checks the content. It verifies the signature φ and makes sure that $V \geq T$ (using vector comparison) and that $V[i] = T[i]$. If not, the client aborts the operation and halts, because this means that S has violated the consistency of the service.

Then C_i verifies the response with respect to a and runs the separated execution by computing $(a', s'_o, r) \leftarrow \mathit{authexec}_F(a, s_o, o)$. If the verification fails, the client again halts. Otherwise, C_i proceeds to copying the received timestamp vector V into its variable T, incrementing $T[i]$, and computing a signature φ' on T and a'. The value $T[i]$ becomes the *timestamp* of o. Finally, C_i returns a COMMIT message to S containing T, a', s'_o, and φ'.

It is not hard to see that all checks are satisfied when S is correct because every client only increments its own entry in a timestamp vector. Therefore, the timestamp vectors sent out by S in REPLY messages appear in strictly increasing order.

The description so far allows the server to learn the authenticator values, which is not foreseen in the model of an authenticated separated execution scheme. To prevent any damage that might be caused by this, all clients know a common secret key k for a symmetric encryption scheme and use it to encrypt the authenticator before sending it to S.

This completes the high-level description of the untrusted service execution protocol; the details are given in Algorithms 1 and 2.

Algorithm 1. Untrusted execution protocol for client C_i

```
State
    $k \in \mathcal{K}$                                         // symmetric encryption key
    $T \in \mathbb{N}_0{}^n$, initially $[0]^n$                        // current timestamp vector

upon operation $\mathit{run}_F(o)$ do
    send message $[\text{SUBMIT}, o]$ to $S$
    wait for message $[\text{REPLY}, V, \bar{a}, s_o, c, \varphi]$
    if $\big(V = [0]^n \vee \mathit{verify}(c, \text{COMMIT}\|V\|\bar{a}, \varphi)\big) \wedge V \geq T \wedge V[i] = T[i]$ then
        if $V = [0]^n$ then
            $a \leftarrow a_{F0}$
        else
            $a \leftarrow \mathit{decrypt}(k, \bar{a})$
        $(a', s'_o, r) \leftarrow \mathit{authexec}_F(a, s_o, o)$
        if $r \neq \bot$ then
            $T \leftarrow V$
            $T[i] \leftarrow T[i] + 1$
            $\varphi' \leftarrow \mathit{sign}(i, \text{COMMIT}\|T\|a')$
            $\bar{a}' \leftarrow \mathit{encrypt}(k, a')$
            send message $[\text{COMMIT}, o, T, \bar{a}', s'_o, \varphi']$ to $S$
            return $r$
    halt
```

Algorithm 2. Untrusted execution protocol for server S

```
State
    $s \in \mathcal{S}$, initially $s_{F0}$                                        // state of $F$
    $c \in \{1, \ldots, n\}$, initially 1              // index of currently or most recently served client
    $V \in \mathbb{N}_0{}^n$, initially $[0]^n$                  // timestamp vector of last committed operation
    $\bar{a}$, initially $\epsilon$                         // encrypted authenticator of last committed operation
    $\varphi$, initially $\epsilon$                                      // signature of last committed operation
    $block \in \{\text{FALSE}, \text{TRUE}\}$, initially FALSE

upon receiving message $[\text{SUBMIT}, o]$ from $C_i$ such that $block = \text{FALSE}$ do
    $s_o \leftarrow authextract_F(s, o)$
    send message $[\text{REPLY}, V, \bar{a}, s_o, c, \varphi]$ to $C_i$
    $c \leftarrow i$
    $block \leftarrow \text{TRUE}$

upon receiving message $[\text{COMMIT}, o, T, \bar{a}', s'_o, \varphi']$ from $C_i$ such that $block = \text{TRUE} \wedge i = c$ do
    $s \leftarrow authreconcile_F(s, s'_o, o)$
    $(V, \bar{a}, \varphi) \leftarrow (T, \bar{a}', \varphi')$
    $block \leftarrow \text{FALSE}$
```

Intuitively, the algorithm relies on the same properties of vector clocks as previous protocols for untrusted storage [15, 4]. Note that S can only send a timestamp vector and authenticator in a REPLY message that have been signed by a client; otherwise, the first verification step in Algorithm 1 fails. Under this condition, S may violate the protocol only by sending a timestamp vector/authenticator pair that is properly signed but does not satisfy a global sequential order of the operations.

In other words, a violation by S means that there is one operation o_0 whose timestamp vector is received in a REPLY by at least two different clients C_1 and C_2, in operations o_1 and o_2, respectively. If all other information is correct, operations o_1 and o_2 both succeed, but the two clients sign *incomparable* timestamp vectors. According to the protocol, one can then show that C_1 will not execute any operation in a view at C_1 that includes o_2 and, vice versa, any operation in a view at C_2 that includes o_1 will cause C_2 to abort.

With the functionality *MEM* from the previous section and n storage locations, this protocol gives the same guarantees as the bare-bones storage protocol of SUNDR [15] and the lock-step protocol of Cachin et al. [4]. As in the latter protocol, our algorithm adds a linear (in n) overhead to the communication complexity of separated execution.

5 Conclusion

This paper has introduced the first precise model for a group of mutually trusting clients to execute an arbitrary service on an untrusted server S, such that the clients observe atomic operations when S is correct and the service respects fork-linearizability when S is Byzantine. An implementation of this notion has been obtained by combining any scheme for authenticated separated execution with elements from untrusted storage protocols.

The protocol is not particularly efficient because a correct server executes all operations in lock-step mode. Similar to untrusted storage protocols, the protocol can be improved by letting the clients execute some operations concurrently, as long as they do not conflict. Some restrictions on the achievable parallelism have been identified [4]. Clarifying the concept of conflicts for arbitrary functionalities and extending the protocol to concurrent operations are deferred to the forthcoming full version of this paper.

Acknowledgments

I thank Alexander Shraer, Idit Keidar, Rüdiger Kapitza, and Matthias Schunter for interesting discussions about the ideas leading to this paper.

This work has been supported in part by the European Commission through the ICT programme under contracts ICT-2007-216676 ECRYPT II and ICT-2009-257243 TCLOUDS.

References

[1] Blum, M., Evans, W., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories. Algorithmica 12, 225–244 (1994)

[2] Cachin, C., Geisler, M.: Integrity protection for revision control. In: Abdalla, M., Pointcheval, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 382–399. Springer, Heidelberg (2009)

[3] Cachin, C., Keidar, I., Shraer, A.: Fail-aware untrusted storage. In: Proc. International Conference on Dependable Systems and Networks (DSN-DCCS), pp. 494–503 (2009)

[4] Cachin, C., Shelat, A., Shraer, A.: Efficient fork-linearizable access to untrusted shared memory. In: Proc. 26th ACM Symposium on Principles of Distributed Computing (PODC), pp. 129–138 (2007)

[5] Charron-Bost, B., Pedone, F., Schiper, A. (eds.): Replication: Theory and Practice. LNCS, vol. 5959. Springer, Heidelberg (2010)

[6] Chun, B.G., Maniatis, P., Shenker, S., Kubiatowicz, J.: Attested append-only memory: Making adversaries stick to their word. In: Proc. 21st ACM Symposium on Operating System Principles (SOSP), pp. 189–204 (2007)

[7] Chun, B.G., Maniatis, P., Shenker, S., Kubiatowicz, J.: Tiered fault tolerance for long-term integrity. In: Proc. 7th USENIX Conference on File and Storage Technologies, FAST (2009)

[8] Cloud Security Alliance, CSA (2010), http://www.cloudsecurityalliance.org/

[9] Feldman, A.J., Zeller, W.P., Freedman, M.J., Felten, E.W.: SPORC: Group collaboration using untrusted cloud resources. In: Proc. 9th Symp. Operating Systems Design and Implementation, OSDI (2010)

[10] Haeberlen, A., Kouznetsov, P., Druschel, P.: PeerReview: Practical accountability for distributed systems. In: Proc. 21st ACM Symposium on Operating System Principles (SOSP), pp. 175–188 (2007)

[11] Herlihy, M.P., Wing, J.M.: Linearizability: A correctness condition for concurrent objects. ACM Transactions on Programming Languages and Systems 12(3), 463–492 (1990)

[12] Li, J., Krohn, M., Mazires, D., Shasha, D.: Secure untrusted data repository (SUNDR). In: Proc. 6th Symp. Operating Systems Design and Implementation (OSDI), pp. 121–136 (2004)

[13] Mahajan, P., Setty, S., Lee, S., Clement, A., Alvisi, L., Dahlin, M., Walfish, M.: Depot: Cloud storage with minimal trust. In: Proc. 9th Symp. Operating Systems Design and Implementation, OSDI (2010)

[14] Martel, C., Nuckolls, G., Devanbu, P., Gertz, M., Kwong, A., Stubblebine, S.G.: A general model for authenticated data structures. Algorithmica 39, 21–41 (2004)

[15] Mazières, D., Shasha, D.: Building secure file systems out of Byzantine storage. In: Proc. 21st ACM Symposium on Principles of Distributed Computing, PODC (2002)

[16] Naor, M., Nissim, K.: Certificate revocation and certificate update. IEEE Journal on Selected Areas in Communications 18(4), 561–570 (2000)

[17] Papamanthou, C., Tamassia, R., Triandopoulos, N.: Authenticated hash tables. In: Proc. 15th ACM Conference on Computer and Communications Security, CCS (2008)

[18] Shraer, A., Cachin, C., Cidon, A., Keidar, I., Michalevsky, Y., Shaket, D.: Venus: Verification for untrusted cloud storage. In: Proc. Cloud Computing Security Workshop (CCSW). ACM, New York (2010)

[19] Tamassia, R., Triandopoulos, N.: Computational bounds on hierarchical data processing with applications to information security. In: Caires, L., et al. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 153–165. Springer, Heidelberg (2005)

[20] Williams, P., Sion, R., Shasha, D.: The blind stone tablet: Outsourcing durability to untrusted parties. In: Proc. Network and Distributed Systems Security Symposium, NDSS (2009)

[21] Yumerefendi, A.R., Chase, J.S.: Strong accountability for network storage. ACM Transactions on Storage 3(3) (2007)

A Structured Codesign Approach to Many-Core Architectures for Embedded Systems

Jürg Gutknecht

ETH Zürich
gutknecht@inf.ethz.ch

Abstract. In this paper we present a structured hardware/ software codesign approach to modeling and development of embedded systems for many-core platforms. Our method is based on the combination of three technologies that are not commonly used in such a context: a.) programmable hardware, b.) high-level computing model and programming language, and c.) hybrid hardware/ software compilation. Resulting benefits are better reliability and performance, as well as higher productivity, portability and scalability thanks to a fully integrated design flow. We shall also show that our method is promising in terms of resource-efficiency and in particular energy-efficiency. An application from the medical ICT area will serve as a proof of concept.

1 Introduction

The context of the following presentation is the goal of advancing the level of embedded systems development. This is a highly relevant goal as a.) more than 95% of all processors ever produced go into embedded systems and many of them go into safety-critical systems, and b.) the technological step from single-core architectures to many-core architectures does not stop in front of embedded systems. Embedded systems will be many-core systems in the future, and the underlying architecture will be custom design rather than some off-the-shelf microprocessor plus standard memory plus device controllers.

Many-core technology will open up an entirely new range of opportunities in terms of computing model, hardware/ software partnership, operating system etc. but it will also challenge the standard development procedures and design tools commonly used today, notably the following peculiarities

- Total separation of hardware architecture and software architecture in the design process
- Use of a low-level programming language, often in combination with hand-crafting
- Use of some general-purpose operating system, downscaled to microprocessors

The approach presented in this paper is based on the vision of a programmable hardware platform that is capable of accommodating a large number of microprocessors (cores) of potentially different kinds, each of them operating on some local memory, and a corresponding number of interprocessor connections for communication purposes. However, neither any specific microprocessor architecture, nor any kind of a full interconnect, nor homogeneity, nor shared global memory is assumed.

In our computing model, programs resemble downscaled distributed systems with *actor objects* interoperating via message passing. The important point here is that each program is a coherent description of both the topology of the actor network and the algorithm to be executed by each actor object. Program development is supported by a hybrid compiler that maps the topology into a description in some hardware language (HDL), and the algorithms into microprocessor code respectively.

In view of the potentially large number of available processor cores, our programming model aims at minimizing or even eliminating processor sharing, preemption and asynchronous interrupts, to the benefit of a deterministic behavior and of better control of real-time guarantees.

The work presented here builds on the work described in [1]. Similar projects sharing our goal of an integrated and automated design flow for embedded systems and our emphasis on data flow architectures are *OpenDF* [2], *Handle-C* [3], *StreamIT* [4], and *Streams-C* [5]. However, our approach clearly distinguishes itself against these projects by its focus on a high-level, general purpose programming language that strictly hides all details about the underlying hardware from the programmers.

2 Computing Model

Our computing model is a collection of communicating actor objects (or *actors*, in short). Each actor consists of a local state space, an *activity* describing its intrinsic behavior, one or more communication activities, and a constructor for the purpose of initializing the state space and constructing the communication links. All activities within an actor are supposed to run in parallel. A *program module* is a collection of logically connected actors plus a constructor for the construction of the actor network, which is supposed to be static. The basic idea is now to compile both the construction of the actor network and the construction of the communication links into HDL, and to compile the “kernel” codes in actors into microprocessor code.

Assume, for example, that we want to construct a network consisting of two instances of some actor type X, one instance of another actor type Y, and two communication links as depicted in Figure 1.

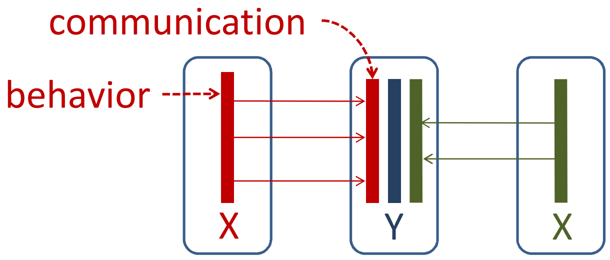


Fig. 1. Sample actor network

In a notation close to the Pascal/ Modula/ Oberon language family, the corresponding program text looks as follows. Notice that the colored/ uncolored part of the code will be compiled into hardware language/ microprocessor code respectively.

```
    module M;
      var x1, x2: X; y: Y;

        type
          X = object
            var c: Y.C;
            activity A;
              var i, j, k: integer;
            begin (*behave*)
              …; c(i, j); …; c(k); …
            end A;
            procedure X (y: Y);
            begin (*construct objec*)
              …; new (c); …
            end X;
        begin new A (*launch behavior*)
        end X;

        Y = object
          activity A;
            begin (*behave*) …
          end A;
          activity C;
            var u, v, w: integer;
          begin (*communicate*)
            …; accept(u, v); …; accept(w); …
          end C;
          procedure Y;
            begin (*construct object*)…
          end Y;
        begin new A
        end Y;

  begin new(y); new(x1, y); new (x2, y)
  end M.
```

Activities running within an actor need to synchronize their actions properly, if they operate on the shared state space. However, as all but one activity are dedicated to communication over a separate channel, synchronization can easily be achieved in a lock-free form, for example by atomically switching between an active buffer and a shadow buffer. Figure 2 illustrates the situation for actors of type Y. The figure shows one microprocessor P and two communication processors C reading from input buffers and accessing the shared state space M.

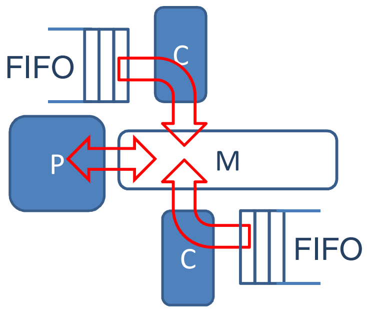


Fig. 2. Microprocessor and communication processors operating on shared local state-space within an actor

With the aim of exploiting all of the potential for parallelization, we allow activities to spawn mutually independent *sub-activities*, where the spawning "parent" activity acts as a *fence* for its sub-activities: it is considered terminated after all sub-activities have terminated.

The following figure specifies an activity *A* spawning three sub-activities, two of type *A1* and one of type *A2*:

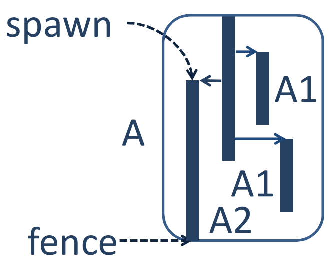


Fig. 3. Activity spawning sub-activities

Here is the corresponding program code:

```
activity A1;
begin …
end A1;

activity A2;
begin …
end A2;

activity A;
begin
  … new(A1); … new(A2); … new(A1); … (*spawn*)
end A;
```

In summary, our computing model supports concurrency on two levels:

- Level of actor objects (intrinsic behavior, communications)
    - Number statically determined at compile time
    - Synchronization needed
    - Scheduling not needed
- Level of activities (mutually independent sub-tasks)
    - Number dynamically determined at runtime
    - Synchronization not needed
    - Scheduling needed

If a large number of processor cores are available, it is justified to assign a separate core to each behavior/ communication activity, and a pool of cores to sub-tasks.

3 Target Platform

The target platform of our development environment for embedded systems is *programmable hardware* or, more precisely, FPGA (Field Programmable Gate Array). We are currently using Xilinx Virtex-5 XC5VLX50T for this project but our integrated development environment is inherently portable.

The basic FPGA resources are lookup tables (LUT), block RAMs (BRAM), and digital signal processor slices (DSP). However, unlike approaches to "hardware compilation" on a statement-by-statement basis, our system does not rely on such basic resources directly but on a library of prefabricated hardware components of some higher-level, synthesized functionality instead. The hardware library typically includes the following kind of components or *modules*:

- On-chip memory
- FIFO buffers
- I/O controllers (UART)
- Floating point units
- Microprocessors

The main purpose of the hardware library is facilitating the mapping task of the compiler. Each module is expressed in "behavioral Verilog", a portable hardware description code that uses no device-specific libraries (apart from some idioms involving DSP and BRAM for optimizing purposes) and that can be compiled for a specific target FPGA by conventional behavioral synthesis tools.

The microprocessor used in this project (see Figure 4) is a custom design called "tiny RISC machine" (TRM). The TRM is a 32 bit machine that comes with both a configurable on-chip instruction memory (IM) and data memory (DM). Each TRM consumes only 2% LUT of the Virtex-5 FPGA.

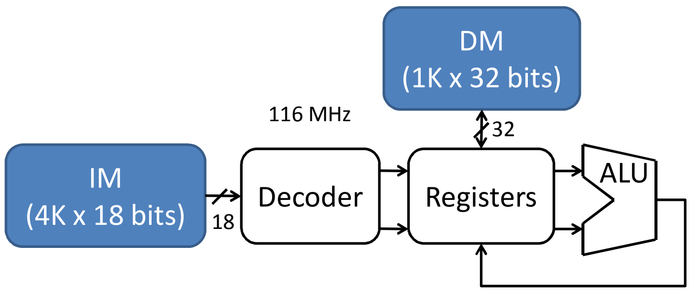


Fig. 4. Diagram of TRM

The TRM comes in different variants, including

- I/O variant TRM.IO
- Direct memory access variant TRM.DMA
- Vector variant TRM.Vector

Each of the TRM variants features a correspondingly extended instruction set. It is worth noting in this context that vector processing is an important ingredient in our project because it supports *high performance computing* and in particular *image processing* in embedded systems. Consequently, our programming language provides powerful built-in types for vectors, matrices, tensors, and generic operations on these types.

The following diagram shows the vector TRM, a basic TRM enhanced by a vector unit.

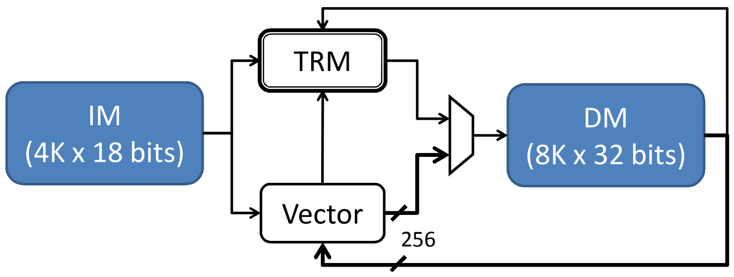


Fig. 5. Diagram of TRM.Vector

The vector unit processes floating point vectors of length 8, and it takes 12% of the Virtex-5 LUT. Built-in operations are vector addition, vector multiplication, and horizontal addition (for fast computing of scalar products). Each TRM.Vector microprocessor is typically coupled with a direct memory access TRM (TRM.DMA) for fast reading and writing of entire vectors from and to the external DDR2 memory.

FIFO buffers are used for communication between TRMs, in particular for communication between TRM.Vector and TRM.DMA but also for actor interoperability according to our programming model as explained in Section 3. FIFOs are implemented with LUTs or BRAMs, depending on their depth.

In concluding this section, let us summarize the entire automated design flow. We can distinguish the following steps, see also Figure 6:

1. Analyze the module body and actor constructors to determine the system topology (actors and communication links). Compile the activities into microprocessor code and generate the instruction image and the data image (*.mem* files) for each microprocessor core (or cluster of cores).
2. Map the results of step 1 to functional modules provided by the hardware library such as processor cores or cluster of cores, on-chip memories, FIFOs, I/O controllers etc. and generate Verilog code. In addition, generate a script file *make.tcl* and a block memory configuration file *ram.bmm*.
3. Call the Xilinx synthesizer, router and configurator to create the target machine from the generated Verilog code, to set the clock target, to patch the memory images (*.mem* files) to the BRAM defined in *ram.bmm*, and to create the final bitstream.
4. Download the bitstream to the FPGA.

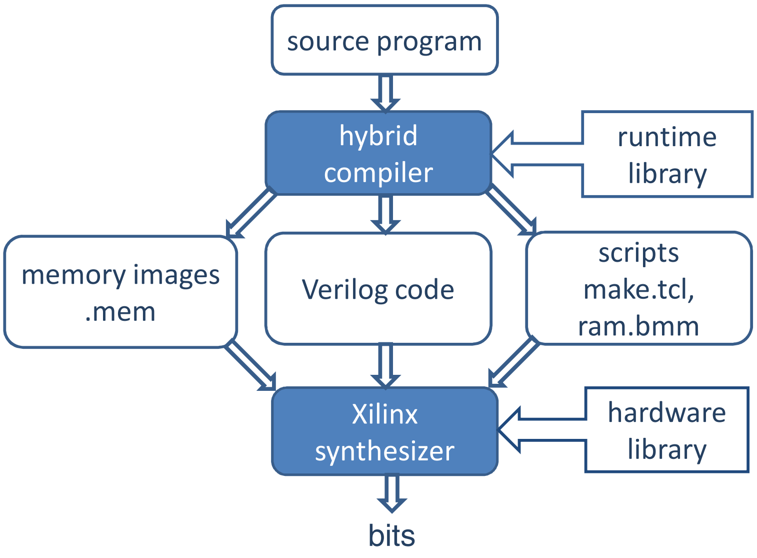


Fig. 6. System design flow

4 Proof of Concept

As a proof of concept, an electrocardiographic (ECG) signal analyzer has been implemented. Its task is to analyze the activity of the heart, the morphology of the corresponding waves, and the heart rate variability (HRV), with the aim of detecting and classifying potential anomalies. The signal to be analyzed decomposes into 8 physical channels, each of them sampled at 500 Hz.

This application belongs to the *data-streaming* class. Figure 7 depicts its topology and the corresponding functional diagram.

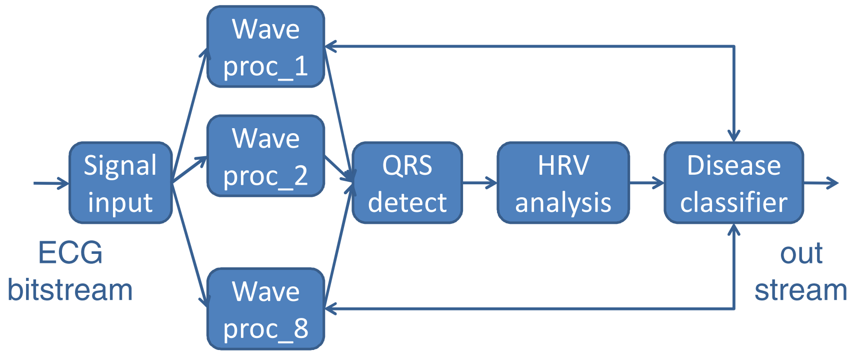


Fig. 7. Program topology of the ECG application

After each sampling, the following steps are performed:

1. Edit input: Receive ECG signal from UART, compose individual samples, and distribute them to channel processors.
2. Wave processing (per channel): Precondition wave by suppressing noise via linear filtering; Detect the heart beats and contractions.
3. QRS detection: Detect QRS patterns and make a final decision about heart rate on the basis of standard multichannel logic.
4. HRV analysis: Analyze the current heart rhythm and the heart rate variability.
5. Disease detection: Use decision tree logic to detect and classify arrhythmia events such as premature ventricular contractions (PVC), ventricular tachycardia etc. Feed results back to configure wave processing.

Notice that the topology in Figure 7 exhibits two concurrency patterns, as they typically occur in data streaming applications:

- Fan out (to separate channel processing)
- Pipeline (to analyze the signal in stages)

Figure 8 shows how the topology of the ECG analyzer is mapped to FPGA hardware.

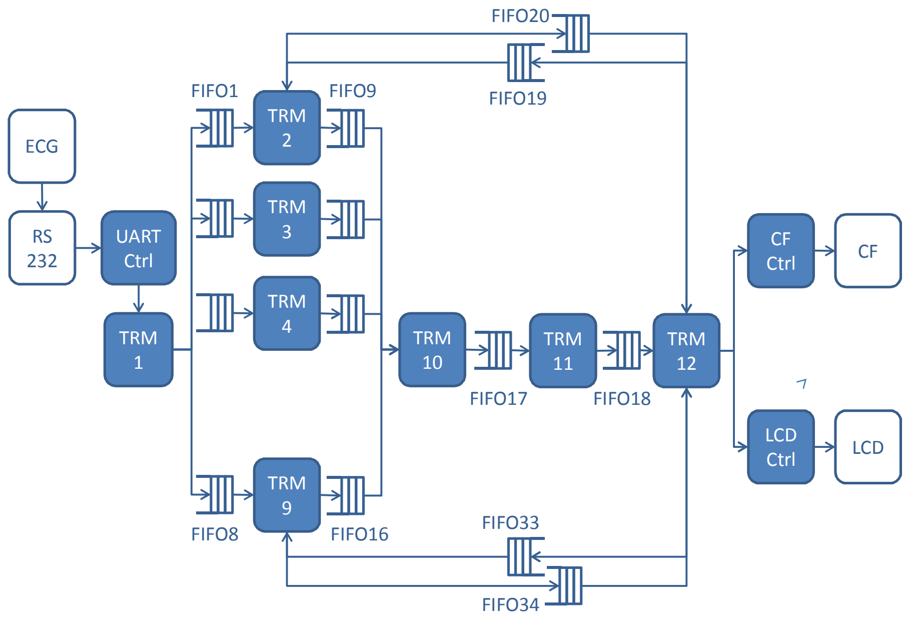


Fig. 8. FPGA implementation of the ECG topology

While the ECG analyzer served primarily the purpose of a proof of concept for our integrated and automated development process for many-core embedded systems, its functional quality is quite remarkable. For example, applied to test record I01 from the 12-lead Arrhythmia Database of the St.-Petersburg Institute of Cardiological Technics, the detection rate for the more than 2700 ECG heart beats was 100.0%, with 328 from 341 PVC contractions correctly classified.

Based on an extensive evaluation procedure of two different FPGA implementations of the ECG analyzer, the following lessons can be learned:

Feasibility of the Approach

Data-streaming applications like the ECG analyzer exhibit a high degree of intrinsic potential for parallel computing. Typical concurrency patterns are fan-outs and pipelines. The use of a combination of an actor model and a data flow model has proved to be highly suitable for this kind of applications, and the hybrid hardware/ software compiler ensures a very high degree of portability. It was merely a matter of minutes to build a new system after (experimental) changes in the underlying hardware.

Use of Resources and Energy

The combination of a tiny RISC processor (TRM), configurable local memory, and configurability of the topology of interprocessor links contributes significantly to an economical use of FPGA resources and energy. A comparison with an older and more rigid model based on constant-size local memory and on a full on-chip interconnect witnesses a slight improvement in terms of FPGA resources used and a dramatic improvement in terms of energy consumption, in particular in quiescent mode, see Table 1 below.

Table 1. Comparative use of FPGA resources and energy consumption

System	#TRM	#LUT	#BRAM	#DSP	**Quiescent power (W)**	Dynamic power (W)
Configurable	12	13859 (48%)	52 (86%)	12 (25%)	**0.49742**	0.48060
Rigid	12	15510 (53%)	56 (93%)	12 (25%)	**3.43823**	0.58988

Performance measurements showed that each of the processor cores is used to less than 10% of its capacity in average at the highest achievable clock rate of 116 MHz. Even though the signal processing algorithms currently used are not at their highest level of sophistication yet, the large amount of unused computing power suggests that, in the interest of optimized energy efficiency, the clock rate should be carefully tuned towards a minimum under the given real-time constraints.

For the purpose of exploiting the limits of the capacity of the Virtex-5 FPGA, a test application comprising 30 actors arranged in a linear sequence was implemented. Each of the actors continuously reads data from its left neighbor and passes it on to its right neighbor, where the left most actor reads its data from an 8-bit switch, and the right most actor displays its data in the form of 8 LEDs. In terms of FPGA resources, the test application consists of 30 TRM cores and of 29 32-element deep LUT RAMs implementing the buffer FIFOs for the communication network. In total, 27692 LUTs, 60 BRAMs and 30 DSPs are used. More advanced FPGAs on the market today such as the Virtex-6 XC6VSX475T can easily accommodate up to 500 TRM cores.

Potential for Improvement

In the current implementation, the full power of our language is not exploited yet. Currently, a single processor core is assigned to each actor, which makes it impossible to exploit fined-grained parallelism expressed in terms of the "fenced statement" construct within individual actors. An obvious solution would be processor clusters, sharing a local memory. Also, separate processors for the handling of communication are not supported at this time. The use of ultra-simple communication processors would be an interesting option.

Finally, mechanisms have been discussed to further improve power-awareness. We already mentioned the goal of down-scaling the clock rate appropriately. Another promising measure is shutting down processors when they are unused. Power-aware variants of the TRM are currently being explored. One idea is to add a register whose contents define the state of the processor. Its value would be set to "sleeping" by a running activity and reset to "running" automatically whenever input data has arrived in one of the communication buffers associated with this TRM.

5 Summary and Conclusion

With the aim of facilitating the construction of sophisticated embedded systems on top of programmable hardware platforms, we have developed a new programming language with built-in concurrency and a corresponding hybrid hardware/ software compiler. Programs in our language exhibit a unique combination of actors and data flow. Our hybrid compiler translates programs into FPGA hardware code but, unlike other "hardware compilers", it uses pre-fabricated hardware modules of a granularity on the level of microprocessor, local memory, FIFO etc. rather than elementary FPGA elements like LUT, BRAM, DSP etc. for this purpose. The basic microprocessor used in this project is a custom designed tiny RISC machine (TRM) that comes in different variants including energy-aware, vector processing, direct memory access etc. As a large number of TRMs fit on a single FPGA, strategies like processor sharing, preemption, asynchronous context-switches and interrupts etc. are no longer used, with the highly beneficial effect of reduced complexity and increased predictability and deterministic behavior.

Other important strengths of our approach are increased productivity and portability. The built-in support for vector processing makes our system particularly promising in connection with high performance computing applications, for example in the areas of signal processing and image processing. And, thanks to multiple configurability options, a high level of resource efficiency can be achieved.

Acknowledgement

Many people have contributed substantially to the current state of this work still in progress. We greatly acknowledge the work done by Chuck Thacker from Microsoft Research, Niklaus Wirth, Lisa (Ling) Liu, and Dan Tecu on the hardware architecture and library, the energy-awareness contributions by Shiao-Li Tsao from the National Chiao Tung University in Taiwan, the compiler and language work done by Felix Friedrich, and the application work done by Oleksi Morozov and Patrick Hunziker from the University Hospital in Basel. We particularly acknowledge the support and funding provided for this project by Microsoft Switzerland and Microsoft Research in the context of the "Innovation Cluster for Embedded Software".

References

1. Liu, L., Morozov, O.: A Process-Oriented Streaming System Design Paradigm for FPGAs. ReConfig. (2010)
2. Bhattacharyya, S.S., Brebner, G.J., Janneck, J.W., Eker, J., von Platen, C., Mattavelli, M., Raulet, M.: OpenDF: A Dataflow Toolset for Reconfigurable Hardware and Multicore Systems. SIGARCH Computer Architecture News 36(5), 29–35 (2008)
3. Software from Celoxica, Inc., Handel-C (2005), `http://celoxica.com/technology/cdesign/handel-c.asp`
4. Gordon, M. I., Thies, W., Karczmarek, M., Lin, J., Meli, A. S., Lamb, A. A., Leger, C., Wong, J., Hoffmann, H., Maze, D., Amarasinghe, S. P.: A Stream Compiler for Communication-Exposed Architectures. In: ASPLOS, pp. 291–303 (2002)
5. Gokhale, M., Stone, J.M., Arnold, J.M., Kalinowski, M.: Stream-Oriented FPGA Computing in the Streams-C High Level Language. In: FCCM, pp. 49–58 (2000)

SIMPL Systems, or: Can We Design Cryptographic Hardware without Secret Key Information?

Ulrich Rührmair

Computer Science Department
Technische Universität München
Boltzmannstr. 3
85748 Garching
ruehrmai@in.tum.de
http://www.pcp.in.tum.de

Abstract. This paper discusses a new cryptographic primitive termed *SIMPL system*. Roughly speaking, a SIMPL system is a special type of Physical Unclonable Function (PUF) which possesses a binary description that allows its (slow) public simulation and prediction. Besides this public key like functionality, SIMPL systems have another advantage: No secret information is, or needs to be, contained in SIMPL systems in order to enable cryptographic protocols — neither in the form of a standard binary key, nor as secret information hidden in random, analog features, as it is the case for PUFs. The cryptographic security of SIMPLs instead rests on (i) a physical assumption on their unclonability, and (ii) a computational assumption regarding the complexity of simulating their output. This novel property makes SIMPL systems potentially immune against many known hardware and software attacks, including malware, side channel, invasive, or modeling attacks.

1 Introduction

Background and Motivation. Electronic communication and security devices are pervasive in our life. Just to name two examples, around five billion mobile phones are currently in use worldwide [1] [2], and the world market of smart cards has an estimated volume of over three billion pieces per year [3] [4]. Their widespread use makes such devices both a well-accessible and a worthwhile target for adversaries. Many security attacks are thereby not directed against the employed cryptographic primitives themselves, some of which have proven attack-resilient over surprisingly long time spans. Rather, they often apply physical or software attacks in order to extract the employed secret keys. Such key-extracting strategies are not just a theoretical concern, but have been demonstrated several times in widespread, commercial systems [5] [6] [7]. This drives the search for new mechanisms that protect — or better still: avoid! — secret keys in vulnerable hardware.

Physical Unclonable Functions (PUFs). The security primitive of a Physical Unclonable Function (PUF) [8] [9] [10] [11] was introduced, at least in part, in order to address some of the above problems. A PUF is a (partly) disordered physical system S that can be challenged with so-called external stimuli or challenges C_i, upon which it reacts with corresponding responses termed R_{C_i}. Contrary to standard digital systems, a PUF's responses shall depend on the nanoscale structural disorder present in the PUF. This disorder cannot be cloned or reproduced exactly, not even by its original manufacturer, and is unique to each PUF. Assuming the stability of the PUFs responses, any PUF S hence implements an individual function F_S that maps challenges C_i to responses R_{C_i} of the PUF. Due to its complex and disordered structure, a PUF can avoid some of the shortcomings associated with digital keys. For example, it is usually harder to read out, predict, or derive its responses than to obtain the values of digital keys stored in non-volatile memory. This fact has been exploited for various PUF-based security protocols [8] [9] [15] [22].

One prominent example are PUF-based identification schemes [8] [9] [10]. They are usually run between a central authority (CA) and a hardware carrying a (unique) PUF S. One assumes that the CA had earlier access to S, and could establish a large, secret list of challenge-response-pairs (CRPs) of S. Whenever the hardware wants to identify itself to the CA at some later point in time, the CA selects some CRPs at random from this list, and sends the challenges contained in these CRPs to the hardware. The hardware applies these challenges to S, and sends the obtained responses to the CA. If these responses match the pre-recorded responses in the CRP-list, the CA believes the identity of the hardware. Note that each CRP can only be used once, whence the CRP-list uses up over time, and needs to be large.

Private Key like Functionality of PUFs. The described protocol has several well-known advantages [8] [9]. However, one potential downside is that it presumes a previously shared piece of secret numerical information (i.e., the CRP-list). This information needs to be established in a secure set-up phase between the CA and the hardware, and must constantly be kept secret. In this particular structural aspect, PUFs are resemblant of classical private key systems.

Secret Information in PUFs. Another noteworthy point is that PUFs in general do not obviate the presence of secret information within cryptographic hardware. The secret information is no longer stored in digital form in two-level systems, such as digital secret keys stored in non-volatile memory cells. But still there is some sort of secret information present in most PUFs, whose disclosure breaks the security of the system. Let us name two examples to illustrate our point: In the case of SRAM PUFs the information that needs to be kept secret is the state of the SRAM cells after power up, or the tiny manufacturing variations of the SRAM cells that determine their state after power up [25]. Once this information is known to an adversary, he can numerically derive the same key as the cryptographic hardware embedding the SRAM PUF. In the case of Arbiter PUFs, the secret information are the internal runtime delays in the circuit stages [11]. If this information is known, the adversary can numerically simulate the behavior of the PUF output by an additive, linear model, again breaking its security [26].

In other words, the architectures of most current PUFs "hide" or "obfuscate" secret, security-relevant information very well in analog characteristics of integrated circuits. But at the same time, they do not avoid the need for secret information in hardware systems in principle; they just store it in a different form.

Our Contributions. This paper introduces a novel security primitive called a *SIMPL system*, whereby the acronym SIMPL stands for “SIMulation Possible, but Laborious”. These systems have two interesting conceptual advantages: First, they are a PUF-like security tool, but possess some type of public key functionality. This improves their practical applicability. Second, they obviate the need for secret information in cryptographic systems, trading it for two other assumptions: (i) their physical unclonability, and (ii) the assumed computational overhead of numerically simulating their output (in comparison with their faster real-time behavior). We show that SIMPLs can realize basic communication protocols such as identification and message authentication, and briefly describe the application of these protocols in some concrete settings. We also discuss implementations of SIMPL systems, thereby surveying existing approaches, and propose a new optical implementation strategy. Proof-of-concept data on this optical implementation, which arose from other, recent research activities in our group [40], is presented in the appendix.

Related Work. The current paper is an extended version of [16]. Since the publication of [16], several papers of our group have dealt with the implementation of SIMPLs by integrated circuits [17] [18] [19] [20]. We emphasize that around the same time as [16], a comparable concept has been described independently in [21] under the name of a Public PUF (PPUF).

While stressing that both pieces of work are very interesting, let us briefly address a few differences between [21] and our studies. One difference is that we focus on SIMPL systems/Public PUFs for which the *relative* speed difference between the real hardware and the simulation is comparably low, for example only a small constant factor. Such systems seem to have milder complexity requirements and less stability issues. We argue that by applying feedback loops, not the relative, but the absolute time difference between such systems and any emulation can still be amplified to a sufficient absolute value. Once this absolute value is large enough, it enables secure identification and message authentication protocols, and could compensate network or other delays. Another reason for concentrating on systems with small speed gaps lies in the fact that the verification step in identification and message authentication must be carried out relatively efficiently (see Protocols 2 and 3).

Second, we center upon applications where the main advantage of SIMPL systems — that they can build security systems without secret key information — is most relevant. Two typical examples are the named identification and message authentication schemes. Should a shared secret key between two parties be required in a SIMPL-based communication infrastructure (for example in order to achieve confidentiality), SIMPL-based message authentication can be used together with the Diffie-Hellman protocol to exchange a session key. But this key ideally will not be stored permanently in the system.

To the contrary, [21] discuss a PPUF-scenario where a one-time, permanent secret key is exchanged in a computationally relatively intensive scheme. This scheme appears too time consuming for multiple session key exchange. Their setting hence puts key exchange on different security assumptions than classical protocols (like Diffie-Hellman), which is a strong achievement on its own. But they do not attempt to generally avoid the long-term presence of secret information in cryptographic hardware, as we aspire with SIMPL systems.

Finally, a very interesting and recommendable, but later source is [24], where time-bounded authentication for FPGAs is discussed.

Organization of this Paper. The rest of this manuscript is organized as follows: In Section 2, we give a semi-formal specification of SIMPL systems, and discuss their properties. Section 3 provides two formal SIMPL-based protocols for entity identification and message authentication. In Section 4 we discuss implementation candidates for SIMPL systems, and conclude the paper in Section 5.

2 SIMPL Systems and Their Properties

2.1 Informal Description of SIMPL Systems

We start by informally listing the properties of a SIMPL system. A physical system S is called a *SIMPL system* (or just a *SIMPL*) if the following holds:

1. S is a (partly) disordered physical system. It can be stimulated with challenges C_i, upon which it reacts with corresponding responses R_{C_i}. The responses are a function of the specific disorder present in S, and of the applied challenge C_i. The responses are assumed to be sufficiently stable to regard the behavior of S as a function F_S that maps challenges C_i to responses R_{C_i}.
2. Given a challenge C_i, it is possible to numerically simulate the corresponding response R_{C_i} of S with high accuracy. The simulation is carried out via an individual, public description $D(S)$ of S, and a public simulation algorithm Sim.
3. Any feasible algorithm, or any physical emulation, that predicts the responses of S correctly (i.e., which computes F_S), is noticeably slower than the real-time behavior of S.
4. It is difficult to physically clone S, i.e. to produce a second system S' which generates the same responses on almost all possible challenges with comparable speed. This must hold even if the internal characteristics and disorder of S, the description $D(S)$, and many CRPs of S are known.

Put in one sentence, the holder of a secure SIMPL system S is able to evaluate a publicly known, publicly computable individual function F_S *faster* than anyone else.

2.2 Semi-formal Specification of SIMPL Systems

The above properties can also be coined into a semi-formal specification of SIMPL systems. The style of the specification follows the specifications and definitions that have been presented in [22]. It specifies the security of SIMPL systems as a "game" with the adversary, thereby introducing a relatively precise adversarial model.

Specification 1 (**$(t_{max}, c, t_C, t_{Ph}, q, \epsilon)$-SIMPL SYSTEMS.**). *Let S be a physical system mapping challenges C_i to responses R_{C_i}, with $\mathbf{C}$ denoting the finite set of all possible challenges. Let $c > 1$ be a constant, and let furthermore t_{max} be the maximum time (over all challenges $C_i \in \mathbf{C}$) which it takes until the system S has generated the response R_{C_i} to the challenge C_i. S is called a $(t_{max}, c, t_C, t_{Ph}, q, \epsilon)$-*SIMPL SYSTEM *if there is a string $D(S)$, called the description of S, and a computer algorithm* Sim *such that the following conditions are met:*

1. *For all challenges $C_i \in \mathbf{C}$, the algorithm* Sim *on input* $\big(C_i,\ D(S)\big)$ *outputs* R_{C_i} *in feasible time.*
2. *Any cryptographic adversary Eve will succeed in the following* **security experiment** *with a probability of at most* ϵ*:*
   - *(a) Eve is given the numerical description* $D(S)$ *and the code of the algorithm* Sim *for a time period of length* t_C*.*
   - *(b) Within the above time period* t_C*, Eve can* q *times adaptively query an oracle* O *for arbitrary responses* R_{C_i} *of* S *.*
   - *(c) Within the above time period* t_C*, Eve is furthermore given physical access to the system* S *at adaptively chosen time points, and for time periods of adaptively chosen lengths. The only restriction is that her access times must add up to a total of at most* t_{Ph}*.*
   - *(d) After the time period* t_C *has expired, Eve is presented with a challenge* C_{i_0} *that was chosen uniformly at random from the set* $\mathbf{C}$*, and is asked to output a value* V_{Eve}*.*

   We say that Eve succeeds in the described experiment if the following conditions are met:

   - *(i)* $V_{Eve} = R_{C_{i_0}}$*.*
   - *(ii) The time that Eve needed to output* V_{Eve} *after she was presented with* C_{i_0} *was at most* $c \cdot t_{max}$*.*

   Please note that the said probability of ϵ *is taken over the uniformly random choice of* $C_{i_0} \in \mathbf{C}$*, and the random choices or actions that Eve might take in steps 2a, 2c and 2d.*

Discussion. Let us briefly discuss the security model underlying Specification 1. In practical applications of SIMPL systems, Eve can gather information about S in three ways: (i) She analyzes the algorithm Sim and the description $D(S)$, which are both public. (ii) She collects as many challenge-response-pairs (C_i, R_{C_i}) of S as possible from external sources, for example protocol eavesdropping. (iii) Eve *physically* measures the system S. She may determine CRPs by such measurements, but also other, more general characteristics of the system.

These three types of attacks must be covered in our security model, and they are: Possibility (i) is covered in item 2a of Spec. 1, (ii) is reflected in item 2b, and (iii) is implicit in item 2c. Since the physical access time and the time in which Eve can prepare her attack by previous computations differ strongly in most application scenarios, it makes sense to distinguish between t_{Ph} and t_C in Spec. 1.

We also chose the value c, which describes the time gap between Eve and the SIMPL system, to be a flexible system parameter. This keeps the definition general and allows its application to different types of SIMPLs. In many practical applications, even small values (e.g. around 2) may suffice for c. See also the discussion in Section 2.3, paragraphs on *constant vs. super-polynomial time gap* and *feedback loops*.

2.3 Properties of SIMPL Systems

Let us discuss a few properties of SIMPL systems implied by Specification 1.

Immunity against ϵ-fraction Read-out and Simulation. Spec. 1 implies that for any SIMPL system it must be impossible to measure the values R_{C_i} for more than an ϵ-fraction of all parameters $C_i \in \mathbf{C}$ within time t_{Ph}. Otherwise, Eve could create a lookup table for an ϵ-fraction of all possible values R_{C_i} during step 2c. This could enable her to succeed in the described experiment with probability greater than ϵ. Therefore, for any SIMPL system the set of possible measurement parameters $\mathbf{C}$ must be very large.

For the same reasons, it must be impossible for Eve to determine more than an ϵ-fraction of all CRPs within time t_C by exhaustive simulation on the basis of Sim and $D(S)$. This again implies that $\mathbf{C}$ must be very large (for example exponential in some system parameter), and/or that the simulation must be time consuming.

Immunity against Cloning. Spec. 1 also implies that previous physical access for time t_{Ph} and computations of time t_C do not allow Eve to build a *physical clone* S' of the system S, for whose responses R'_{C_i} it holds that

$$R_{C_i} = R'_{C_i} \quad \text{for more than an } \epsilon\text{-fraction of all } C_i \in \mathbf{C},$$

and for which the evaluation of the R'_{C_i} works within time $c \cdot t_{max}$. Spec. 1 both rules out the possibility to build an exact physical reproduction of S, or the feasibility to fabricate a *functional* clone, i.e., a physical system of a possibly very different structure or lengthscale than S, which still generates its response R'_{C_i} within time $c \cdot t_{max}$.

Constant vs. Super-polynomial Time Gap. Spec. 1 stipulates that the time gap between Eve and the real SIMPL system S must be at least a constant factor $c > 1$. This seems surprising: Being used to the formalism of complexity-based classical cryptography, one might expect the stipulation of an exponential gap. But it is unclear whether SIMPLs with an exponential time margin between Eve and the SIMPL exist at all. The only known, realistic computational systems which might outperform Turing architectures by a super-polynomial factor are quantum computers [35]. But standard quantum computers possess no immunity against physical cloning, since they could be mass-fabricated with the same functionality. They are hence unsuited as SIMPL systems. A further setback in the search for SIMPLs with an exponential security margin is that it has been frequently hypothesized within the computational complexity community that there are no realistic hardware systems that solve NP-complete problems efficiently in practice, i.e. by using polynomial resources. Two recent sources in this context are [33] [34].

Still, meaningful applications for SIMPL systems may not require exponential speed gaps. In the appliances we suggest in this paper (namely identification and on-the-fly message authentication), a constant, detectable time difference suffices. An exponential time gap between the SIMPL system and any simulation machine may even be undesirable there, since it could lead to time consuming verification steps in the Protocols 2 and 3.

Feedback Loops. In order to enable large absolute time margins, the absolute (but not the relative!) time difference between the original SIMPL system and any fraudster can be amplified via feedback loops. In a nutshell, such feedback-loops can be set up as follows: Presented with a challenge C_1, the SIMPL systems successively determines a sequence of k challenge-responses-pairs $(C_1, R_{C_1}), (C_2, R_{C_2}), \ldots, (C_k, R_{C_k})$, where later challenges C_n are determined by earlier results R_{C_m}, with $k \geq n > m \geq 1$. The tuple (C_1, R_{C_k}) can then be regarded as the overall challenge-response pair determined by the SIMPL. The application of such feed-back loops can help us to compensate network and transmission delays.

Let us make a concrete example in order to illustrate our point. Suppose that we possess a SIMPL system S which produces its responses in t_{max} of 10 nanoseconds (ns), and which possesses a speed advantage of $c = 2$ over all simulations. That means that any adversary cannot produce the response to a randomly chosen challenge within 20 ns. This tiny difference would not be detectable in many practical settings, for example in large networks with natural delays. Nevertheless, the application of repeated feedback loops can amplify not the relative, but the absolute time margin, such as to 1 millisecond (ms) vs. 2 ms, or also 1 sec vs. 2 sec.

SIMPLs with Multi-bit Output. In some applications, it is found convenient if a SIMPL system produces not just one bit as response, but a multi-bit output. Some implementations of SIMPLs have this property naturally (for example the optical implementation of section 4.3). Otherwise, feedback loops can allow us to create multi-bit outputs from SIMPL systems with 1-bit outputs: One simply considers a concatenation (or some other function, for example a hash function) of the last n responses $R_{C_{k-n+1}}, \ldots, R_{C_k}$ in the feedback loop. This concatenation (or function) can be taken as the overall output of the SIMPL.

Another option to create "large" SIMPL systems with k-bit outputs from "small" SIMPL systems with 1-bit outputs is to employ k such SIMPL systems in parallel, and to directly concatenate their responses to produce a k-bit overall output. This method has been suggested already in the context of PUFs in [13].

Error Correction. Please note that in the Spec. 1, in the above discussion, and also in the upcoming protocols in Section 3, we assume that the responses of the SIMPL system are stable. In practice, error correction and helper data must, and can, be applied to achieve this goal; see, for example, [9] [37] [38] [39].

3 Protocols and Applications

We will now describe two exemplary protocols that can be realized by SIMPL systems, and discuss some application scenarios.

3.1 Identification of Entities

We assume that Alice holds an individual $(t_{max}, c, t_C, t_{Ph}, q, \epsilon)$-SIMPL system S, and has made the corresponding data $D(S)$, Sim, the value $c \cdot t_{max}$, and a description of $\mathbf{C}$ public. Now, she can prove her identity to an arbitrary second party Bob, who knows $D(S)$, Sim, $c \cdot t_{max}$ and $\mathbf{C}$, as follows (with k being the security parameter of the protocol):

Protocol 2: IDENTIFICATION OF ENTITIES BY SIMPL SYSTEMS

1. Bob chooses k challenges $C_1, \ldots, C_k$ uniformly at random from $\mathbf{C}$.
2. **For** $i = 1, \ldots, k$ **do**:
   (a) Bob sends the value C_i to Alice.
   (b) Alice determines the corresponding response R_{C_i} by an experiment on her SIMPL system S, and sends this value to Bob.
   (c) Bob receives an answer from Alice, which we denote by V_i. If Alice's answer did not arrive within time $c \cdot t_{max}$, then Bob sets $V_i = \perp$ and continues the for-loop.
3. Bob computes the value $R_{C_i}^{Sim} = \mathsf{Sim}(C_i, D(S))$ for all $i = 1, \ldots, k$, and verifies if $R_{C_i}^{Sim} = V_i \neq \perp$. If this is the case, Bob believes Alice's identity, otherwise not.

Discussion. In a nutshell, the security of the protocol follows from the fact that an adversary is unable to determine the values R_{C_i} for randomly chosen C_i comparably quickly as Alice, provided that: (i) The lifetime of the system S (and the period since $D(S)$ was made public) does not exceed t_C, and (ii) the adversary's accumulated physical access times do not exceed t_{Ph} (see Spec. 1). In that case, the adversary's probability to succeed in the protocol without possessing S decrease exponential in k.

Bob can improve his computational efficiency by verifying the correctness of the responses R_{C_i} only for a randomly chosen subset of all responses. If necessary, possible network and transmission delays can be compensated for by amplifying the absolute time gap between Eve and S through feedback loops (see Section 2.3).

If the SIMPL system has multi-bit output, then a value of $k = 1$, i.e. a protocol with one round, may suffice. In these cases, the parameter ϵ of the multi-output SIMPL system will in itself be exponentially small in some system parameter (for example in the size of the sensor array in the optical SIMPLs discussed in Section 4).

3.2 Authentication of Messages

Alice can also employ an individual $(t_{max}, c, t_C, t_{Ph}, q, \epsilon)$-SIMPL system S in her possession to authenticate messages to Bob. Again, we suppose that the values $D(S)$, Sim, $c \cdot t_{max}$, and a description of $\mathbf{C}$ are public.

Protocol 3: AUTHENTICATION OF A MESSAGE N BY SIMPL SYSTEMS

1. Alice sends the message N, which shall be authenticated, to Bob.
2. Bob chooses $k \cdot l$ challenges $C_1^1, \ldots, C_k^1, C_1^2, \ldots, C_k^2, \ldots, C_1^l, \ldots, C_k^l$ uniformly at random from $\mathbf{C}$.
3. **For** $i = 1, \ldots, l$ **do**:
   (a) Bob sends the values $C_1^i, \ldots, C_k^i$ to Alice.
   (b) Alice determines the corresponding responses $R_{C_1^i}, \ldots, R_{C_k^i}$ by experiments on her SIMPL system S.
   (c) Alice derives a MAC-key K_i from $R_{C_1^i}, \ldots, R_{C_k^i}$ by a publicly known procedure, for example by applying a publicly known hash function to these values. She sends $MAC_{K_i}(N)$ to Bob.

   (d) Let us denote the answer Bob receives from Alice by V_i. If V_i did not arrive in time $c \cdot t_{max} + t_{MAC}$, where t_{MAC} is the time to derive K_i and compute $MAC_{K_i}(N)$, then Bob sets $V_i = \perp$ and continues the for-loop.
4. For $i = 1, \ldots, k$ and $j = 1, \ldots, l$, Bob computes the values $R^{Sim}_{C_i^j} = \mathsf{Sim}(C_i^j, D(S))$ by simulation via Sim. He derives the keys $K_1^{Sim} \ldots, K_k^{Sim}$ by application of the same procedure (e.g. the same publicly known hash function) as Alice in step 3c.
5. For all $i = 1, \ldots, k$, Bob checks if it holds that $MAC_{K_i^{Sim}}(N) = V_i \neq \perp$. If this is the case, he regards the message N as properly authenticated, otherwise not.

Discussion. In a nutshell, the security of the protocol follows from the fact that an adversary cannot determine the responses $R_{C_i^j}$ and the MAC-Keys $K_1, \ldots, K_l$ as quickly as Alice. As earlier, verification of a randomly chosen subset of all MACs can improve Bob's computational efficiency in step 5. Depending on the exact circumstances, a few erroneous V_i may be tolerated in step 5, too.

We assume without loss of generality that the MAC can be computed quickly (including the derivation of the MAC keys $K_1, \ldots, K_l$), i.e., within time t_{MAC}, and that t_{MAC} is small compared to t_{max}. Again, this condition could be realized by amplification through feedback loops if necessary (see Section 2.3). Furthermore, it is known that MACs can be implemented very efficiently [27]. If information-theoretically secure hash functions and MACs are used, the security of the protocol will not depend on any assumptions other than the security of the SIMPL system.

If the SIMPL system has a multi-bit output, then values of $k = 1$, i.e., sending just one challenge in each round, or of $l = 1$, i.e., employing just one round of communication, may suffice. Such a multi-bit output can arise either naturally, for example through the choice of the SIMPL system itself (as noted earlier, the optical SIMPL system presented in Section 4.3 has this property). Or it can be enforced by feedback loops, or by using several independent SIMPL systems in parallel (see Section 2.3, page 32). In fact, such measures even are strictly necessary to uphold the protocol's security if the constant c has got a very low value.

3.3 Application Scenarios

Secure Communication Infrastructures. Within the given space restrictions, we will now discuss the application of SIMPL systems to secure communication in networks, illustrating their potential in such a setting. Consider a situation where k parties $P_1, \ldots, P_k$ and a trusted authority TA participate in a communication network. Assume that each party P_i carries its own SIMPL S_i in its hardware, and that a certificate C_i has been issued for each party by the TA. The certificate includes the identity and the rights of Party P_i, and has the form

$$C_i = \big(Id_i, Rights_i, D(S_i), Sig_{TA}(Id_i, Rights_i, D(S_i))\big).$$

Under these provisions, the parties can mutually identify themselves by Protocol 2, they can establish authenticated channels with each other by Protocol 3, and they can exchange session keys via the Diffie-Hellman protocol [32] over these authenticated channels. The whole architecture works without permanent secret keys, or without any other secret information that is stored permanently in the hardware of the parties $P_1, \ldots, P_k$.

It also seems well applicable to cloud computing: All personal data could be stored centrally. Session keys could be exchanged by the Diffie-Hellman protocol over channels authenticated by the SIMPL systems. These keys can be used to download the personal data in encrypted form from the central storage. The keys can be new in each session, no permanent secret keys in the mobile hardware are be necessary.

The above approaches can further be combined with tamper-sensitive SIMPL systems. These SIMPLs may cover hardware which has a functionality $Func_i$ as long as it is non-manipulated. Each certificate C_i could then also include the functionality of the hardware, i.e., it could be of the form

$$C_i = \big(Id_i, Rights_i, Func_i, D(S_i), Sig_{TA}(Id_i, Rights_i, Func_i, D(S_i))\big).$$

By running the identification protocol (Prot. 2), party P_i can prove to party P_j that the SIMPL system S_i is non-tampered, and that the hardware hence has the claimed functionality $Func_i$. Please note that the optical SIMPL systems we propose in this paper is naturally tamper sensitive; the tamper sensitivity of such optical scattering structures has already been shown in detail in [8].

Two other Applications. Let us, in all brevity, point to two other applications of SIMPL systems. They are described in more detail in [16].

A first application is the generation of unforgeable labels for products or security tokens. SIMPL systems can create labels which do not contain any secret information, which can be verified offline, and which only require remote, digital communication between the label and a testing device. These properties are not met by other known labeling techniques: RFID-tags with secret keys obviously contain secret information; PUF-based labels contain secret information in the case of Weak PUFs, and require an online database in the case of Strong PUFs [8]; and current Certificates of Authenticity (COAs) [28] [30] require analog near-field measurements in the verification step.

Another application area of SIMPLs lies in the context of the digital rights management problem. SIMPLs can create unclonable representations of digital content [16]. Similar to the unforgeable labels, these unclonable representations of digitial content do not contain any secret information. They can be verified for their validity offline and by mere digital communication between a tester and the device carrying the unclonable representation. Again, in combination these features are not met by any comparable technique known to the author. In [29] [30] [31], for example, the random features of the data carrier must be determined in the near-field by analog measurements.

4 Implementation of SIMPL Systems

Let us now turn to the practical implementation of SIMPL systems. We will give an overview of existing ideas and challenges, and propose one new, optical concept.

4.1 Challenges

It turns out that there are some strong challenges in the realization of SIMPL systems. The three non-trivial requirements that need to be balanced are complexity, stability, and simulatability: On the one hand, the output of a SIMPL system must be sufficiently complex to require a long computation/simulation time. On the other hand, it must be simple enough to allow simulation at all, and to enable the determination of $D(S)$ by measurement or numeric analysis techniques. A final requirement is that the simulation can be carried out *relatively* efficiently by everyone (this is necessary to complete the verification steps in the identification and message authentication protocols quickly); while, at the same time, even a very well equipped attacker, who can potentially attempt to parallelize the simulation on many powerful machines, cannot simulate as fast as the real-time behavior of the SIMPL system. In the sequel, we will discuss a few implementations that try to meet these seemingly conflicting requirements.

4.2 Electrical SIMPL Systems

Since the first publication of [16], a sequence of papers of our group has dealt with the implementation of SIMPL systems by electrical, integrated circuits [17] [18] [19] [20]. We tried to exploit two known speed bottlenecks of modern CPUs: Their problems in dealing simultaneously with very large amounts of data, and the complexity of simulating analog, parallel phenomena. Let us briefly summarize these approaches, quoting from said papers.

"Skew" SRAM Memories. A first suggestion made in [17] [18] [19] [20] is to employ large arrays of SRAM cells with a special architecture named "skew design". In this design, the read- and write behavior of the cells is dependent on the applied operational voltage. The simulation of a skew SRAM memory in a feedback loop of a very large number of successive read- and write events then seems somewhat laborious to simulate on a standard architecture. The hypothesis put forward in [17] [18] [19] [20] is that this creates a small, constant simulation overhead. Two essential assumptions in this concept are: (i) No parallelization is possible, since the successive read- and write events in the feedback loop are made dependent on the previous read results. And (ii), since no parallelization is possible, the limiting factor for an adversary is his clock frequency, which is quite strongly limited by current technology.

As described in the listed references, the idea shows strong promise to succeed against any adversaries with a limited financial budget, and in particular against any FPGA-based attacks. Future work will need to show how large the exact simulation margin is, and whether it is indeed sufficient to defeat an adversary with large resources, who is capable of fabricating ASICs. Due to its relatively easy realizability and good security level, the idea could have a strong potential for the consumer market.

Two-dimensional Analog Computing Arrays. A second suggestion of [17] [18] [19] [20] consists of using analog, two-dimensional computing arrays. The authors suggest the use of so-called cellular non-linear networks (CNNs) which are designed to imitate non-linear optical systems. Due to their analog and inherently parallel nature (many cells exchange information at the same time), it is suggested that CNNs are time consuming to simulate on a digital, sequential architecture.

This idea has its assets on the security side: Since it is based on manufacturing mismatches in CNN fabrication that currently seem unavoidable, it shows promise of defeating even attackers with very strong financial resources, and of being manufacturer resistant in the sense of [23]. It requires the use of analog circuits, though, which might potentially be unsuited for low-cost applications.

Other Approaches. Independently, the work of other groups has lead to different structures that could be used as SIMPLs. The implementation of PPUFs presented in [21] could potentially be downscaled to become a SIMPL system, even though it would have to be carefully investigated how resilient such small-scale instances are against parallelization attacks. Another very interesting, FPGA-based candidate for SIMPLs is implicit in the work of [24].

4.3 Integrated Optical SIMPLs

Also optical structures can be used as SIMPL systems. The rationale behind employing optics is as follows: First, optical systems can potentially achieve faster component interaction than electronic systems; this promises to create the desired speed advantage over any electronic simulator. The phenomenon of optical interference has no electronic analog at room temperature [41], and can create a computational overhead for electronic simulators. Second, the material degradation of optical systems is low, and their temperature stability is known to be high [41] [42]. Even very complex and randomly structured optical systems, whose internal complexity creates the desired speed gaps, can produce outputs that are stable against aging and environmental conditions.

The concrete optical SIMPL we suggest is depicted schematically in Figure 1. It comprises of an immobile laser diode array with k phase-locked diodes $D_1, \ldots, D_k$ [43], which is used to excite a disordered, random scattering medium. The diodes can be switched on and off independently, leading to 2^k challenges C_i. These can be written as $C_i = (b_1, \ldots, b_k)$, where each $b_i \in \{0, 1\}$ indicates whether diode D_i is switched on or off. (Note that the diode array must indeed be phase locked in order to allow interference of the different diode signals.) At the right hand side of the system, an array of l light sensors $S_1, \ldots, S_l$, e.g. photodiodes, measures the resulting light intensities locally. A response R_{C_i} consist of the intensities $I_1, \ldots, I_l$ in the l sensors. Instead of phase-locked diode arrays, also a single laser source with a subsequently placed, inexpensive light modulator (as contained in any commercially available beamer) can be employed.

Under the provision that a *linear* scattering medium is used in such integrated optical SIMPLs, the following analysis holds[44]. Every diode D_i with $b_i = 1$ creates a

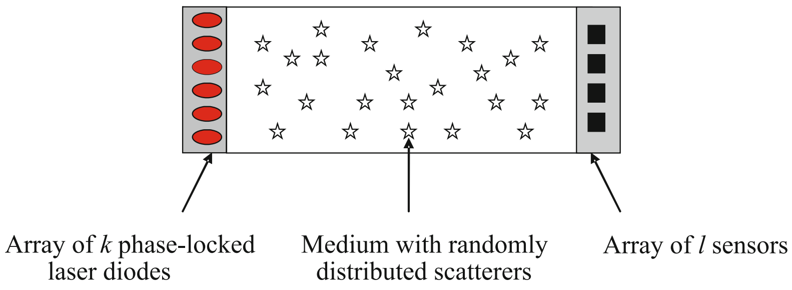

Array of k phase-locked laser diodes
Medium with randomly distributed scatterers
Array of l sensors

Fig. 1. An integrated optical SIMPL system

lightwave, which is scattered in the medium and arrives at the sensor S_j with amplitude E_{ij} and phase shift θ_{ij}. The intensity I_j at the sensor S_j is then given by [42]

$$I_j = \left|E_j\right|^2 = \left|\sum_i b_i\, E_{ij}\, \cos\, \theta_{ij}\right|^2. \tag{1}$$

For the said linear scattering medium, the amplitude E_{ij} and phase shift θ_{ij} are independent of whether the other diodes are switched on or off. One can hence collect many CRPs

$$(C_m, R_{C_m}) = ((b_1, \ldots, b_k), (I_1, \ldots, I_l)),$$

and derive the values E_{ij} and θ_{ij} from knowledge of these many (C_m, R_{C_m}). One suited approach are machine learning techniques, for example a standard machine learning regression.

Once the parameters E_{ij} and θ_{ij} are known, the simulation of a response $R_{C_m} = (I_1, \ldots, I_l)$ to a given challenge $C_m = (b_1, \ldots, b_k)$ can be executed by simple calculation following Eqn. 1. The time margin to the real system will be small, but likely detectable: The real system creates its output and the complex interference in nanoseconds, while the calculation of Eqn. 1 requires around $k \cdot l$ multiplications and $k \cdot l$ additions. Some of these computations can be parallelized, and the values $E_{ij} \cdot \cos\theta_{ij}$ can be precomputed. Still, even for a moderate size of the two-dimensional diode and sensor arrays of around $100 \times 100 = 10^4$ each, the number of additions is on the order of 10^8. This seems to create exactly the constant, notable time gap that we require in SIMPLs.

A first proof-of-concept for this integrated optical approach, which is not optimized in terms of speed, but shows the feasibility of the output simulation/prediction on the basis of real data, is given in the appendix.

4.4 Further Implementation Strategies

Let us discuss a few further implementation strategies for SIMPLs.

Employing PUFs with Reduced Complexity. One generic strategy for the realization of SIMPL systems, which has been suggested already in [16], is the following: Employ a PUF or a PUF-like structure; and reduce its inner complexity until it can be characterized by measurements and simulated, or until it can successfully be machine learned. If the level of complexity is still sufficient, then this simulation will be more time consuming than the real-time behavior of the system. In fact, the suggestions of the previous subsections used this strategy already, since both CNNs and integrated optical structures have already been suggested as PUFs in earlier work [36] [12]. But also any other PUFs could be used in this strategy, for example Pappu's original optical PUF with a reduced number of scatterers [8], as suggested in [16].

Simulation vs. Verification. Another interesting idea is to exploit the well-known asymmetry between actively computing a solution for a certain problem and verifying the correctness of a proposed solution (as also implicit in the infamous P vs. NP question) [16]. Exploiting this asymmetry could lead to protocols of the following kind: A SIMPL system provides the verifier in an identification/authentication protocols with some extra information that allows the verifier to *verify* its answers fast. To illustrate our point, imagine an analog, two-dimensional, cellular computing array whose behavior is governed by partial differential equations (PDEs), such as the CNN described in section 4.2. Then, verifying the validity of a given final state of such a PDE-driven system (i.e. verifying that this state is indeed a solution of the PDEs driving the system) could be much more time efficient than computing this solution from scratch. Furthermore, the verifier could not only be given external outputs of such a two-dimensional array (e.g. values in boundary cells), but also internal submeasurements (e.g. values in inner cells) that help him to verify the output quickly.

The simulation vs. verification strategy can help to relieve the seeming conflict between the requirement for fast simulation on the side of the verifier (who may not be well equipped on the hardware side) and the necessary time margin to an attacker (who may be very well equipped on the hardware side), which we already addressed in Section 4.1.

5 Summary, Discussion, and Future Work

Summary. This paper introduced a security concept termed "SIMPL Systems". We started by a explaining the basic idea and by giving a semi-formal specification of SIMPL systems. We subsequently discussed some basic properties that follow from this specification. We then presented two protocols that can be realized by SIMPL systems, namely identification and message authentication. These protocols exploit the fact that the holder of a SIMPL system is the only person who can determine the response of the SIMPL to a randomly chosen challenge within a certain time frame. We argued that the can be used to set up special, secure communication infrastructures which obviate the long-term storage of any form of secret keys in hardware. We listed other applications of SIMPL systems, for example as unforgeable labels and in the digital rights management problem.

We next discussed the practical implementation of SIMPL systems. We gave an overview of existing, electrical candidates, and then suggested a new optical implementation based on light scattering. We gave a proof-of-concept for this optical SIMPL by using data from a first prototype, which had been set-up by our group in a different context [40]. This data shows the general feasibility of predicting such systems, but was not yet optimized in terms of speed. We also presented generic and/or future implementation strategies for SIMPLs, for example the use of PUFs with reduced complexity, or exploiting the asymmetry between actively computing and merely verifying a solution to a given problem (as implicit in the well-known P vs. NP question).

Discussion. Let us conclude this work by a detailed comparative analysis. As said earlier, there are some similarities between classical private/public key cryptoschemes and SIMPL systems: The numeric description $D(S)$ is some analog to a public key, while the physical system S itself constitutes some functional equivalent to a private key. This provides SIMPLs with some public-key like functionality and with the resulting practicality advantages.

At the same time, there is one important difference to classical public-key systems: This new type of "private key" S is no secret numeric information, but a randomly structured, hard-to-clone *physical system*. It has the interesting feature of not containing any form of secret information. Neither in an explicit digital form like a digital key in classical hardware. Nor in a hidden, analog form such as internal PUF parameters (for example the mentioned delay values in the Arbiter PUFs, or the parameters determining SRAM behavior). All internal characteristics of a SIMPL, including its precise internal configuration, can be publicly known without compromising the security of the derived cryptographic protocols.

The security of SIMPL systems is not free of assumptions, though. Instead of presupposing the secrecy of some sort of information, it rests on the following two hypotheses: (i) on the computational assumption that no other, well-controllable, configurable, or even programmable hardware can generate the complex responses of a SIMPL with the same speed, and (ii) on the physical assumption that it is practically infeasible for Eve to exactly clone or rebuild the SIMPL system, even though she knows its internal structure and properties.[1]

It is long accepted that computational assumptions play a standard role in classical cryptography, and they are also a part of the security assumptions for SIMPL systems; but SIMPLs show that one can trade the need for secret information in the hardware against assumptions on the physical unclonability of the SIMPL system. This can surprisingly obviate the familiar requirement that cryptographic hardware must contain secret key information of some sort.

Future Work and Prospects. Future work on SIMPLs will likely concentrate on new protocols beyond identification and message authentication, and on formal security proofs for such protocols. But perhaps the greater challenge lies on the hardware side: Even though there are several promising candidates (see Section 4), the issue of finding a highly secure, practical, and cheap implementation of SIMPL systems appears not to be fully settled yet. If such an implementation is found, or if the existing implementation candidates are shown to possess all necessary properties, this could change the way we exercise cryptography today.

References

1. `http://www.cbsnews.com/stories/2010/02/15/business/main6209772.shtml`
2. `http://www.bbc.co.uk/news/10569081`
3. `http://www.eurosmart.com/images/doc/Eurosmart-in-the-press/2006/cardtechnologytoday_dec2006.pdf`
4. `http://www.gsaietsemiconductorforum.com/2010/delegate/documents/GASSELGSALondon20100518presented.pdf` (Slide 23)

[1] The reader can verify the plausibility of the latter unclonability property by considering the optical implementation of section 4.3: Even if the positions of all scattering centers and the other irregularities in the plastic matrix were known in full detail, it would still be infeasible to rebuild the whole system with perfect precision.

5. Eisenbarth, T., Kasper, T., Moradi, A., Paar, C., Salmasizadeh, M., Manzuri Shalmani, M.T.: On the power of power analysis in the real world: A complete break of the KEELOQ code hopping scheme. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 203–220. Springer, Heidelberg (2008)
6. Kasper, T., Silbermann, M., Paar, C.: All you can eat or breaking a real-world contactless payment system. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 343–350. Springer, Heidelberg (2010)
7. Anderson, R.J.: Security Engineering: A Guide to Building Dependable Distributed Systems, 2nd edn. Wiley, Chichester (2008) ISBN: 978-0-470-06852-6
8. Pappu, R., Recht, B., Taylor, J., Gershenfeld, N.: Physical One-Way Functions. Science 297, 2026–2030 (2002)
9. Pappu, R.: Physical One-Way Functions, PhD Thesis, MIT
10. Gassend, B., Clarke, D.E., van Dijk, M., Devadas, S.: Silicon physical random functions. In: ACM Conference on Computer and Communications Security 2002, pp. 148–160 (2002)
11. Gassend, B., Lim, D., Clarke, D., Dijk, M.v., Devadas, S.: Identification and authentication of integrated circuits. Concurrency and Computation: Practice & Experience 16(11), 1077–1098 (2004)
12. Tuyls, P., Skoric, B.: Strong Authentication with Physical Unclonable Functions. In: Petkovic, M., Jonker, W. (eds.) Security, Privacy and Trust in Modern Data Management, Springer, Heidelberg (2007)
13. Edward Suh, G., Devadas, S.: Physical Unclonable Functions for Device Authentication and Secret Key Generation. In: DAC 2007, pp. 9–14 (2007)
14. Gassend, B., Dijk, M.v., Clarke, D.E., Torlak, E., Devadas, S., Tuyls, P.: Controlled physical random functions and applications. ACM Trans. Inf. Syst. Secur. 10(4) (2008)
15. Rührmair, U.: Oblivious Transfer based on Physical Unclonable Functions (Extended Abstract). In: Acquisti, A., Smith, S.W., Sadeghi, A.-R. (eds.) TRUST 2010. LNCS, vol. 6101, pp. 430–440. Springer, Heidelberg (2010)
16. Rührmair, U.: SIMPL Systems: On a Public Key Variant of Physical Unclonable Functions. Cryptology ePrint Archive, Report 2009/255 (2009)
17. Rührmair, U., Chen, Q., Lugli, P., Schlichtmann, U., Stutzmann, M., Csaba, G.: Towards Electrical, Integrated Implementations of SIMPL Systems. Cryptology ePrint Archive, Report 2009/278 (2009)
18. Chen, Q., Csaba, G., Ju, X., Natarajan, S.B., Lugli, P., Stutzmann, M., Schlichtmann, U., Rührmair, U.: Analog Circuits for Physical Cryptography. In: 12th International Symposium on Integrated Circuits (ISIC 2009), Singapore, December 14-16 (2009)
19. Rührmair, U., Chen, Q., Stutzmann, M., Lugli, P., Schlichtmann, U., Csaba, G.: Towards electrical, integrated implementations of SIMPL systems. In: Samarati, P., Tunstall, M., Posegga, J., Markantonakis, K., Sauveron, D. (eds.) WISTP 2010. LNCS, vol. 6033, pp. 277–292. Springer, Heidelberg (2010)
20. Chen, Q., Csaba, G., Lugli, P., Schlichtmann, U., Stutzmann, M., Rührmair, U.: Circuit-based Approaches to SIMPL Systems. Accepted by Journal of Circuits, Systems and Computers (2010) (to appear)
21. Beckmann, N., Potkonjak, M.: Hardware-Based Public-Key Cryptography with Public Physically Unclonable Functions. In: Katzenbeisser, S., Sadeghi, A.-R. (eds.) IH 2009. LNCS, vol. 5806, pp. 206–220. Springer, Heidelberg (2009)
22. Rührmair, U., Busch, H., Katzenbeisser, S.: Strong PUFs: Models, Constructions and Security Proofs. In: Sadeghi, A.-R., Naccache, D. (eds.) Towards Hardware Intrinsic Security: Foundation and Practice, Springer, Heidelberg (2010) (to appear)
23. Gassend, B.: Physical Random Functions, MSc Thesis, MIT (2003)

24. Majzoobi, M., Elnably, A., Koushanfar, F.: FPGA Time-Bounded Unclonable Authentication. In: Böhme, R., Fong, P.W.L., Safavi-Naini, R. (eds.) IH 2010. LNCS, vol. 6387, pp. 1–16. Springer, Heidelberg (2010)
25. Guajardo, J., Kumar, S.S., Schrijen, G.-J., Tuyls, P.: FPGA Intrinsic PFs and Their Ue For IP Potection. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 63–80. Springer, Heidelberg (2007)
26. Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Devadas, S., Schmidhuber, J.: Modeling Attacks on Physical Unclonable Functions. In: 17th ACM Conference on Computer and Communications Security (2010); Previous versions available from Cryptology ePrint Archive, Report 251/2010
27. Halevi, S., Krawczyk, H.: MMH: Software message authentication in the gbit/Second rates. In: Biham, E. (ed.) FSE 1997. LNCS, vol. 1267, pp. 172–189. Springer, Heidelberg (1997)
28. DeJean, G., Kirovski, D.: RF-DNA: Radio-Frequency Certificates of Authenticity. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 346–363. Springer, Heidelberg (2007)
29. Kariakin, Y.: Authentication of Articles. Patent writing, WO/1997/024699 (1995), `http://www.wipo.int/pctdb/en/wo.jsp?wo=1997024699`
30. Vijaywargi, D., Lewis, D., Kirovski, D.: Optical DNA. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 222–229. Springer, Heidelberg (2009)
31. Hammouri, G., Dana, A., Sunar, B.: CDs Have Fingerprints Too. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 348–362. Springer, Heidelberg (2009)
32. Diffie, W., Hellman, M.E.: New Directions in Cryptography. IEEE Transactions on Information Theory IT-22, 644–654 (1976)
33. Yao, A.C.-C.: Classical physics and the Church-Turing Thesis. Journal of the ACM 50(1), 100–105 (2003)
34. Aaronson, S.: NP-complete Problems and Physical Reality. In: Electronic Colloquium on Computational Complexity (ECCC), 026 (2005)
35. Shor, P.W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM J. Comput. 26(5), 1484–1509 (1997)
36. Csaba, G., Ju, X., Ma, Z., Chen, Q., Porod, W., Schmidhuber, J., Schlichtmann, U., Lugli, P., Rührmair, U.: Application of Mismatched Cellular Nonlinear Networks for Physical Cryptography. In: IEEE CNNA (2010)
37. Lim, D.: Extracting Secret Keys from Integrated Circuits. M.Sc. Thesis, MIT (2004)
38. Suh, G.E., O'Donnell, C.W., Sachdev, I., Devadas, S.: Design and Implementation of the AEGIS Single-Chip Secure Processor Using Physical Random Functions. In: Proc. 32nd ISCA, New York (2005)
39. Yu, M.-D.(Mandel) Devadas, S.: Secure and Robust Error Correction for Physical Unclonable Functions. IEEE Design & Test of Computers 27(1), 48–65 (2010)
40. Rührmair, U., Knobling, R., Weiershäuser, A., Urban, S., Finley, J.: Secure Integrated Optical Physical Unclonable Functions (2010) (in preparation)
41. Lipson, S.G.: Optical Physics, 3rd edn. Cambridge University Press, Cambridge (1995) ISBN 0-5214-3631-1
42. Demtröder, W.: Experimentalphysik 2: Elektrizität und Optik. Springer, Heidelberg (2004) ISBN-10: 3540202102
43. Zhou, D., Mawst, L.J.: Two-dimensional phase-locked antiguided vertical-cavity surface-emitting laser arrays. Applied Physics Letters (2000)
44. Sölter, J.: Personal Communication (2010)

A A First Proof-of-Concept for Optical SIMPLs

In order to rigorously prove the validity of the suggested optical SIMPL implementation, two statements would have to be shown. (i) The system indeed has the desired speed advantage. (ii) Our suggestion is workable in the sense that its responses can be

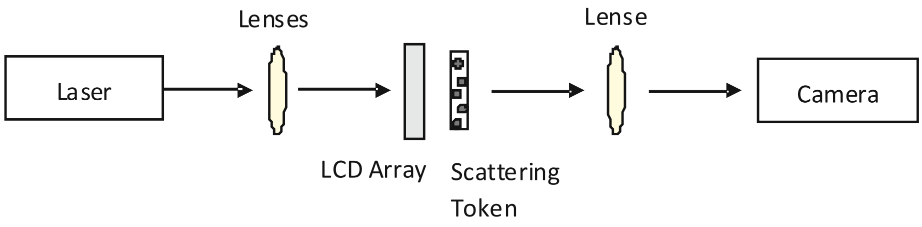


Fig. 2. Schematic illustration of our measurement set-up. The schematic icon "lenses" on the left stands for several lenses that were used to shape the light beam.

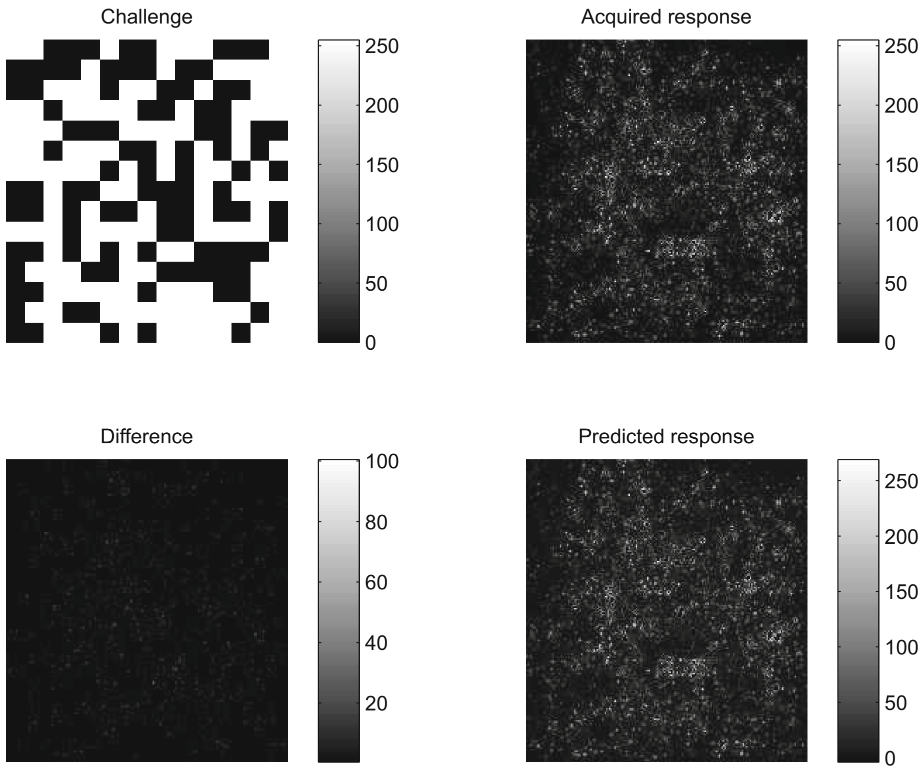


Fig. 3. A randomly chosen 15×15 excitation pattern (top left), a CCD image of the response of the optical SIMPL (top right), the predicted response (bottom right), and the difference map (bottom left)

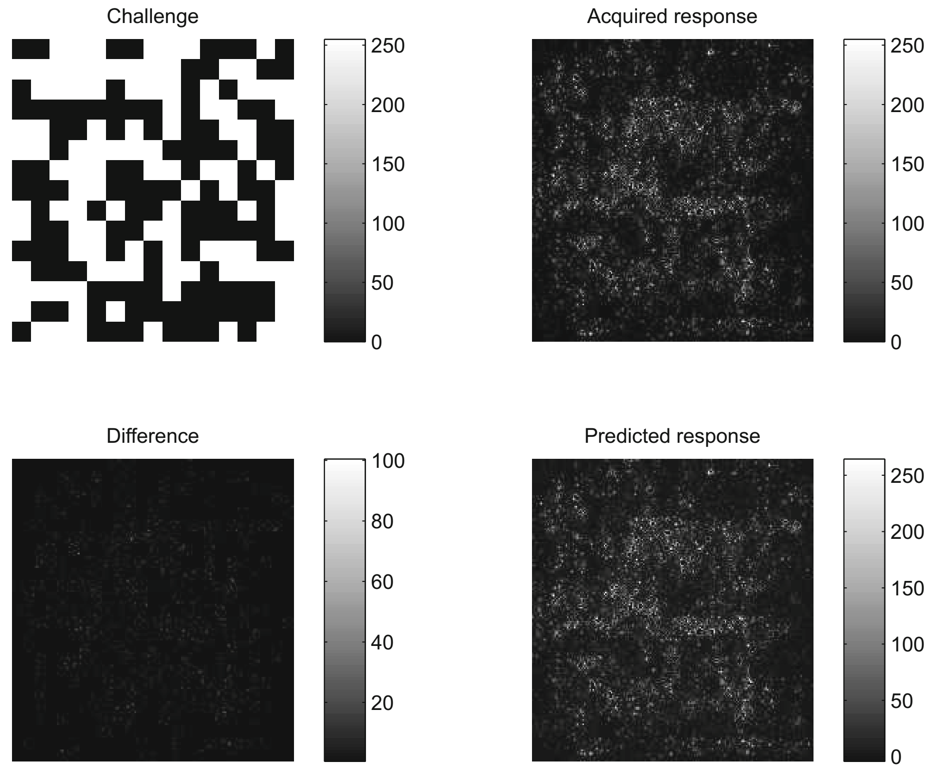


Fig. 4. A second, randomly chosen 15 $\times$ 15 excitation pattern (top left), a CCD image of the response of the optical SIMPL (top right), the predicted response (bottom right), and the difference map (bottom left)

predicted sufficiently accurately by the described approach. Similar to the security of classical cryptoschemes, statement (i) cannot be shown or proven mathematically in a strict sense given the current state of computational complexity theory. Also building an optical SIMPL prototype that operates at optimized operational speed is expensive and beyond the scope of this paper.

Nevertheless, it proved well doable to build a prototype that is not optimized in terms of speed, but which verifies statement (ii). It occured that such a prototype had been already set up in our group in the course of a different study, where we generally investigated the machine learnability of integrated optical PUFs [40]. We found there that it was indeed possible to machine learn the output of linear optical PUFs with high accuracy. This has direct implications for the realizability of optical SIMPL systems; we quote from the work [40] in the sequel.

The set-up we used in [40] is depicted schematically in Figure 2. It consists of a LCD array from an old beamer acquired via ebay for 20 Euros, several lenses (depicted schematically in one symbol) and a scattering token of small glasspheres and a transparent glue ("UHU Schnellfest"). A pattern is switched on in the LCD array, and the laser is directed towards it. The set-up has the same effective operational functionality as a array of phase-locked laser diodes, but was easier to realize with components present in our laboratory.

We applied the method described in Section 4.3 in order to predict the outcome of the speckle pattern. We found that the predictability on the basis of optical wave superposition works not only well in theory, but also in practice. Standard ML regression were applied to 53,700 different CRPs (i.e. patterns on the LCD array and corresponding CCD images) that were collected. The success for two different excitation patterns in shown in Figures 3 and 4. The difference map between the actually acquired optical image and the prediction is contained in the figures, and is very small compared to the natural fluctuations in optical speckle patterns, for example due to laser fluctuations, which were already reported in [8] [9]. The variations observed in the difference map will presumably likely not be noticeable after the usual image transformations have been applied to the output. This illustrates the basic feasibility of predicting the output of optical SIMPLs.

Verification of Timed-Arc Petri Nets

Lasse Jacobsen, Morten Jacobsen, Mikael H. Møller, and Jiří Srba*

Department of Computer Science, Aalborg University,
Selma Lagerlöfs Vej 300, DK-9220 Aalborg East, Denmark

Abstract. Timed-Arc Petri Nets (TAPN) are an extension of the classical P/T nets with continuous time. Tokens in TAPN carry an age and arcs between places and transitions are labelled with time intervals restricting the age of tokens available for transition firing. The TAPN model posses a number of interesting theoretical properties distinguishing them from other time extensions of Petri nets. We shall give an overview of the recent theory developed in the verification of TAPN extended with features like read/transport arcs, timed inhibitor arcs and age invariants. We will examine in detail the boundaries of automatic verification and the connections between TAPN and the model of timed automata. Finally, we will mention the tool TAPAAL that supports modelling, simulation and verification of TAPN and discuss a small case study of alternating bit protocol.

1 Introduction

Formal verification of embedded and hybrid systems is an active research area. Recently, a lot of attention has been devoted to the analysis of systems with quantitative attributes like timing, cost and probability. In particular, several different time-dependent models were developed over the two last decades or so. These models are often introduced as a time extension of some well-studied untimed formalism and include, among others, (networks of) timed automata [7, 8] and different time extensions of the Petri net model [46]. These formalisms are nowadays supported by a number of tools [1, 2, 12, 20, 21, 27, 30, 37] and exploited in model-driven system design methodologies.

We shall focus on the Petri net model extended with continuous time. The timing aspects are associated with different parts of the model in the various time-extended Petri net formalisms. For example, *timed transitions Petri nets* where transitions are annotated with their durations were proposed in [47]. A model in which time parameters are associated with places is called *timed places Petri nets* and it was introduced in [52]. *Time Petri nets* of Merlin and Faber [39, 40] were introduced in 1976 and associate time intervals to each transition. The intervals define the earliest and latest firing time of the transition since it became enabled. Yet another model of *timed-arc Petri nets* was first studied around 1990 by Bolognesi, Lucidi, Trigila and Hanisch [15, 29]. Here time information is

* Partially supported by Ministry of Education of the Czech Republic, project No. MSM 0021622419.

attached to the tokens in the net representing their relative age while arcs from places to transition contain time intervals that restrict the enableness of the transitions. For an overview of the different extensions see e.g. [19, 45, 55, 56].

In this paper we will survey the results and techniques connected with timed-arc Petri nets (TAPN). This model is particularly suitable for modelling of manufacturing systems, work-flow management and similar applications [4, 5, 43, 44, 50, 51] and a recently developed tool TAPAAL [23] enables automatic verification of bounded TAPNs extended with transport/inhibitor arcs and age invariants.

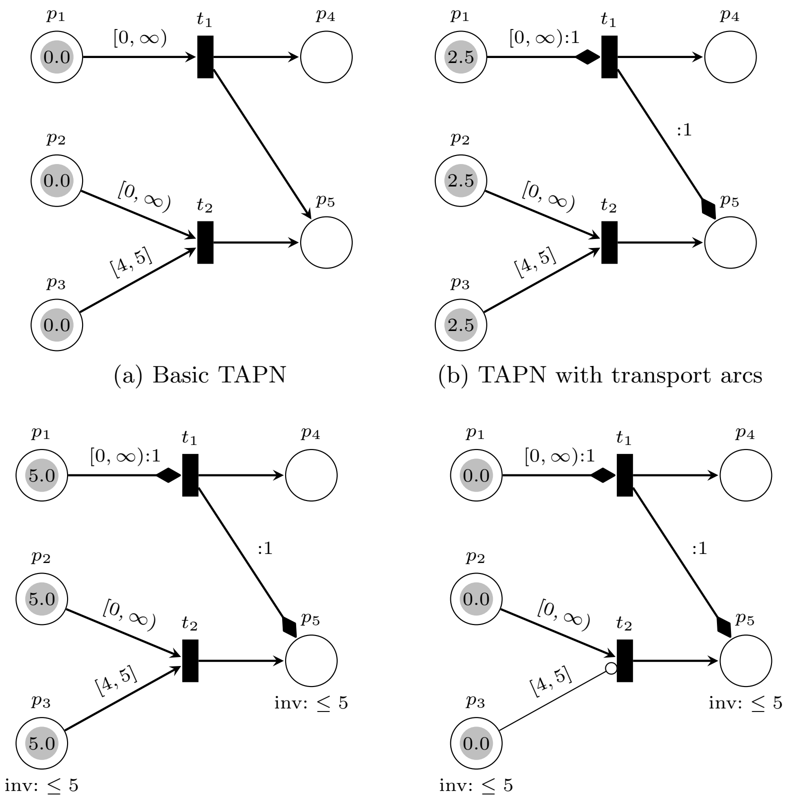
The outline of the paper is as follows. In Section 2 we give an informal introduction to the TAPN model and in Section 3 we describe its formal syntax and semantics. Section 4 illustrates the modeling of alternating bit protocol as a TAPN and Section 5 explains the main decidability and complexity results. Sections 6 and 7 define the TCTL logic and explain a translation from TAPN with transport arcs, inhibitor arcs and age invariants to UPPAAL timed automata. The translation preserves TCTL model checking including liveness properties. Finally, Section 8 gives a short conclusion.

2 Informal Introduction to Timed-Arc Petri Nets

We shall first informally introduce the TAPN model extended with transport arcs, age invariants and inhibitor arcs. A basic timed-arc Petri net is presented in Figure 1a. It consists of two transitions t_1 and t_2 drawn as rectangels and five places $p_1, \ldots, p_5$ drawn as circles. There is one token of age 0.0 in each of the places p_1, p_2 and p_3. Initially, only the transition t_1 is enabled because its input place p_1 contains a token of an age that fits into the interval $[0, \infty]$ present on the arc from p_1 to t_1. The transition t_2 requires a token of any age in p_2 but also a token of an age in the interval $[4, 5]$ in p_3. This is why t_2 is not enabled at the moment. Because t_1 is enabled, it can fire, whereby it removes the token from p_1 and produces a new fresh token of age 0.0 in each of the output places p_4 and p_5. Instead, it is also possible that the net performs a time delay of, say, 4.5 time units. By this step all tokens in the net grow 4.5 time units older. As all tokens are now of age 4.5, both t_1 and t_2 are enabled and can fire. Notice that the tokens that are produced even after the time delay are of age 0.0.

Let us now introduce *transport arcs* into our example net. Specifically, we will replace the normal arcs from p_1 to t_1 and from t_1 to p_5 with a pair of transport arcs with solid arrow tips as illustrated in Figure 1b. Transport arcs come always in pairs like this. Note that the symbol ':1' on the transport arcs is there to denote the pairing of the arcs (as it is in general possible to have more than one pair of transport arcs connected to a transition).

For the sake of illustration, assume that we have initially made a time delay of 2.5 time units such that all tokens are now of age 2.5. Transition t_1 is still the only enabled one in the net at this point, however, there is a difference when we fire t_1. Firing t_1 will remove a token of age 2.5 from p_1 and produce a token of age 0.0 in the place p_4 as before. However, due to the transport arcs, it will



(a) Basic TAPN (b) TAPN with transport arcs

(c) TAPN with age invariants (d) TAPN with inhibitor arc

Fig. 1. Examples of timed-arc Petri nets

produce a token at p_5 of the same age as the token it removed from p_1, i.e. of the age 2.5 in this case. Hence transport arcs allow us to preserve the ages of tokens as they are transported through the net. This is a feature particularly suitable for modelling of e.g. product lines where we need to track the age of a product from its start until its final production stage.

The next modelling feature is called *age invariant.* It simply restricts the maximum age of tokens that can appear in certain places. In our running example, we can add age invariants to the places p_3 and p_5 as illustrated in Figure 1c. These invariants will disallow tokens older than 5 time units in these two places.

Assume a situation after a time delay such that all tokens are of age 5.0 as in the figure. At this point no further time delays are possible as they would violate the invariant at p_3. We are thus forced to fire either t_1 or t_2, both of which are enabled. Invariants hence facilitate the modelling of urgency. One of the particularities of the combination of invariants and transport arcs is that transitions are disabled should their transport arcs move a token to a place where the age of the token violates the age invariant. In our example t_1 is enabled but should the invariant at p_5 allow tokens only of age at most 4, then it would be disabled.

Finally, we introduce *inhibitor arcs* that disable the firing of a transition based on the presence of tokens in certain places. In our example we will replace the arc from p_3 to t_2 with an inhibitor arc with circle arrow tip as illustrated in Figure 1d. Transition t_2 is then blocked whenever there is a token in place p_3 with age in the interval $[4, 5]$. As this is not the case in the depicted situation, both t_1 and t_2 are enabled and can be fired. Firing of t_2 has no effect on the tokens in the place p_3. If instead a time delay of 4 time units was performed, t_2 would be blocked and only t_1 could fire. This concludes the informal introduction to timed-arc Petri nets.

3 Formal Definition of Timed-Arc Petri Nets

We start with the preliminaries and the definition of timed transition system.

We let $\mathbb{N}_0$ and $\mathbb{R}_{\geq 0}$ denote the sets of nonnegative integers and nonnegative real numbers, respectively.

A *timed transition system* (TTS) is a pair $T = (S, \longrightarrow)$ where S is a set of states (or processes) and $\longrightarrow \subseteq S \times S \cup S \times \mathbb{R}_{\geq 0} \times S$ is a transition relation.

We write $s \longrightarrow s'$ whenever $(s, s') \in \longrightarrow$ and call them *discrete transitions*, and $s \xrightarrow{d} s'$ whenever $(s, d, s') \in \longrightarrow$ and call them *delay transitions*. We require that all TTS we consider satisfy the following standard axioms for delay transitions (see e.g. [13]). For all $d, d' \in \mathbb{R}_{\geq 0}$ and $s, s', s'' \in S$:

1. **Time Additivity:** if $s \xrightarrow{d} s'$ and $s' \xrightarrow{d'} s''$ then $s \xrightarrow{d+d'} s''$,
2. **Time Continuity:** if $s \xrightarrow{d+d'} s''$ then $s \xrightarrow{d} s' \xrightarrow{d'} s''$ for some s',
3. **Zero Delay:** $s \xrightarrow{0} s$ for each state s, and
4. **Time Determinism:** if $s \xrightarrow{d} s'$ and $s \xrightarrow{d} s''$ then $s' = s''$.

By $s[d]$ we denote the state s' (if it exists) such that $s \xrightarrow{d} s'$. Time determinism ensures the uniqueness of $s[d]$. We write $s \Longrightarrow s'$ if $s \longrightarrow s'$ or $\xrightarrow{d} s'$ for some d. The notation $\Longrightarrow^*$ denotes the reflexive and transitive closure of $\Longrightarrow$.

3.1 Syntax

We shall now define the timed-arc Petri net model. First, we define the set of well-formed time intervals by the abstract syntax where $a, b \in \mathbb{N}_0$ and $a < b$:

$$I ::= [a, a] \mid [a, b] \mid [a, b) \mid (a, b] \mid (a, b) \mid [a, \infty) \mid (a, \infty).$$

We denote the set of all well-formed time intervals by $\mathcal{I}$. Further, the set of all well-formed time intervals for invariants is denoted by $\mathcal{I}^{inv}$ and defined according to the following abstract syntax:

$$I_{\text{Inv}} ::= [0, 0] \mid [0, b] \mid [0, b) \mid [0, \infty).$$

The predicate $r \in I$ is defined for $r \in \mathbb{R}_{\geq 0}$ in the expected way.

Definition 1. *A* TAPN *is a 7-tuple* $(P, T, IA, OA, Transport, Inhib, Inv)$*, where*

- P *is a finite set of* places*,*
- T *is a finite set of* transitions *s.t.* $P \cap T = \emptyset$*,*
- $IA \subseteq P \times \mathcal{I} \times T$ *is a finite set of* input arcs *s.t.*

$$((p, I, t) \in IA \wedge (p, I', t) \in IA) \Rightarrow I = I'$$

- $OA \subseteq T \times P$ *is a finite set of* output arcs*,*
- $Transport : IA \times OA \rightarrow \{true, false\}$ *is a function defining* transport arcs *which are pairs of input and output arcs connected to some transition, formally we require that for all* $(p, I, t) \in IA$ *and* $(t', p') \in OA$ *whenever* $Transport((p, I, t),(t', p'))$ *then* $t = t'$ *and moreover for all* $\alpha \in IA$ *and all* $\beta \in OA$

$$(Transport(\alpha, (t', p')) \Rightarrow \ \alpha = (p, I, t)) \wedge$$

$$(Transport((p, I, t), \beta) \Rightarrow \ \beta = (t', p'))$$

- $Inhib : IA \longrightarrow \{true, false\}$ *is a function defining* inhibitor arcs *which do not collide with transport arcs, i.e. whenever* $Transport(\alpha, \beta)$ *for some* $\alpha \in IA$ *and* $\beta \in OA$ *then* $\neg Inhib(\alpha)$*, and*
- $Inv : P \rightarrow \mathcal{I}^{inv}$ *is a function assigning* age invariants *to places.*

A TAPN is called basic *if the functions Transport and Inhib return false for all arcs.*

The preset of a transition $t \in T$ is defined as ${}^\bullet t = \{p \in P \mid (p, I, t) \in IA\}$. Similarly, the postset of t is defined as $t^\bullet = \{p \in P \mid (t, p) \in OA\}$. For technical convenience we do not allow multiple arcs.

3.2 Semantics

We will now define the semantics of the TAPN. First we define a marking, which is a function assigning to each place a finite multiset of nonnegative real numbers (all such finite multisets are denoted by $\mathcal{B}(\mathbb{R}_{\geq 0})$). The real numbers represent the age of tokens that are currently at a given place; the age of every token must moreover respect the age invariant of the place where the token is located.

Definition 2 (Marking). *Let* $N = (P, T, IA, OA, Transport, Inhib, Inv)$ *be a TAPN. A* marking M *on* N *is a function* $M : P \longrightarrow \mathcal{B}(\mathbb{R}_{\geq 0})$ *where for every place* $p \in P$ *and every token* $x \in M(p)$ *we have* $x \in Inv(p)$*. The set of all markings over* N *is denoted by* $\mathcal{M}(N)$*.*

We shall sometimes use the notation (p, x) to refer to a token in the place p of age $x \in \mathbb{R}_{\geq 0}$. Likewise, we shall sometimes write $M = \{(p_1, x_1), (p_2, x_2), \ldots, (p_n, x_n)\}$ for a multiset representing a marking M with n tokens located in the places p_i and with age x_i for $1 \leq i \leq n$.

A *marked* TAPN is a pair (N, M_0) where N is a TAPN and M_0 is an initial marking on N where all tokens have the age 0.

Definition 3 (Enabledness). *Let $N = (P, T, IA, OA, Transport, Inhib, Inv)$ be a TAPN. We say that a transition $t \in T$ is* enabled *in a marking M by tokens $In = \{(p, x_p) \mid p \in {}^\bullet t\} \subseteq M$ and $Out = \{(p', x_{p'}) \mid p' \in t^\bullet\}$ if*

- *for all input arcs except the inhibitor arcs there is a token in the input place with an age satisfying the age guard of the arc, i.e.*

$$\forall (p, I, t) \in IA \; . \; \neg Inhib((p, I, t)) \Rightarrow x_p \in I$$

- *for all inhibitor arcs there is no token in the input place of the arc with an age satisfying the age guard of the arcs respectively, i.e.*

$$\forall (p, I, t) \in IA \; . \; Inhib((p, I, t)) \Rightarrow \neg \exists x \in M(p) \; . \; x \in I$$

- *for all input arcs and output arcs which constitute a transport arc the age of the input token must be equal to the age of the output token and satisfy the invariant of the output place, i.e.*

$$\forall (p, I, t) \in IA \; . \; \forall (t, p') \in OA \; . \; Transport((p, I, t), (t, p')) \Rightarrow \\ (x_p = x_{p'}) \;\wedge\; (x_p \in Inv(p'))$$

- *for all output arcs that are not part of a transport arc the age of the output token is 0, i.e.*

$$\forall (t, p') \in OA \; . \big(\neg(\exists \alpha \in IA . Transport(\alpha, (t, p'))) \;\Rightarrow x_{p'} = 0\big) \; .$$

Definition 4 (Firing Rule). *Let $N = (P, T, IA, OA, Transport, Inhib, Inv)$ be a TAPN, M a marking on N and $t \in T$ a transition . If t is enabled in the marking M by tokens In and Out then it can* fire *and produce a marking M' defined as*

$$M' = (M \setminus In) \cup Out$$

where $\setminus$ and $\cup$ are operations on multisets.

Definition 5 (Time Delay). *Let $N = (P, T, IA, OA, Transport, Inhib, Inv)$ be a TAPN and M a marking on N. A* time delay *$d \in \mathbb{R}_{\geq 0}$ is allowed in M if $(x + d) \in Inv(p)$ for all $p \in P$ and all $x \in M(p)$, i.e. by delaying d time units no token violates any of the age invariants. By delaying d time units in M we reach a marking M' defined as*

$$M'(p) = \{x + d \mid x \in M(p)\}$$

for all $p \in P$.

A given TAPN N now defines a timed transition system $(\mathcal{M}(N), \longrightarrow)$ where states are markings of N and for two markings M and M' we have $M \longrightarrow M'$ if by firing some transition in M we can reach the marking M' and $M \xrightarrow{d} M'$ if by delaying d time units in M we reach the marking M'. We say that a marking M' is reachable from marking M if $M \Longrightarrow^* M'$.

4 A Small Case Study: Alternating Bit Protocol

In this section we shall discuss a small case study of the well-known Alternating Bit Protocol (ABP) [9] and show its modelling by timed-arc Petri nets. The purpose of the protocol is to ensure a safe communication between a sender and a receiver over an unreliable medium. To achieve this, messages are labelled with a control bit in order to compensate (via message retransmission) for the possibility of losing messages in transfer. In order to avoid a confusion between new and old messages, each message is moreover time-stamped and after its expiration it is ignored.

Figure 2 shows a TAPN model of the protocol. The model contains four places associated to the sender (on the left side) and four places associated to the receiver (on the right side). The sender and receiver can communicate over a lossy medium represented by the four places in the middle.

Initially, the only enabled transition is Send0 which moves the sender from the place Sender0A to Sender0B and at the same time places a message (token) with the appended bit 0 into the place Medium0A. At any time the message can be lost by firing the transition Loss0A. Now the receiver can read the message and move it to the place Receiver0B by firing the transition Receive0, followed by firing AckSend0 and placing a token (acknowledgment) in the place Medium0B. There is at least a one time unit delay before the acknowledgment is sent. The acknowledgment can now be read by the sender using the transition AckRec0. Notice that the path of the token from place Medium0A to Medium0B consists of transport arcs and hence the time-stamp of the message is preserved. The sender accepts only acknowledgments that are no older than three time units since the message was sent. As the medium is lossy, the communication may fail and it may be necessary to retransmit the message by firing the transition ReSend0, which must happen any time between five to six time units since the last time the message was sent. Similarly, the receiver may retransmit the acknowledgment by firing the transitions ReceiveOld0 and AckSend0. If the first phase with the appended bit 0 succeeded, the protocol continues in a symmetric way with the next message that gets appended the bit 1.

Having the formal model of alternating bit protocol in place, we can now start analysing its behaviour. One possible analysis technique is the simulation of transition firings that can reveal possible flaws in the design, however, it cannot be used to argue about the correctness of the protocol. We will postpone the actual definition of the correctness requirement of the protocol to Section 6 once an appropriate logic for the formulation of this property is defined. In the meantime we can observe that the net can exhibit a behaviour in which the places representing the medium become unbounded (there is no a priori given constant that bounds the number of tokens in these places). This can be seen by the fact that e.g. the transition ReSend0 can be repeatedly fired and, as the net is not forced to ever perform the transition Loss0A, more and more tokens will accumulate in the place Medium0A. In the section to follow, we will show that automatic verification of unbounded nets with invariants is not possible, as the model has the full Turing power.

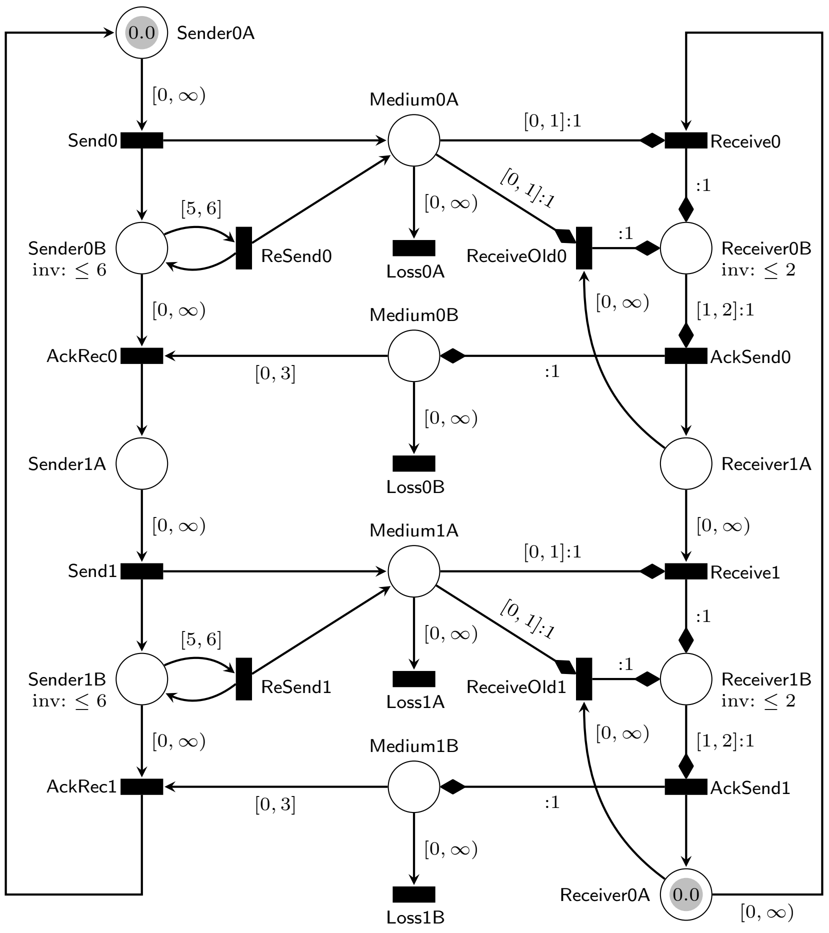


Fig. 2. A TAPN model of the alternating bit protocol

A possible solution to this problem (which is though specific to our concrete model) is to introduce age invariants to all places representing the medium which will disallow tokens older than two time units. This will enforce urgency on the transitions that lose messages and it can be proved (or automatically verified) that the net becomes bounded after such an addition, while the interesting behaviour of the protocol does not change.

Another approach that works in general is to consider an under-approximation of the net behaviour where we give a limit on the maximum number of new tokens that the net can produce and explore the behaviour of the net only up to that many tokens. An experiment using this approach is described in Section 7.

A similarly looking model of the alternating bit protocol was given also for time Petri nets (see e.g. [11]) where clocks are associated to each transition of the net. Unlike our TAPN model, TPN do not allow time-stamps on tokens and messages are instead automatically discarded after one time unit. Hence the behaviour of the TPN is less general and does not allow us to model (at least not in a straightforward way) all the features available in TAPN.

5 Overview of (Un)Decidability and Complexity Results

In this section we shall discuss results about (un)decidability and complexity questions of the classical Petri net problems like reachability, coverability and boundedness in the TAPN context.

We start with the problem of *reachability*: given a marked net (N, M_0) and a marking M, is M reachable from M_0? In spite of the fact that reachability is decidable for untimed Petri nets [38], it is undecidable for timed-arc Petri nets [48], even for nets without any transport/inhibitor arcs and age invariants. The result can be further extended to the case where tokens in different places are not required to age synchronously [42].

We shall now recall the idea of the undecidability result by Ruiz et al. [48] as it is easy to explain and illustrates the power of tokens with age. For showing the undecidability of many Petri net problems the halting problem for Minsky two counter machine is often exploited. The proof from [48] is no exception.

A *Minsky machine* with two nonnegative counters c_1 and c_2 is a sequence of labelled instructions

$$1 : \mathsf{inst}_1;\ 2 : \mathsf{inst}_2;\ \ldots, n : \mathsf{inst}_n$$

where $\mathsf{inst}_n = \mathsf{HALT}$ and each inst_i, $1 \leq i < n$, is of one of the following forms

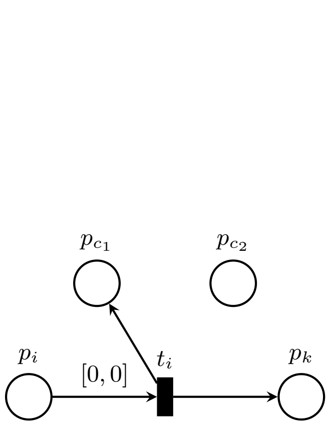
– (Inc) i: c_j`++; goto` k
– (Dec) i: `if` c_j`=0 then goto` k `else (`c_j`--; goto` ℓ`)`

for $j \in \{1, 2\}$ and $1 \leq k, \ell \leq n$.

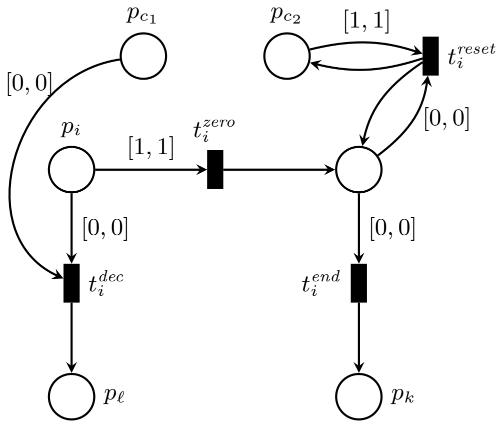
Instructions of type (Inc) are called *increment* instructions and of type (Dec) are called *test and decrement* instructions. A configuration is a triple (i, v_1, v_2) where i is the current instruction and v_1 and v_2 are the values of the counters c_1 and c_2, respectively. A computation step between configurations is defined in the natural way. If starting from the initial configuration $(1, 0, 0)$ the machine reaches the instruction HALT then we say it *halts*, otherwise it *loops*. It is well known that the problem whether a given Minsky machine halts is undecidable [41]. This is the case even for the question whether it halts with both counters empty (as they can be easily emptied before the halting instruction is reached).

The main idea of simulating a Minsky machine by a Petri net is to create two places called p_{c_1} and p_{c_2} such that the number of tokens in these places represents the value of the counters c_1 and c_2, respectively. Also, for every instruction label i, $1 \leq i \leq n$, we create a new place called p_i in the net. During the behaviour of the net the sum of the tokens in $p_1, \ldots, p_n$ will be invariantly equal to one. The presence of a token at p_i represents the fact that the next instruction to be executed is the one with label i.

Given a Minsky machine with two counters, we shall now construct a basic TAPN such that, given an initial marking with just one token in p_1, the final



(a) i: `c`$_1$`++; goto` k



(b) i: `if` c_1`=0 then goto` k `else (`c_1`--; goto` ℓ`)`

Fig. 3. Simulation of (Inc) and (Dec) instructions by basic TAPN

marking where there is exactly one token of age 0 in place p_n is reachable if and only if the given Minsky machine halts.

The instruction of type (Inc) is easy to simulate as depicted in Figure 3a for the increment of c_1; a symmetric construction is used for the increment of c_2. The interval $[0,0]$ disallows any time delay before the transition t_i is fired. After the firing, the control is given to the instruction k and the counter c_1 is incremented by one. Note that there is no invariant in p_i so we are also allowed to delay here but in this case the whole net will get stuck and it will thus not be possible to place a token at the place p_n.

For any instruction of type (Dec) we add the places and transitions as depicted in Figure 3b (again the test on counter c_2 is completely symmetric). The modelling of the decrement branch via the transition t_i^{dec} is straightforward. The difficult part is the simulation of the jump to label k when the counter c_1 is empty. As Petri nets, unless equipped with inhibitor arcs, do not allow to test for zero number of tokens in a place, we need to introduce a few more transitions that will detect a *cheating*, i.e. when the transition t_i^{zero} is taken while there are some tokens in p_{c_1}. Notice that the transition t_i^{zero} can be fired only after a delay of one time unit, hence all tokens in both p_{c_1} and p_{c_2} will also grow older by one time unit. Now the transition t_i^{reset} will allow to reset the ages of all tokens in p_{c_2} to 0, however, any potential tokens in place p_{c_1} will remain of age 1 (if we cheated). The simulation then continues by firing the transition t_i^{end} which gives the control to the instruction k.

Clearly, if the given Minsky machine halts with both counters empty, we can faithfully simulate its computation in the Petri net such that the place p_n will be eventually marked and all other places will be empty.

On the other hand, if the Minsky machine loops then we can either faithfully simulate it in the net but then the final marking with one token in p_n will never be reached, or we can cheat but as a result the net will contain tokens which are too old (also called dead tokens) in either p_{c_1} or p_{c_2} that cannot be removed, and hence the final marking will not be reachable either.

Theorem 1 ([48]). *Reachability is undecidable for the basic timed-arc Petri net model (with only ordinary arcs and no age invariants).*

On the other hand, coverability, boundedness and other problems remain decidable for the basic TAPN [4, 6, 49] model, which is also known to offer 'weak' expressiveness, in the sense that basic TAPN cannot simulate Turing machines [14].

The *coverability problem* asks, given an initial marking M_0 and a final marking M, is there a marking M' reachable from M_0 such that $M(p) \subseteq M'(p)$ for all places p?

The *boundedness problem* asks, given an initial marking M_0, is there a constant k such that the total number of tokens in any reachable marking from M_0 is less than or equal to k? If this is the case, the net is called k-bounded.

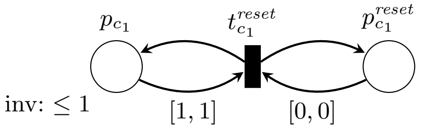
It is known that coverability remains decidable for the basic TAPN extended with read arcs [17] where a read arc is a special case of a pair of two transport arcs that return the consumed token to the same place (and hence do not change its age). These results hold due to the monotonicity property (adding more tokens to the net does not restrict the possible executions) and the application of well-quasi-ordering (for a general introduction see [26]) resp. better-quasi-ordering [3] techniques.

One of the major weaknesses of the basic TAPN is the lack of the possibility to model urgent behaviour. On the other hand, when allowing age invariants, both coverability and boundedness become undecidable as shown in [31] and demonstrated in what follows.

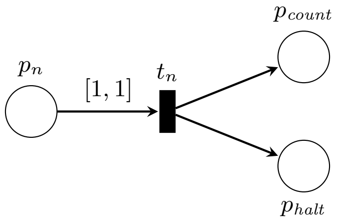
The basic idea is similar as in the previous reduction and illustrations are depicted in Figure 4. We have two places p_{c_1} and p_{c_2} representing the counters plus we add one more place called p_{count} which records the number of already executed instructions and will be used for the undecidability of boundedness. The counters are each equipped with a place called $p_{c_j}^{reset}$ such that a presence of a token in this place will allow us to reset the age of all tokens of age 1 in p_{c_j}.

The simulation of the increment instruction in Figure 4c starts by delaying one time unit. Now all tokens in the counters become of age 1 but can subsequently be reset to 0 due to the presence of the tokens in $p_{c_1}^{reset}$ and $p_{c_2}^{reset}$. By performing t_i^{goto} the simulating finishes, increases the number of tokens in p_{count} by one, gives the control to the instruction k and adds 1 to the counter c_1.

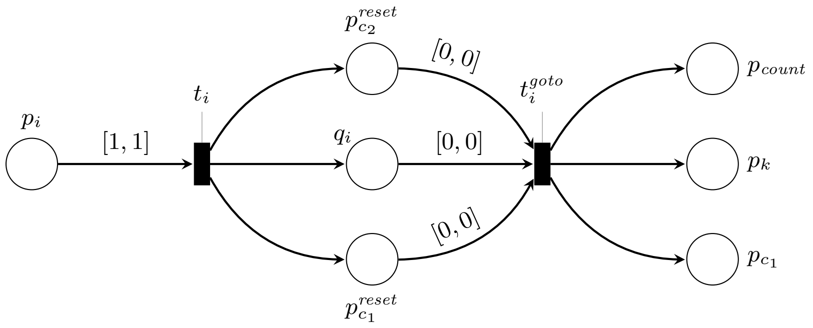
The decrement instruction, depicted in Figure 4d, can fire the transition t_i^{dec}, provided that there is a token of age 0 in p_{c_1}, increase the number of counted steps and give the control to the instruction ℓ. It can also delay one time unit and perform the transition t_i^{zero} which will allow us to reset the ages of tokens in p_{c_2} and after firing t_i^{end} add one token to p_{count} and continue with the simulation of the instruction k. The point is that if we were cheating in the simulation and fired the transition t_i^{zero} with a nonempty counter c_1, tokens of age 1 will necessarily appear in the place p_{c_1}. Notice that the simulation of most instructions (in particular of the halt instruction) must start with a time delay, however, if in



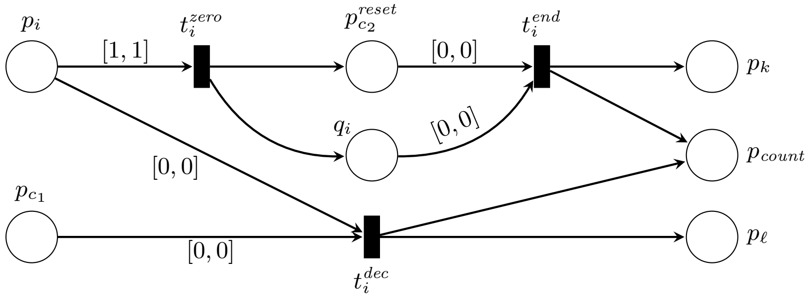
(a) Simulation of the counter c_1



(b) Simulation of instruction halt



(c) i: c_1`++; goto` k



(d) i: `if` c_1`=0 then goto` k `else (`c_1`--; goto` ℓ`)`.

Fig. 4. Simulation of a Minsky machine by a TAPN with invariants

some of the counters there were tokens of age 1, no time delay is possible due to the age invariants ≤ 1 in p_{c_1} and p_{c_2}. Hence the place p_{halt} can be marked if and only if the net did not cheat and this gives the undecidability of coverability.

The same construction also serves as a reduction showing undecidability of boundedness. Assume that the given Minsky machine halts in k steps. This means that if the net faithfully simulates its behaviour, it will terminate with a token in p_{halt} and at most k tokens in any of the two counter places and p_{count}. If the net cheated at some point, most of the instructions will be disabled as discussed above, except for the firing of t_i^{dec}, which can however only decrease the number of tokens in the places. The net is hence bounded. On the other hand, if the Minsky machine loops, the net can faithfully simulate this infinite

Table 1. Overview of (un)decidability results for TAPN

	Reachability	**Coverability**	**Boundedness**
basic TAPN	× [48]	✓ [4]	✓ [6]
basic TAPN plus transport arcs	× [48]	$\checkmark^?$	$\checkmark^?$
basic TAPN plus age invariants	× [48]	× [31]	× [31]
basic TAPN plus inhibitor arcs	× [28]	× [28]	× [28]

behaviour and the place p_{count} will be unbounded. Hence the undecidability of boundedness for basic TAPN with invariants is established too.

Theorem 2 ([31]). *Coverability and boundedness are undecidable for basic TAPN with invariants.*

A summary of the results is provided in Table 1. The decidability of coverability and boundedness for TAPN with transport arcs is marked with a question mark as it is only a claim, though the proofs from [17] for read arcs seem easy to extend to TAPN with transport arcs too.

In applications, the fact that coverability is decidable for the basic TAPN model can be useful as demonstrated in [4] where the authors verified a parameterized version of Fischer's protocol [36] using their prototype implementation of the coverability algorithm.

Most often though, we may like to use the additional features like age invariants and inhibitor arcs to facilitate the modelling process. While all interesting problems become quickly undecidable for such models, we may still verify a number of interesting properties by restricting ourselves to bounded nets where the maximum number of tokens in the net is given as a constant. Recent work shows that bounded TAPN and 1-safe (at most one token in any place) nets offer a similar expressive power as networks of timed automata, even though the models are rather different. Sifakis and Yovine [53] provided a translation of 1-safe timed-arc Petri nets into timed automata which preserves strong timed bisimilarity but their translation causes an exponential blow up in the size. Srba established in [54] a strong relationship (up to isomorphism of timed transition systems) between networks of timed automata and a superclass of 1-safe TAPN extended with read arcs. For reachability questions the reductions in [54] work in polynomial time. Recently Bouyer et al. [17] presented a reduction from bounded TAPN (with read-arcs) to 1-safe TAPN (with read-arcs), which preserves timed language equivalence. Hence PSPACE-completeness of reachability on 1-safe and bounded TAPN was established [17, 54].

Nevertheless the translations described in these papers are inefficient from the practical point of view as they either cause an exponential blow-up in the size or create a new parallel component with a fresh local clock for *each place* in the net. In connection with the development of the tool TAPAAL [1] for modelling, simulation and verification of extended timed-arc Petri nets, more efficient translations were investigated [22, 23].

Recently, in [33] we identified a general class of translations that preserve Timed Computation Tree Logic (TCTL), a logic suitable for practical specification of many useful temporal properties (see e.g. [45]). In the next two sections we shall present the framework and give an example of an efficient translation from TAPN to UPPAAL networks of timed automata [2].

6 Timed Computation Tree Logic

In this section we introduce the Timed Computation Tree Logic (TCTL). Unlike much work on TCTL where only infinite alternating runs are considered [45] or the details are simply not discussed [16, 24], we consider also finite maximal runs that appear in the presence of stuck computations or time invariants (strict or nonstrict) and treat the semantics in its full generality as used in most of the verification tools nowadays. This fact is particularly important for the verification of liveness properties.

Before we define the syntax and semantics of TCTL, we extend the notion of timed transition systems (TTS) as defined in Section 3 with propositions. A *timed transition system with propositions* is a quadruple $T = (S, \longrightarrow, \mathcal{AP}, \mu)$ where $(S, \longrightarrow)$ is a TTS, $\mathcal{AP}$ is a set of atomic propositions, and $\mu : S \to 2^{\mathcal{AP}}$ is a function assigning sets of true atomic propositions to states.

For a TAPN $N = (P, T, \mathit{IA}, \mathit{OA}, \mathit{Transport}, \mathit{Inhib}, \mathit{Inv})$ the set of atomic propositions $\mathcal{AP}$ and the labeling function μ can be defined as

$$\mathcal{AP} \overset{def}{=} \{(p \bowtie n) \mid p \in P, n \in \mathbb{N}_0 \text{ and } \bowtie \in \{<, \leq, =, \geq, >\}\}$$

and for a marking M we have

$$\mu(M) \overset{def}{=} \{(p \bowtie n) \mid |M(p)| \bowtie n \text{ and } \bowtie \in \{<, \leq, =, \geq, >\}\} .$$

The intuition is that a proposition $(p \bowtie n)$ is true in a marking M iff the number of tokens in the place p satisfies the given relation with respect to n.

A *run* $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} \ldots$ in a TTS is a (finite or infinite) alternating sequence of time delays and discrete actions.

We shall now introduce the syntax and semantics of TCTL. The presentation is inspired by [45]. Let $\mathcal{AP}$ be a set of atomic propositions. The set of TCTL formulae $\Phi(\mathcal{AP})$ over $\mathcal{AP}$ is given by

$$\varphi ::= \wp \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid E(\varphi_1 \, U_I \, \varphi_2) \mid A(\varphi_1 \, U_I \, \varphi_2) \mid E(\varphi_1 \, R_I \, \varphi_2) \mid A(\varphi_1 \, R_I \, \varphi_2)$$

where $\wp \in \mathcal{AP}$ ranges over atomic propositions and $I \in \mathcal{I}$ ranges over time intervals. Formulae without any occurrence of the operators $A(\varphi_1 \, U_I \, \varphi_2)$ and $E(\varphi_1 \, R_I \, \varphi_2)$ form the *safety fragment* of TCTL.

The intuition of the until and release TCTL operators (formalized later on) is as follows:

- $A(\varphi_1 \, U_I \, \varphi_2)$ is true if on all maximal runs φ_2 eventually holds within the interval I, and until it does, φ_1 continuously holds;

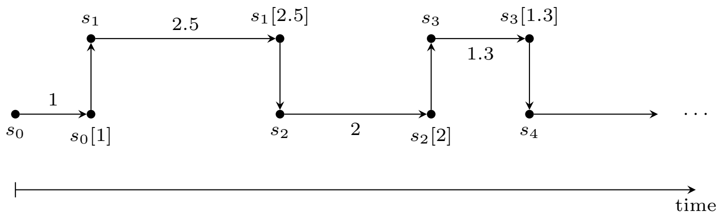


Fig. 5. Illustration of a concrete run

- $E(\varphi_1\, U_I\, \varphi_2)$ is true if there exists a maximal run such that φ_2 eventually holds within the interval I, and until it does, φ_1 continuously holds;
- $A(\varphi_1\, R_I\, \varphi_2)$ is true if on all maximal runs either φ_2 always holds within the interval I or φ_1 occurred previously;
- $E(\varphi_1\, R_I\, \varphi_2)$ is true if there exists a maximal run such that either φ_2 always holds within the interval I or φ_1 occurred previously.

In the semantics, we handle maximal runs in their full generality. Hence we have to consider all possibilities in which a run can be "stuck". In this case, we annotate the last transition of such a run with one of the three special ending symbols (denoted δ in the definition below).

A *maximal run* ρ is either

(i) an infinite alternating sequence of the form $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} s_2[d_2] \longrightarrow \ldots$, or

(ii) a finite alternating sequence of the form $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow \ldots \longrightarrow s_n \xrightarrow{\delta}$ where $\delta \in \{\infty, d_n^{\leq}, d_n^{<}\}$ for some $d_n \in \mathbb{R}_{\geq 0}$ s.t.
   - if $\delta = \infty$ then $s_n \xrightarrow{d} s_n[d]$ for all $d \in \mathbb{R}_{\geq 0}$,
   - if $\delta = d_n^{\leq}$ then $s_n \not\xrightarrow{d}$ for all $d > d_n$ and $s_n \xrightarrow{d_n} s_n[d_n]$ s.t. $s_n[d_n] \not\longrightarrow$, and
   - if $\delta = d_n^{<}$ then $s_n \not\xrightarrow{d}$ for all $d \geq d_n$, and there exists d_s, $0 \leq d_s < d_n$, such that for all d, $d_s \leq d < d_n$, we have $s_n \xrightarrow{d} s_n[d]$ and $s_n[d] \not\longrightarrow$.

By *MaxRuns*(T, s) we denote the set of maximal runs in a TTS T starting at s.

Intuitively, the three conditions in case (ii) describe all possible ways in which a finite run can terminate. First, a run can end in a state where time diverges. The other two cases define a run which ends in a state from which no discrete transition is allowed after some time delay, but time cannot diverge either (typically caused by the presence of invariants in the model). These cases differ in whether the bound on the maximal time delay can be reached or not.

Figure 5 illustrates a part of a maximal run $\rho = s_0 \xrightarrow{1} s_0[1] \longrightarrow s_1 \xrightarrow{2.5} s_1[2.5] \longrightarrow s_2 \xrightarrow{2} s_2[2] \longrightarrow s_3 \xrightarrow{1.3} s_3[1.3] \longrightarrow s_4 \longrightarrow \ldots$. Note that actions take zero time units and that, although not shown in this example, time delays can be zero so it is possible to do multiple actions in succession without any time

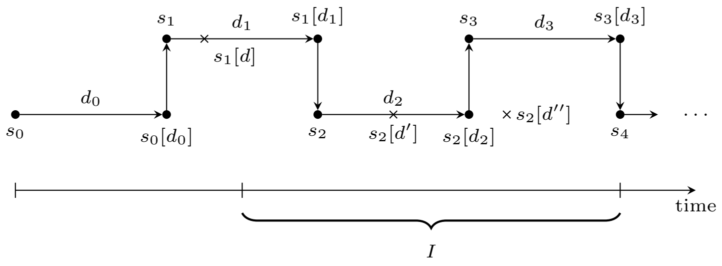


Fig. 6. Illustration of a run

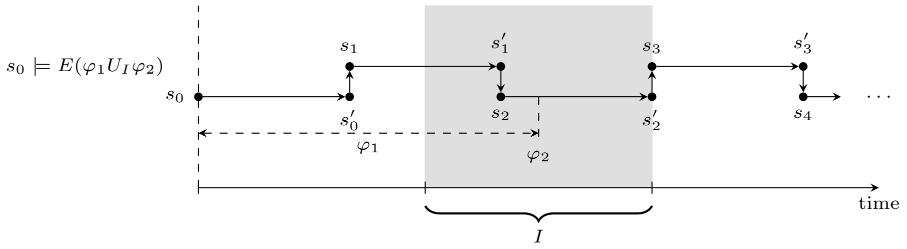


Fig. 7. Illustration of a run satisfying an until formula

progression in between. Further, there is no special meaning as to whether the arrow for an action goes up or down, this is simply to keep the figure small.

Let us now introduce some notation for a given maximal run $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} \ldots$. First, $r(i,d)$ denotes the total time elapsed from the beginning of the run up to some delay $d \in \mathbb{R}_{\geq 0}$ after the i'th discrete transition. Formally, $r(i,d) = \left(\sum_{j=0}^{i-1} d_j\right) + d$. Second, we define a predicate $valid_\rho : \mathbb{N}_0 \times \mathbb{R}_{\geq 0} \times \mathcal{I} \to \{true, false\}$ such that $valid_\rho(i,d,I)$ checks whether the total time for reaching the state $s_i[d]$ in ρ belongs to the time interval I, formally

$$valid_\rho(i,d,I) = \begin{cases} d \leq d_i \wedge r(i,d) \in I & \text{if } d_i \in \mathbb{R}_{\geq 0} \\ r(i,d) \in I & \text{if } d_i = \infty \\ d \leq d_n \wedge r(i,d) \in I & \text{if } d_i = d_n^{\leq} \\ d < d_n \wedge r(i,d) \in I & \text{if } d_i = d_n^{<} \;. \end{cases}$$

Let us now give some example of the application of the $valid_\rho(i,d,I)$ function. Figure 6 illustrates a run $\rho = s_0 \xrightarrow{d_0} s_0[d_0] \longrightarrow s_1 \xrightarrow{d_1} s_1[d_1] \longrightarrow s_2 \xrightarrow{d_2} s_2[d_2] \longrightarrow \ldots$ and three points (marked with $\times$). We see that $valid_\rho(1,d,I)$ is false because $s_1[d]$ lies outside the interval I. Similarly, $valid_\rho(2,d'',I)$ is false because $s_2[d'']$ is not a part of the run (since $d'' > d_2$). Finally, $valid_\rho(2,d',I)$ is true because $s_2[d']$ is a part of the run and within I.

Next, we define a function $history_\rho : \mathbb{N}_0 \times \mathbb{R}_{\geq 0} \to 2^{\mathbb{N}_0 \times \mathbb{R}_{\geq 0}}$ s.t. $history_\rho(i,d)$ returns the set of pairs (j,d') that constitute all states $s_j[d']$ in ρ preceding $s_i[d]$, formally $history_\rho(i,d) = \{(j,d') \mid 0 \leq j < i \wedge 0 \leq d' \leq d_j\} \cup \{(i,d') \mid 0 \leq d' < d\}$.

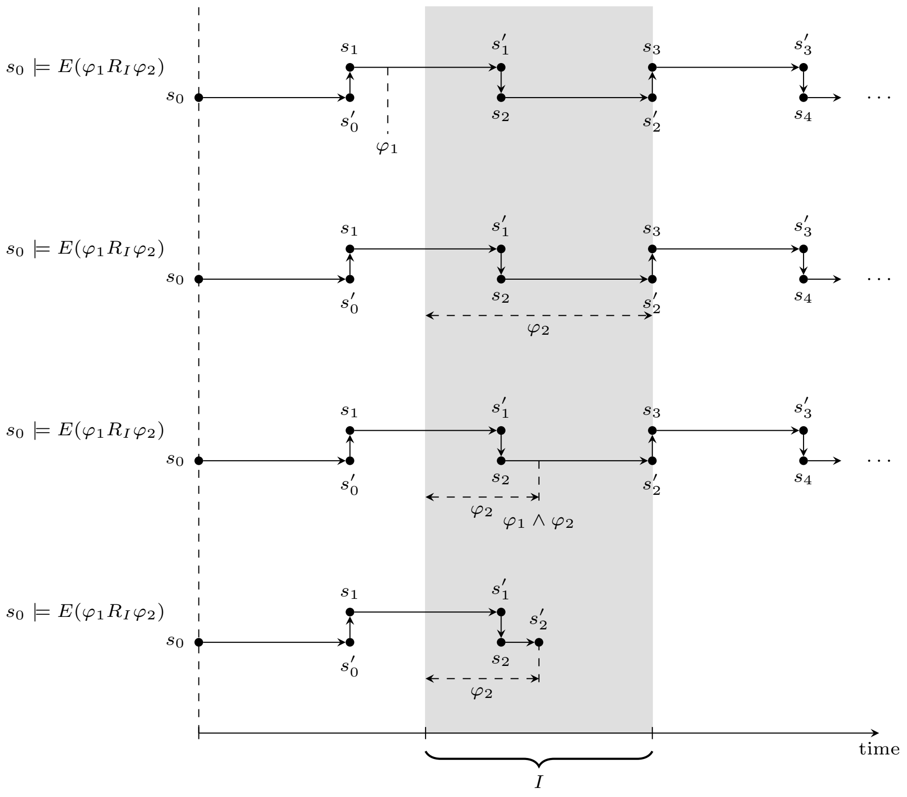


Fig. 8. Illustration of runs satisfying a release formula

Now we can define the satisfaction relation $s \models \varphi$ for a state $s \in S$ in a TTS with propositions $T = (S, \longrightarrow, \mathcal{AP}, \mu)$ and a TCTL formula φ.

$$\begin{array}{ll}
s \models \wp & \text{iff } \wp \in \mu(s) \\
s \models \neg\varphi & \text{iff } s \not\models \varphi \\
s \models \varphi_1 \wedge \varphi_2 & \text{iff } s \models \varphi_1 \text{ and } s \models \varphi_2 \\
s \models E(\varphi_1\, U_I\, \varphi_2) & \text{iff } \exists \rho \in \mathit{MaxRuns}(T, s)\,. \\
& \quad \exists i \geq 0\,.\, \exists d \in \mathbb{R}_{\geq 0}\,.\, \big[\mathit{valid}_\rho(i, d, I) \wedge s_i[d] \models \varphi_2 \wedge \\
& \quad \forall (j, d') \in \mathit{history}_\rho(i, d)\,.\, s_j[d'] \models \varphi_1\big] \\
s \models E(\varphi_1\, R_I\, \varphi_2) & \text{iff } \exists \rho \in \mathit{MaxRuns}(T, s)\,. \\
& \quad \forall i \geq 0\,.\, \forall d \in \mathbb{R}_{\geq 0}\,.\, \mathit{valid}_\rho(i, d, I) \Rightarrow \\
& \quad \big[s_i[d] \models \varphi_2 \vee \exists (j, d') \in \mathit{history}_\rho(i, d)\,.\, s_j[d'] \models \varphi_1\big]
\end{array}$$

The operators $A(\varphi_1\, U_I\, \varphi_2)$ and $A(\varphi_1\, R_I\, \varphi_2)$ are defined analogously by replacing the quantification $\exists \rho \in \mathit{MaxRuns}(T, s)$ with $\forall \rho \in \mathit{MaxRuns}(T, s)$.

Figure 7 illustrates the satisfaction of the until formula and Figure 8 illustrates the release formula. In particular, notice that there are four possible ways for a release formula to be satisfied. First, φ_1 may have occurred in the past (outside the interval), which releases φ_2, effectively ensuring that φ_2 need not hold in the interval I at all. Second, φ_2 may not be released, which means that it must hold continuously within the entire interval I. Third, φ_2 can hold continuously in the interval I, until some point in the interval where $\varphi_1 \wedge \varphi_2$ holds, thereby releasing φ_2. Finally, φ_2 can hold continuously in the interval I until the run deadlocks.

As expected, the until and release operators are dual.

Lemma 1 ([33]). *We have* $s \models A(\varphi_1 R_I \varphi_2)$ *iff* $s \models \neg E(\neg\varphi_1 U_I \neg\varphi_2)$*, and* $s \models A(\varphi_1 U_I \varphi_2)$ *iff* $s \models \neg E(\neg\varphi_1 R_I \neg\varphi_2)$*.*

Example 1. Consider again the TAPN model of alternating bit protocol from Section 4. We can express the correctness of the protocol as the property that the sender and receiver never get out of synchrony. This property is violated if the sender is about to send a message with the bit 0 but the receiver is either in the state Receiver0B or Receiver1A, in other words when the receiver is sending or resending an acknowledgment for the bit 0. Such situation should not happen, and symmetrically for the second part of the protocol where a message with the bit 1 is about to be sent. We can express the violation of synchrony by the following TCTL formula: $E(\mathit{true}\, U_{[0,\mathrm{inf})}\, (\mathsf{Sender0A} = 1 \wedge (\mathsf{Receiver0B} = 1 \vee \mathsf{Receiver1A} = 1)) \vee (\mathsf{Sender1A} = 1 \wedge (\mathsf{Receiver1B} = 1 \vee \mathsf{Receiver0A} = 1)))$. The time interval in the until operator is set to $[0, \mathrm{inf})$ as the correct protocol behaviour should not be violated at any point of its execution. In Section 7 we discuss automatic tool-supported verification of this property.

Another example of a property can require that during the first 20 time units of the protocol execution there are never more than 5 acknowledgment messages in transfer. This can be expressed by the TCTL formula $A(\mathit{false}\, R_{[0,20]}\, (\mathsf{Medium0B} \leq 5 \wedge \mathsf{Medium1B} \leq 5))$ and it is satisfied in the initial marking of the alternating bit protocol.

Finally, we may also ask whether the sender and the receiver eventually finish the transmission of the message with bit 0 and proceed to a message with bit 1. However, the TCTL formula $A(\mathit{true}\, U_{[0,\infty)}\, (\mathsf{Sender1A} = 1 \wedge \mathsf{Receiver1A} = 1))$ expressing this property is false due to several reasons. First of all, in the initial marking the sender is not forced to initiate the sending of the first message and time can elapse for ever. This can be fixed by adding age invariants at all sender and receiver places in order to enforce urgency. However, as the medium is lossy, there is another maximal run where the retransmitted message gets repeatedly lost and such run also violates our formula.

7 Translations Preserving TCTL Model Checking

In this section, we shall present a general framework for arguing when a simulation of one time dependent system by another preserves satisfiability of TCTL formulae. We define the notion of one-by-many correspondence, a relation between two TTSs A and B, such that if A is in one-by-many correspondence with

B then every transition in A can be simulated by a sequence of transitions in B. Further, every TCTL formula φ can be algorithmically translated into a formulate $tr(\varphi)$ s.t. $A \models \varphi$ iff $B \models tr(\varphi)$. In the rest of this section, we shall use A and B to refer to the original and the translated system, respectively. The text of the next subsection is to a large extend based on [33] where the reader can find a more detailed exposition and proofs.

7.1 One-by-Many Correspondence

As the system B is simulating a single transition of A by a sequence of transitions, the systems A and B are comparable only in the states before and after this sequence was performed. We say that B is *stable* in such states and introduce a fresh atomic proposition called *stable* to explicitly identify this situation. We now define three conditions that B should possess in order to apply to our framework. A TTS $(S, \rightarrow, \mathcal{AP}, \mu)$ s.t. $\mathit{stable} \in \mathcal{AP}$ is

- *delay-implies-stable* if for any $s \in S$, it holds that $s \xrightarrow{d}$ for some $d > 0$ implies $s \models \mathit{stable}$,
- *delay-preserves-stable* if for any $s \in S$ such that $s \models \mathit{stable}$, if $s \xrightarrow{d} s[d]$ then $s[d] \models \mathit{stable}$ for all $d \in \mathbb{R}_{\geq 0}$, and
- *eventually-stable* if for any $s_0 \in S$ such that $s_0 \models \mathit{stable}$ and for any infinite sequence of discrete transitions $\rho = s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow s_4 \longrightarrow \ldots$ or any finite nonempty sequence of discrete transitions $\rho = s_0 \longrightarrow s_1 \longrightarrow \cdots \longrightarrow s_n \not\longrightarrow$ there exists an index $i \geq 1$ such that $s_i \models \mathit{stable}$.

We write $s \rightsquigarrow s'$ if there is a sequence $s = s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow \cdots \longrightarrow s_n = s'$ s.t. $s \models \mathit{stable}$, $s' \models \mathit{stable}$, and $s_j \not\models \mathit{stable}$ for $1 \leq j \leq n-1$.

Definition 6. *Let $A = (S, \rightarrow_A, \mathcal{AP}_A, \mu_A)$ and $B = (T, \rightarrow_B, \mathcal{AP}_B, \mu_B)$ be two TTSs s.t. stable $\in \mathcal{AP}_B$ and B is a* delay-implies-stable *and* delay-preserves-stable *TTS. A relation $\mathcal{R} \subseteq S \times T$ is a* one-by-many correspondence *if there exists a function $tr_p : \mathcal{AP}_A \longrightarrow \mathcal{AP}_B$ such that whenever $s \,\mathcal{R}\, t$ then*

1. *$t \models \mathit{stable}$,*
2. *$s \models \wp$ iff $t \models tr_p(\wp)$ for all $\wp \in \mathcal{AP}_A$,*
3. *if $s \longrightarrow s'$ then $t \rightsquigarrow t'$ and $s' \,\mathcal{R}\, t'$,*
4. *if $s \xrightarrow{d} s[d]$ then $t \xrightarrow{d} t[d]$ and $s[d] \,\mathcal{R}\, t[d]$ for all $d \in \mathbb{R}_{\geq 0}$,*
5. *if $t \rightsquigarrow t'$ then $s \longrightarrow s'$ and $s' \,\mathcal{R}\, t'$, and*
6. *if $t \xrightarrow{d} t[d]$ then $s \xrightarrow{d} s[d]$ and $s[d] \,\mathcal{R}\, t[d]$ for all $d \in \mathbb{R}_{\geq 0}$.*

If B is moreover an eventually-stable *TTS, then we say that $\mathcal{R}$ is a* complete one-by-many correspondence. *We write $s \trianglerighteq t$ (resp. $s \trianglerighteq_c t$) if there exists a relation $\mathcal{R}$ which is a one-by-many correspondence (resp. a complete one-by-many correspondence) such that $s \,\mathcal{R}\, t$.*

Now we translate TCTL formulae. Let $\mathcal{AP}_A$ and $\mathcal{AP}_B$ be sets of atomic propositions such that $\mathit{stable} \in \mathcal{AP}_B$ and let $tr_p : \mathcal{AP}_A \longrightarrow \mathcal{AP}_B$ be a function translating atomic propositions. We define $tr : \Phi(\mathcal{AP}_A) \rightarrow \Phi(\mathcal{AP}_B)$ as follows.

$$\begin{aligned}
tr(\wp) &= tr_p(\wp)\\
tr(\neg\varphi_1) &= \neg tr(\varphi_1)\\
tr(\varphi_1 \wedge \varphi_2) &= tr(\varphi_1) \wedge tr(\varphi_2)\\
tr(E(\varphi_1\, U_I\, \varphi_2)) &= E((tr(\varphi_1) \vee \neg stable)\, U_I\, (tr(\varphi_2) \wedge stable))\\
tr(A(\varphi_1\, U_I\, \varphi_2)) &= A((tr(\varphi_1) \vee \neg stable)\, U_I\, (tr(\varphi_2) \wedge stable))\\
tr(E(\varphi_1\, R_I\, \varphi_2)) &= E((tr(\varphi_1) \wedge stable)\, R_I\, (tr(\varphi_2) \vee \neg stable))\\
tr(A(\varphi_1\, R_I\, \varphi_2)) &= A((tr(\varphi_1) \wedge stable)\, R_I\, (tr(\varphi_2) \vee \neg stable))
\end{aligned}$$

We are now ready to state the main result (see [34] for its full proof).

Theorem 3. *Let $A = (S, \rightarrow_A, \mathcal{AP}_A, \mu_A)$ and $B = (T, \rightarrow_B, \mathcal{AP}_B, \mu_B)$ be two TTSs such that $stable \in \mathcal{AP}_B$ and let $s_0 \in S$ and $t_0 \in T$. If $s_0 \mathrel{\overset{\rightharpoonup}{\supseteq}}_c t_0$ then for any TCTL formula φ we have $s_0 \models \varphi$ if and only if $t_0 \models tr(\varphi)$. If $s_0 \mathrel{\overset{\rightharpoonup}{\supseteq}} t_0$ then the claim holds only for any formula φ from the safety fragment of TCTL.*

We finish this subsection by recalling the steps needed in order to apply the framework to a particular translation between two time-dependent systems. Assume that we designed an algorithm that for a given system A constructs a system B together with the notion of stable states in the system B.

1. Show that B is a *delay-implies-stable* and *delay-preserves-stable* TTS (and optionally an *eventually-stable* TTS).
2. Define a proposition translation function $tr_p : \mathcal{AP}_A \longrightarrow \mathcal{AP}_B$.
3. Define a relation $\mathcal{R}$ and show that it fulfills conditions 1–6 of Definition 6.

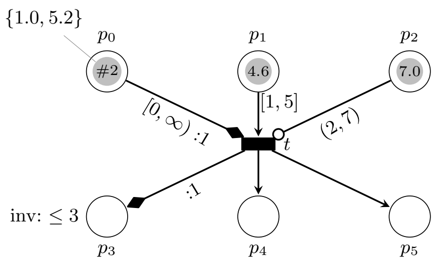
Theorem 3 now allows us to conclude that the translation preserves the full TCTL (or its safety fragment if $\mathcal{R}$ is only a one-by-many correspondence).

There are several reductions from TAPN to networks of timed automata that fit into the general framework [22, 33, 54] and the theory is applicable also to reductions between other time-dependent models including Time Petri nets [18, 24, 25, 35]. For more discussion we refer the reader to [33].

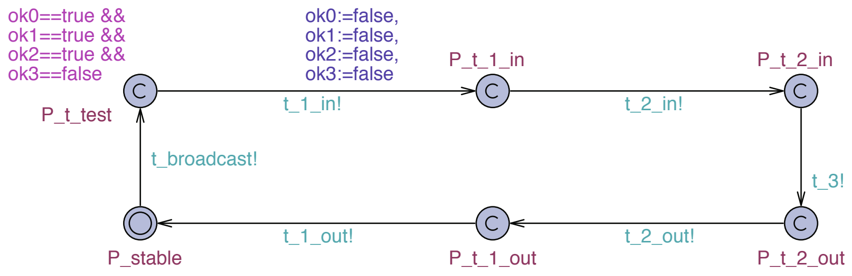
7.2 Translation from TAPN to Networks of Timed Automata

We will now present a translation from k-bounded TAPN (where the maximum number of tokens in every reachable marking is at most k) to networks of timed automata [7, 8] in the UPPAAL style (see e.g. [10] for an introduction to the formalism) in order to demonstrate the applicability of the framework described in the previous subsection.

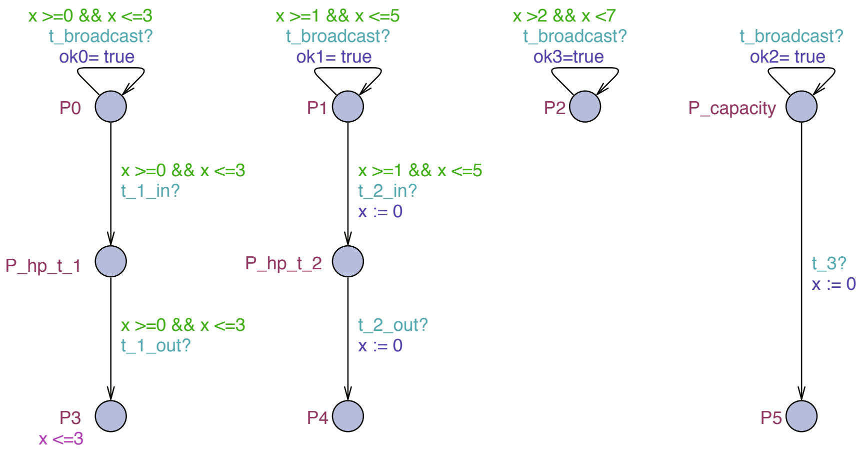
For each token in the net, we create a parallel component in the network of timed automata. As the net is k-bounded, we will need at most k such components. In each of these parallel automata there is a location corresponding to each place in the net. Whenever a TA is in one of these locations, it simulates a token in the corresponding place. Moreover, each automaton has a local clock x which represents the age of the token. All automata simulating the tokens have the same structure, the only difference being their initial locations that



(a) A simple TAPN net



(b) Timed automaton controlling the firing of the transition t



(c) Timed automata templates for each token in the net with local clock x

Fig. 9. Translation from TAPN to UPPAAL network of timed automata

correspond to the initial placement of tokens in the net. Because there may not always be exactly k tokens present during the execution of the net, we add a new location P_capacity to represent currently unused tokens.

In addition to these 'token' automata we create a single control automaton. The purpose is to simulate the firing of transitions and to move tokens around via handshake synchronization initiated by the control automaton. This automaton has a location P_stable which acts as a mutex in the sense that the control automaton moves out of this location once the simulation of a transition begins and returns back once the simulation of the transition ends. Hence the proposition *stable* is defined as (P_stable $= 1$). Moreover, each time the automaton is in P_stable, the token automata in the composed UPPAAL network correspond to a marking in the TAPN. This directly implies that the generated TTS is *delay-preserves-stable* as time delay steps do not change the placement of tokens. We shall now demonstrate how the translation works on an example; the full algorithm is described in [32].

Consider the 5-bounded TAPN in Figure 9a. The translated network of UPPAAL timed automata is given below it. It consists of the control automaton in Figure 9b and five token automata like the one in Figure 9c. The token automata differ only in their initial locations, otherwise they are identical. Hence, in our example, we have two token automata whose initial locations are P0, and the remaining three have initial locations P1, P2 and P_capacity, respectively.

The communication in the network of timed automata begins when the controller broadcasts on the channel t_broadcast. All token automata that can accept the broadcast (i.e. their guards evaluate to true) will participate and set the corresponding global boolean variables ok0, $\ldots$, ok3 to true. The UPPAAL implementation of broadcast allows the controller to move to the location P_t_test only if the associated invariant where ok0, $\ldots$, ok2 are all true and ok3 is false is satisfied, otherwise the broadcast cannot be executed. It is now clear that performing the broadcast is possible only if there is at least one token of an appropriate age in all input places of t, including P_capacity as a new token will be produced, and at the same time there is no token of age in the interval $(2, 7)$ in the place p_2. This is very important for the preservation of liveness TCTL properties, as once the transition firing is initiated, it should be always possibile to successfully finish it. Otherwise the generated transition system would not be *eventually-stable*. For this reason, the reader can notice that while the interval on the arc from p_0 to t is $[0, \infty)$, the corresponding guard in the token automaton on the edge from P0 to P_hp_t_1 requires the age of the token to be also less or equal to 3. The reason for this is that the token will be transported to the place p_3 and its age will be preserved. Any age value larger than 3 would violate the invariant in place p_3; hence as before the *eventually-stable* property might fail.

After the broadcast transition was successfully executed, then the effect of firing the transition t is simulated by a series of handshake synchronizations on channels t_1_in, t_2_in, t_3, t_2_out, t_1_out initiated by the controller and we have a guarantee that such a sequence will always bring the controller to the stable location P_stable, hence ensuring that the generated TTS is *eventually-stable*. Moreover, all locations of the controller are committed (do not allow any time delay steps), which means that the generated TTS is also *delay-implies-stable*.

Table 2. Verification of ABP; dashes indicate more than 5 minutes running time

(a) ABP without symmetry reduction

Messages	UPPAAL	TAPAAL
1	< 1s	< 1s
2	< 1s	< 1s
3	< 1s	1.4s
4	< 1s	16.3s
5	2.2s	165.3s
6	14.4s	-
7	141.9s	-

(b) ABP **with** symmetry reduction

Messages	UPPAAL	TAPAAL
9	2.5s	1.1s
10	3.6s	1.9s
11	10.9s	2.9s
12	24.7s	4.2s
13	89.0s	6.1s
14	239.3s	8.8s
15	-	12.8s

The reason why the tokens are not moved directly to their destinations but are temporarily stored at the locations P_hp_t_1 and and P_hp_t_2 is to avoid the situation where a newly produced token is immediately consumed by firing of the same transition (as it may happen if the transition shared some input and output places).

In case the net contains more transitions, the controller automaton contains a similar loop for all such transitions. This concludes our example. It is relatively easy to argue (see [32] for details) that the original and the translated systems are in one-by-many equivalence. This gives us a polynomial time reduction from the full TCTL model checking problem of timed-arc Petri nets to TCTL model checking problem on networks of timed automata.

We have implemented the reduction described in this section in the open source verification tool TAPAAL [1]. The tool provides a graphical user interface for modelling, simulation and verification of timed-arc Petri nets. Further, we have modelled our example of alternating bit protocol described in Section 4 in TAPAAL and verified the correctness of its behaviour by asking about the violation of synchronization property described in Example 1. As the protocol model is unbounded and contains invariants, automatic verification is not possible. Instead, we considered an under-approximation of the protocol behaviour by limiting the maximum number of messages in transit so that the net becomes bounded. The protocol does not violate the correctness property for any number of messages in transit that we were able to verify. The verification times are measured on an Intel®CPU @ 2.67GHz based computer with 4 GB of memory. The results for a different maximum number of messages in transit are compared in Table 2 with a manually created UPPAAL model of the protocol.

It is clear that both the UPPAAL and TAPAAL models experience the state-space explosion problem so that even relatively small instances take a long verification time. Here the native UPPAAL model is verified faster than the one automatically translated to UPPAAL automata from the TAPN model. On the other hand, the models contain lots of symmetric behaviour, so we also verified both models with symmetry reduction activated. Here, on the other hand, TAPAAL translation provides significantly faster verification compared to the native UPPAAL model. A similar story is true also for a few other experiments we ran and it seems to be connected to the fact that even though the translated models are larger than the manually created UPPAAL models, TAPAAL better exploits the benefits of symmetry reduction. A more detailed investigation of this phenomenon is a part of the future research.

8 Conclusion

In this article we provided an overview of decidability and complexity results related to verification of timed-arc Petri nets extended with transport arcs, age invariants and inhibitor arcs. We described a general framework for arguing when a translation between two time-dependent models preserves TCTL model checking and provided an example of such a translation from timed-arc Petri nets to networks of timed automata. The initial experimental data look promising and in the future we shall consider larger case studies and invest a significant effort into further development of the tool TAPAAL, including its own verification engine.

Among the different extensions of Petri nets with time aspects, we believe that timed-arc Petri nets constitute a convenient modeling formalism and with the recent development of its tool support, TAPN will become an attractive alternative to other modeling approaches.

References

[1] TAPAAL, `http://www.tapaal.net`

[2] UPPAAL, `http://www.uppaal.com`

[3] Abdulla, P.A., Nylén, A.: Better is better than well: On efficient verification of infinite-state systems. In: Proceedings of 15th Annual IEEE Symposium on Logic in Computer Science (LICS 2000), pp. 132–140 (2000)

[4] Abdulla, P.A., Nylén, A.: Timed Petri nets and BQOs. In: Colom, J.-M., Koutny, M. (eds.) ICATPN 2001. LNCS, vol. 2075, pp. 53–70. Springer, Heidelberg (2001)

[5] Abdulla, P.A., Deneux, J., Mahata, P., Nylén, A.: Forward reachability analysis of timed Petri nets. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 343–362. Springer, Heidelberg (2004)

[6] Abdulla, P.A., Mahata, P., Mayr, R.: Dense-timed Petri nets: Checking zenoness, token liveness and boundedness. Logical Methods in Computer Science 3(1), 1–61 (2007)

[7] Alur, R., Dill, D.: Automata for modelling real-time systems. In: Paterson, M. (ed.) ICALP 1990. LNCS, vol. 443, pp. 322–335. Springer, Heidelberg (1990)

[8] Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)

[9] Bartlett, K.A., Scantlebury, R.A., Wilkinson, P.T.: A note on reliable full-duplex transmission over half-duplex links. Communications of the ACM 12(5), 260–261 (1969)

[10] Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)

[11] Berthomieu, B., Diaz, M.: Modeling and verification of time dependent systems using time Petri nets. IEEE Trans. Software Eng. 17(3), 259–273 (1991)

[12] Berthomieu, B., Ribet, P.-O., Vernadat, F.: The tool TINA — construction of abstract state spaces for Petri nets and time Petri nets. International Journal of Production Research 42(14), 2741–2756 (2004)

[13] Berthomieu, B., Peres, F., Vernadat, F.: Bridging the gap between timed automata and bounded time Petri nets. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 82–97. Springer, Heidelberg (2006)

[14] Bolognesi, T., Cremonese, P.: The weakness of some timed models for concurrent systems. Technical Report CNUCE C89-29, CNUCE–C.N.R (1989)

[15] Bolognesi, T., Lucidi, F., Trigila, S.: From timed Petri nets to timed LOTOS. In: Proceedings of the IFIP WG 6.1 Tenth International Symposium on Protocol Specification, Testing and Verification (Ottawa 1990), pp. 1–14. North-Holland, Amsterdam (1990)

[16] Boucheneb, H., Gardey, G., Roux, O.H.: TCTL model checking of time Petri nets. Journal of Logic and Computation 19(6), 1509–1540 (2009)

[17] Bouyer, P., Haddad, S., Reynier, P.-A.: Timed Petri nets and timed automata: On the discriminating power of zeno sequences. Information and Computation 206(1), 73–107 (2008)

[18] Bouyer, P., Haddad, S., Reynier, P.A.: Timed Petri nets and timed automata: On the discriminating power of Zeno sequences. Information and Computation 206(1), 73–107 (2008)

[19] Bowden, F.D.J.: Modelling time in Petri nets. In: Proceedings of the Second Australia-Japan Workshop on Stochastic Models (1996)

[20] Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S.: Kronos: A model-checking tool for real-time systems. In: Y. Vardi, M. (ed.) CAV 1998. LNCS, vol. 1427, pp. 546–550. Springer, Heidelberg (1998)

[21] Bozga, M., Graf, S., Ober, I., Ober, I., Sifakis, J.: The IF toolset. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 237–267. Springer, Heidelberg (2004)

[22] Byg, J., Jørgensen, K.Y., Srba, J.: An efficient translation of timed-arc Petri nets to networks of timed automata. In: Breitman, K., Cavalcanti, A. (eds.) ICFEM 2009. LNCS, vol. 5885, pp. 698–716. Springer, Heidelberg (2009)

[23] Byg, J., Jørgensen, K.Y., Srba, J.: TAPAAL: Editor, simulator and verifier of timed-arc Petri nets. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 84–89. Springer, Heidelberg (2009)

[24] Cassez, F., Roux, O.H.: Structural translation from time Petri nets to timed automata. ENTCS 128(6), 145 (2005); Proc. of AVoCS 2004 (2004)

[25] Dong, J.S., Hao, P., Qin, S., Sun, J., Yi, W.: Timed Automata Patterns. IEEE Transactions on Software Engingeering 34(6), 844–859 (2008)

[26] Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere! Theoretical Computer Science 256(1-2), 63–92 (2001)

[27] Gardey, G., Lime, D., Magnin, M., Roux, O.H.: Romeo: A tool for analyzing time Petri nets. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 418–423. Springer, Heidelberg (2005)

[28] Hack, M.: Petri Net Language. Technical Report MIT-LCS-TR-159, Massachusetts Institute of Technology, Cambridge, MA, USA (1976)

[29] Hanisch, H.M.: Analysis of place/transition nets with timed-arcs and its application to batch process control. In: Ajmone Marsan, M. (ed.) ICATPN 1993. LNCS, vol. 691, pp. 282–299. Springer, Heidelberg (1993)

[30] Heitmann, F., Moldt, D., Mortensen, K.H., Rölke, H.: Petri nets tools database quick overview,
http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html
(accessed: 28.10.2010)

[31] Jacobsen, L., Jacobsen, M., Møller, M.H.: Undecidability of coverability and boundedness for timed-arc Petri nets with invariants. In: Proc. of MEMICS 2009. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2009) ISBN 978-3-939897-15-6

[32] Jacobsen, L., Jacobsen, M., Møller, M.H.: Modelling and verification of timed-arc Petri nets. Master's thesis, Department of Computer Science, Aalborg University, Denmark (2010a), http://tapaal.net

[33] Jacobsen, L., Jacobsen, M., Møller, M.H., Srba, J.: A framework for relating timed transition systems and preserving TCTL model checking. In: Aldini, A., Bernardo, M., Bononi, L., Cortellessa, V. (eds.) EPEW 2010. LNCS, vol. 6342, pp. 83–98. Springer, Heidelberg (2010)

[34] Jacobsen, L., Jacobsen, M., Møller, M.H., Srba, J.: A framework for relating timed transition systems and preserving TCTL model checking. Technical Report FIMU-RS-2010-09, Faculty of Informatics, Masaryk Univ. (2010c)

[35] Janowska, A., Janowski, P., Wróblewski, D.: Translation of Intermediate Language to Timed Automata with Discrete Data. Fundamenta Informaticae 85(1-4), 235–248 (2008)

[36] Lamport, L.: A fast mutual exclusion algorithm. ACM Transactions on Computer Systems 5(1), 1–11 (1987)

[37] Laroussinie, F., Larsen, K.G.: CMC: A tool for compositional model-checking of real-time systems. In: Proceedings of the FIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XI) and Protocol Specification, Testing and Verification (PSTV XVIII), pp. 439–456. Kluwer, B.V (1998)

[38] Mayr, E.W.: An algorithm for the general Petri net reachability problem (preliminary version). In: Proceedings of the 13th Ann. ACM Symposium on Theory of Computing, pp. 238–246. ACM, New York (1981)

[39] Merlin, P.M.: A Study of the Recoverability of Computing Systems. PhD thesis, University of California, Irvine, CA, USA (1974)

[40] Merlin, P.M., Faber, D.J.: Recoverability of communication protocols: Implications of a theoretical study. IEEE Transactions on Communications 24(9), 1036–1043 (1976)

[41] Minsky, M.L.: Computation: Finite and Infinite Machines. Prentice-Hall, Englewood Cliffs (1967)

[42] Nielsen, M., Sassone, V., Srba, J.: Properties of distributed timed-arc Petri nets. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FSTTCS 2001. LNCS, vol. 2245, pp. 280–291. Springer, Heidelberg (2001)

[43] Pelayo, F.L., Cuartero, F., Valero, V., Macia, H., Pelayo, M.L.: Applying timed-arc Petri nets to improve the performance of the MPEG-2 encoding algorithm. In: Proceedings of the 10th International Multimedia Modelling Conference (MMM 2004), pp. 49–56. IEEE Computer Society, Los Alamitos (2004)

[44] Pelayo, F.L., Cuartero, F., Valero, V., Pelayo, M.L., Merayo, M.G.: How does the memory work? by timed-arc Petri nets. In: Proceedings of the 4th IEEE International Conference on Cognitive Informatics (ICCI 2005), pp. 128–135 (2005)

[45] Penczek, W., Pólrola, A.: Advances in Verification of Time Petri Nets and Timed Automata: A Temporal Logic Approach. Springer, Heidelberg (2006)

[46] Petri, C.A.: Kommunikation mit Automaten. PhD thesis, Darmstadt (1962)

[47] Ramchandani, C.: Performance Evaluation of Asynchronous Concurrent Systems by Timed Petri Nets. PhD thesis, Massachusetts Institute of Technology, Cambridge (1973)

[48] Ruiz, V.V., Cuartero Gomez, F., de Frutos Escrig, D.: On non-decidability of reachability for timed-arc Petri nets. In: Proceedings of the 8th International Workshop on Petri Net and Performance Models (PNPM 1999), pp. 188–196 (1999)

[49] Ruiz, V.V., de Frutos Escrig, D., Marroquin Alonso, O.: Decidability of properties of timed-arc petri nets. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 187–206. Springer, Heidelberg (2000)

[50] Ruiz, V.V., Pardo, J.J., Cuartero, F.: Translating TPAL specifications into timed-arc Petri nets. In: Esparza, J., Lakos, C.A. (eds.) ICATPN 2002. LNCS, vol. 2360, pp. 414–433. Springer, Heidelberg (2002)

[51] Ruiz, V.V., Pelayo, F.L., Cuartero, F., Cazorla, D.: Specification and analysis of the MPEG-2 video encoder with timed-arc Petri nets. Electronic Notes Theoretial Computer Science 66(2) (2002)

[52] Sifakis, J.: Use of Petri nets for performance evaluation. In: Proceedings of the Third International Symposium IFIP W.G. 7.3., Measuring, Modelling and Evaluating Computer Systems (Bonn-Bad Godesberg), pp. 75–93. Elsevier Science Publishers, Amsterdam (1977)

[53] Sifakis, J., Yovine, S.: Compositional specification of timed systems. In: Puech, C., Reischuk, R. (eds.) STACS 1996. LNCS, vol. 1046, pp. 347–359. Springer, Heidelberg (1996)

[54] Srba, J.: Timed-arc Petri nets vs. networks of timed automata. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 385–402. Springer, Heidelberg (2005)

[55] Srba, J.: Comparing the expressiveness of timed automata and timed extensions of Petri nets. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 15–32. Springer, Heidelberg (2008)

[56] Wang, J.: Timed Petri Nets, Theory and Application. Kluwer Academic Publishers, Dordrecht (1998) ISBN ISBN 0-7923-8270-6

Efficient Algorithms for Handling Nondeterministic Automata

Tomáš Vojnar

Faculty of Information Technology, Brno University of Technology
Božetěchova 2, CZ-612 66 Brno, Czech Republic
vojnar@fit.vutbr.cz

Finite (word, tree, or omega) automata play an important role in different areas of computer science, including, for instance, formal verification. Often, deterministic automata are used for which traditional algorithms for important operations such as minimisation and inclusion checking are available. However, the use of deterministic automata implies a need to determinise nondeterministic automata that often arise during various computations even when the computations start with deterministic automata. Unfortunately, determinisation is a very expensive step since deterministic automata may be exponentially bigger than the original nondeterministic automata. That is why, it appears advantageous to avoid determinisation and work directly with nondeterministic automata. This, however, brings a need to be able to implement operations traditionally done on deterministic automata on nondeterministic automata instead. In particular, this is the case of inclusion checking and minimisation (or rather reduction of the size of automata). In the talk, we review several recently proposed techniques for inclusion checking on nondeterministic finite word and tree automata as well as Büchi automata. These techniques are based on using the so called antichains, possibly combined with a use of suitable simulation relations (and, in the case of Büchi automata, the so called Ramsey-based or rank-based approaches). Further, we discuss techniques for reducing the size of nondeterministic word and tree automata using quotienting based on the recently proposed notion of mediated equivalences. The talk is based on several common works with Parosh Aziz Abdulla, Ahmed Bouajjani, Yu-Fang Chen, Peter Habermehl, Lisa Kaati, Richard Mayr, Tayssir Touili, Lorenzo Clemente, Lukáš Holík, and Chih-Duo Hong.

Acknowledgement. The works on which the talk is based were supported by the Czech Science Foundation (projects P103/10/0306 and 102/09/H042) and the Czech Ministry of Education (projects COST OC10009 and MSM 0021630528).

The Straight-Line RAC Drawing Problem Is NP-Hard*

Evmorfia N. Argyriou, Michael A. Bekos, and Antonios Symvonis

School of Applied Mathematical & Physical Sciences,
National Technical University of Athens, Greece
{fargyriou,mikebekos,symvonis}@math.ntua.gr

Abstract. Recent cognitive experiments have shown that the negative impact of an edge crossing on the human understanding of a graph drawing, tends to be eliminated in the case where the crossing angles are greater than 70 degrees. This motivated the study of *RAC drawings*, in which every pair of crossing edges intersects at right angle. In this work, we demonstrate a class of graphs with unique RAC combinatorial embedding and we employ members of this class in order to show that it is $\mathcal{NP}$-hard to decide whether a graph admits a straight-line RAC drawing.

1 Introduction

In the graph drawing literature, the problem of finding aesthetically pleasant drawings of graphs has been extensively studied. The graph drawing community has introduced and studied several criteria that judge the quality of a graph drawing, such as the number of crossings among pairs of edges, the number of edge bends, the maximum edge length, the total area occupied by the drawing and so on (see the books [5,17]).

Motivated by the fact that the edge crossings have negative impact on the human understanding of a graph drawing [20], a great amount of research effort has been devoted on the problem of finding drawings with minimum number of edge crossings. Unfortunately, this problem is $\mathcal{NP}$-complete in general [12]. However, recent eye-tracking experiments by Huang et al. [15,16] indicate that the negative impact of an edge crossing is eliminated in the case where the crossing angle is greater than 70 degrees. These results motivated the study of a new class of drawings, called *right-angle drawings* or *RAC drawings* for short [1,7,8,9]. A RAC drawing of a graph is a polyline drawing in which every pair of crossing edges intersects at right angle.

Didimo, Eades and Liota [8] proved that it is always feasible to construct a RAC drawing of a given graph with at most three bends per edge. In this work, we prove that the problem of determining whether an input graph admits a straight-line RAC drawing is $\mathcal{NP}$-hard.

* Funded by the Operational Programme on Education and Lifelong Learning (Action Hrakleitos-II) which is co-financed by Greece and the European Union (European Social Fund NSRF 2007-2013).

1.1 Related Work

Didimo et al. [8] initiated the study of RAC drawings and showed that any straight-line RAC drawing with n vertices has at most $4n - 10$ edges and that any graph admits a RAC drawing with at most three bends per edge. Angelini et al. [1] showed that the problem of determining whether an acyclic planar digraph admits a straight-line upward RAC drawing is $\mathcal{NP}$-hard. Furthermore, they constructed digraphs admitting straight-line upward RAC drawings, that require exponential area. Di Giacomo et al. [7] studied the interplay between the crossing resolution, the maximum number of bends per edges and the required area. Didimo et al. [9] presented a characterization of complete bipartite graphs that admit a straight-line RAC drawing. Arikushi et al. [4] studied polyline RAC drawings in which each edge has at most one or two bends and proved that the number of edges is at most $O(n)$ and $O(n \log^2 n)$, respectively. Dujmovic et al. [10] studied α *Angle Crossing* (or αAC for short) drawings, i.e., drawings in which the smallest angle formed by an edge crossing is at least α. In their work, they presented upper and lower bounds on the number of edges. Van Kreveld [18] showed that the quality of a planar drawing of a planar graph, evaluated in terms of area required, edge-length and angular resolution, can be improved if one allows right-angle crossings.

Closely related to the RAC drawing problem, is the angular resolution maximization problem, i.e., the problem of maximizing the smallest angle formed by any two adjacent edges. Note that both problems correlate the resolution of a graph with the visual distinctiveness of the edges in a graph drawing. Formann et al. [11] introduced the notion of the angular resolution of straight-line drawings. In their work, they proved that determining whether a graph of maximum degree d admits a drawing of angular resolution $\frac{2\pi}{d}$ (i.e., the obvious upper bound) is $\mathcal{NP}$-hard. They also presented upper and lower bounds on the angular resolution for several types of graphs of maximum degree d. Malitz and Papakostas [19] proved that for any planar graph of maximum degree d, it is possible to construct a planar straight-line drawing with angular resolution $\Omega(\frac{1}{7^d})$. Garg and Tamassia [13] presented a continuous tradeoff between the area and the angular resolution of planar straight-line drawings. For the case of connected planar graphs with n vertices and maximum degree d, Gutwenger and Mutzel [14] presented a linear time algorithm that constructs planar polyline grid drawings on a $(2n-5) \times (\frac{3}{2}n - \frac{7}{2})$ grid with at most $5n-15$ bends and minimum angle greater than $\frac{2}{d}$. Bodlaender and Tel [6] showed that planar graphs with angular resolution at least $\frac{\pi}{2}$ are rectilinear. Argyriou et al. [2] studied a generalization of the crossing and angular resolution maximization problems, in which the minimum of these quantities is maximized and presented optimal algorithms for complete graphs and a force-directed algorithm for general graphs.

The rest of this paper is structured as follows: In Section 2, we introduce preliminary properties and notation. In Section 3, we present a class of graphs with unique RAC combinatorial embedding. In Section 4, we show that the straight-line RAC drawing problem is $\mathcal{NP}$-hard. We conclude in Section 5 with open problems.

2 Preliminaries

Let $G = (V, E)$ be a simple, undirected graph drawn in the plane. We denote by $\Gamma(G)$ the drawing of G. The following properties are used in the rest of this paper.

Property 1 (Didimo, Eades and Liota [8]). In a straight-line RAC drawing there cannot be three mutually crossing edges.

Property 2 (Didimo, Eades and Liota [8]). In a straight-line RAC drawing there cannot be a triangle $\mathcal{T}$ and two edges (a, b) and (a, b'), such that a lies outside $\mathcal{T}$ and b, b' lie inside $\mathcal{T}$.

3 A Class of Graphs with Unique RAC Combinatorial Embedding

The $\mathcal{NP}$-hardness proof employs a reduction from the well-known 3-SAT problem. However, before we proceed with the reduction details, we first provide a graph, referred to as *augmented square antiprism graph*, which has the following property: All RAC drawings of this graph have two "symmetric" combinatorial embeddings. Figures 1a and 1b illustrate this property. Observe that the augmented square antiprism graph consists of a "central" vertex v_0, which is incident to all vertices of the graph, and two quadrilaterals (refer to the dashed and bold drawn squares in Figure 1b), that are denoted by $\mathcal{Q}_1$ and $\mathcal{Q}_2$ in the remainder of this paper. Removing the central vertex, the remaining graph corresponds to the skeleton of a square antiprism, and, it is commonly referred to as *square antiprism graph*.

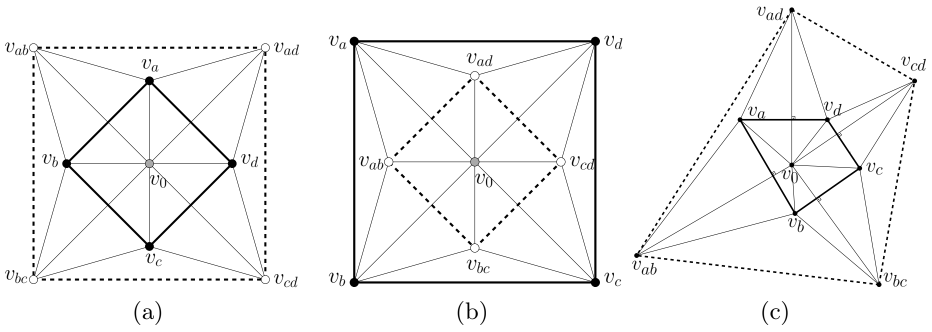


Fig. 1. (a)-(b) Two different RAC drawings of the augmented square antiprism graph with different combinatorial embeddings. (a)-(c) Two different RAC drawings with the same combinatorial embedding

If we replace the two quadrilaterals with two triangles, then the implied graph is the *augmented triangular antiprism graph*. Didimo et al. [8], who showed that any n-vertex graph which admits a RAC-drawing can have at most $4n-10$ edges, used the augmented triangular antiprism graph, as an example of a graph that achieves the bound of $4n - 10$ edges (see Figure 1c in [8]). In contrast to the augmented triangular antiprism graph, the augmented square antiprism graph does not achieve this upper bound. In general, the class of *the augmented k-gon antiprism graphs,* $k \geq 3$, is a class of non-planar graphs, that all admit RAC drawings. Recall that any planar n-vertices graph, should have $3n-6$ edges, and since an augmented k-gon antiprism graph has $2k + 1$ vertices and $5k$ edges, it is not planar for the entire class of these graphs.

Theorem 1. *Any straight-line RAC drawing of the augmented square antiprism graph has two combinatorial embeddings.*

Sketch of proof. The proof of this theorem is implied by the following properties:

- *Property A:* There does not exist a RAC drawing of the augmented square antiprism graph in which the central vertex v_0 lies on the exterior of quadrilateral $\mathcal{Q}_i$, $i = 1, 2$, and an edge connecting v_0 with a vertex of $\mathcal{Q}_i$ crosses an edge of $\mathcal{Q}_i$.
- *Property B:* In any RAC drawing of the augmented square antiprism graph, quadrilateral $\mathcal{Q}_i$, $i = 1, 2$, is drawn planar.
- *Property C:* In any RAC drawing of the augmented square antiprism graph, the central vertex v_0 lies in the interior of quadrilateral $\mathcal{Q}_i$, $i = 1, 2$.
- *Property D:* There does not exist a RAC drawing of the augmented square antiprism graph where an edge emanating from vertex v_0 towards a vertex of quadrilateral $\mathcal{Q}_i$, $i = 1, 2$, crosses quadrilateral $\mathcal{Q}_i$.
- *Property E:* There does not exist a RAC drawing of the augmented square antiprism graph in which quadrilateral $\mathcal{Q}_1$ intersects $\mathcal{Q}_2$.

Due to space constraints, the proofs of these properties are omitted. The proofs make use of elementary geometric properties, they heavily use Properties 1 and 2, and are based on exhaustive cases analysis on the relative positions of (a) the central vertex v_0, and, (b) quadrilaterals $\mathcal{Q}_1$ and $\mathcal{Q}_2$. For more details, we refer the reader to [3]. $\square$

We extend the augmented square antiprism graph, by appropriately "glueing" multiple instances of it, the one next to the other, either horizontally or vertically. Figure 2a demonstrates how a horizontal extension of two instances, say G and G', is realized, i.e., by identifying two "external" vertices, say v and v', of G with two "external" vertices of G' (refer to the gray-colored vertices of Figure 2a), and by employing an additional edge (refer to the dashed drawn edge of Figure 2a), which connects an "internal" vertex, say u, of G with the corresponding "internal" vertex, say u', of G'. Let $G \oplus G'$ be the graph produced by a horizontal or vertical extension of G and G'. Since each of G and G' has two RAC combinatorial embeddings each, one would expect that $G \oplus G'$ would have four possible RAC combinatorial embeddings. We will show that this is not true and, more precisely, that there only exists a single RAC combinatorial embedding.

Theorem 2. *Let G and G' be two instances of the augmented square antiprism graph. Then, $G \oplus G'$ has a unique RAC combinatorial embedding.*

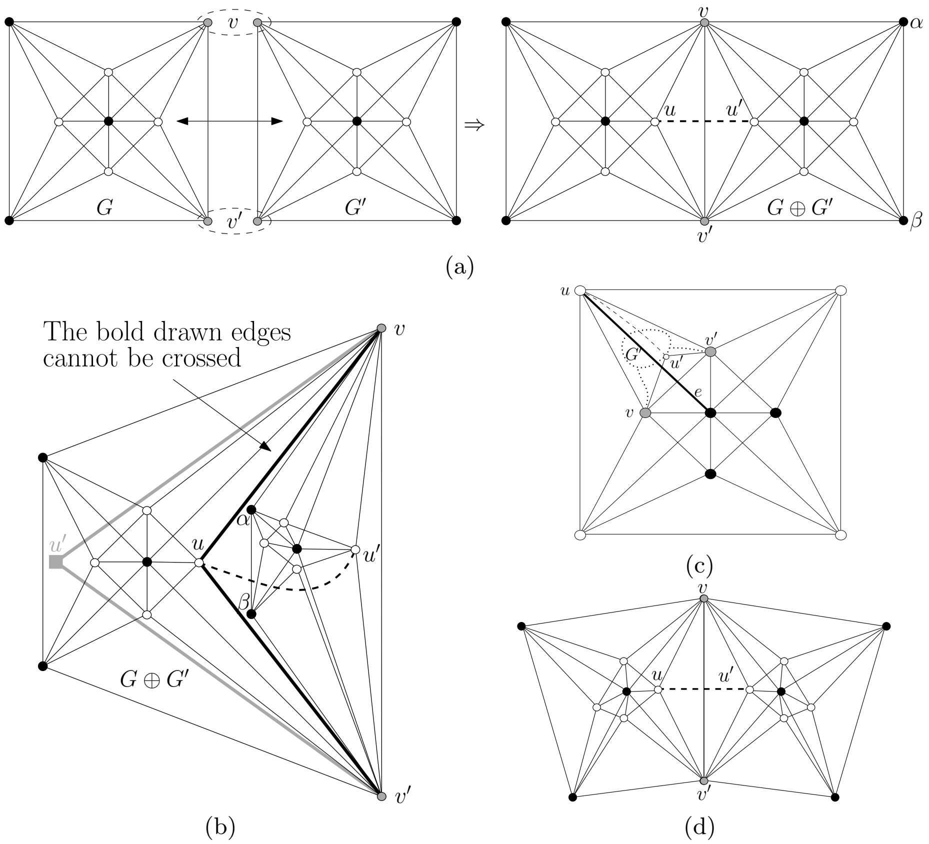


Fig. 2. (a) Horizontal extension of two instances of the augmented square antiprism graph, (b) The additional (dashed) edge does not permit the second instance to be drawn in the interior of the first one. (c) The vertices which are identified, during a horizontal or vertical extension (v and v' in Figure), should be on the external face of each augmented square antiprism graph. (d) At each extension step the new instance of the augmented square antiprism graph may introduce a "turn".

Proof. Assume first that in a RAC drawing of $G \oplus G'$, vertices v and v' are on the external quadrilateral of G and graph G' is drawn completely in the interior of G (see Figure 2b; since v and v' are on the external face of G', vertices α and β in Figure 2b should also be on the external face of G'). First observe that vertex u' of G', which is incident to vertices v and v', cannot reside to the "left" of both edges (u, v) and (u, v') (refer to the bold drawn edges of Figure 2b), since this would lead to a situation where three edges mutually cross and, subsequently, to a violation of Property 1 (see the gray-colored square vertex of Figure 2b). Therefore, vertex u' should lie within the triangular face of G formed by vertices u, v and v'. The same similarly holds for the central vertex of G', which is also incident to vertices v and v'. By Property 2, any common neighbor of vertices u' and v should also lie within the same triangular face of G, which progressively implies that entire graph G' should reside within this face, as in Figure 2b. However, in this case and since u' is incident to v and v', edge (u, u'), which is used on a horizontal or a vertical extension, crosses the interior of G', which is not permitted. This suggests that graph G' should be on the exterior of G.

Now assume that vertices v and v', which are identified, during a horizontal or vertical extension, are along the internal quadrilateral of G in a RAC drawing of $G \oplus G'$. This is illustrated in Figure 2c. Then, the edge, say e, which perpendicularly crosses edge (v, v') and emanates from the external quadrilateral towards the central vertex of G (refer to the bold solid edge of Figure 2c) will be involved in crossings with G'. More precisely, we focus on vertex u' of G', which is incident to vertices v and v'. These edges will inevitably introduce non-right angle crossings, since one of them should cross edge e. Therefore, the vertices that are identified, during a horizontal or vertical extension, should always be on the external face of each augmented square antiprism graph and, subsequently, the drawing of the graph produced by a horizontal or vertical extension will resemble the one of Figure 2a, i.e., it has a unique embedding. □

Note that the extension which is given in Figure 2a, is ideal. In the general case, at each extension step the new instance of the augmented square antiprism graph may introduce a "turn", as in Figure 2d. We observe that by "glueing" a new instance of the augmented square antiprism graph on $G \oplus G'$ either by a horizontal or a vertical extension, we obtain another graph of unique RAC combinatorial embedding. In this way, we can define an infinite class of graphs of unique RAC combinatorial embedding. This is summarized in the following theorem.

Theorem 3. *There exists a class of graphs of unique RAC combinatorial embedding.*

4 The Straight-Line RAC Drawing Problem is NP-Hard

Theorem 4. *It is $\mathcal{NP}$-hard to decide whether an input graph admits a straight-line RAC drawing.*

Proof. We will reduce the well-known 3-SAT problem to the straight-line RAC drawing problem. In a 3-SAT instance, we are given a formula ϕ in conjunctive normal form with variables $x_1, x_2, \ldots, x_n$ and clauses $C_1, C_2, \ldots, C_m$, each with three literals. We show how to construct a graph G_ϕ that admits a straight-line RAC drawing $\Gamma(G_\phi)$ if and only if formula ϕ is satisfiable.

Figure 3 illustrates the gadgets of our construction. Each gray-colored square in these drawings corresponds to an augmented square antiprism graph. Adjacent gray squares form an extension (refer, for example, to the topmost gray squares of Figure 3a, which form a "horizontal" extension). There also exist gray squares that are not adjacent, but connected with edges. The legend in Figure 3 describes how the connections are realized.

The gadget that encodes variable x_i of formula ϕ is given in Figure 3a. The gadget of variable x_i consists of a combination of augmented square antiprism

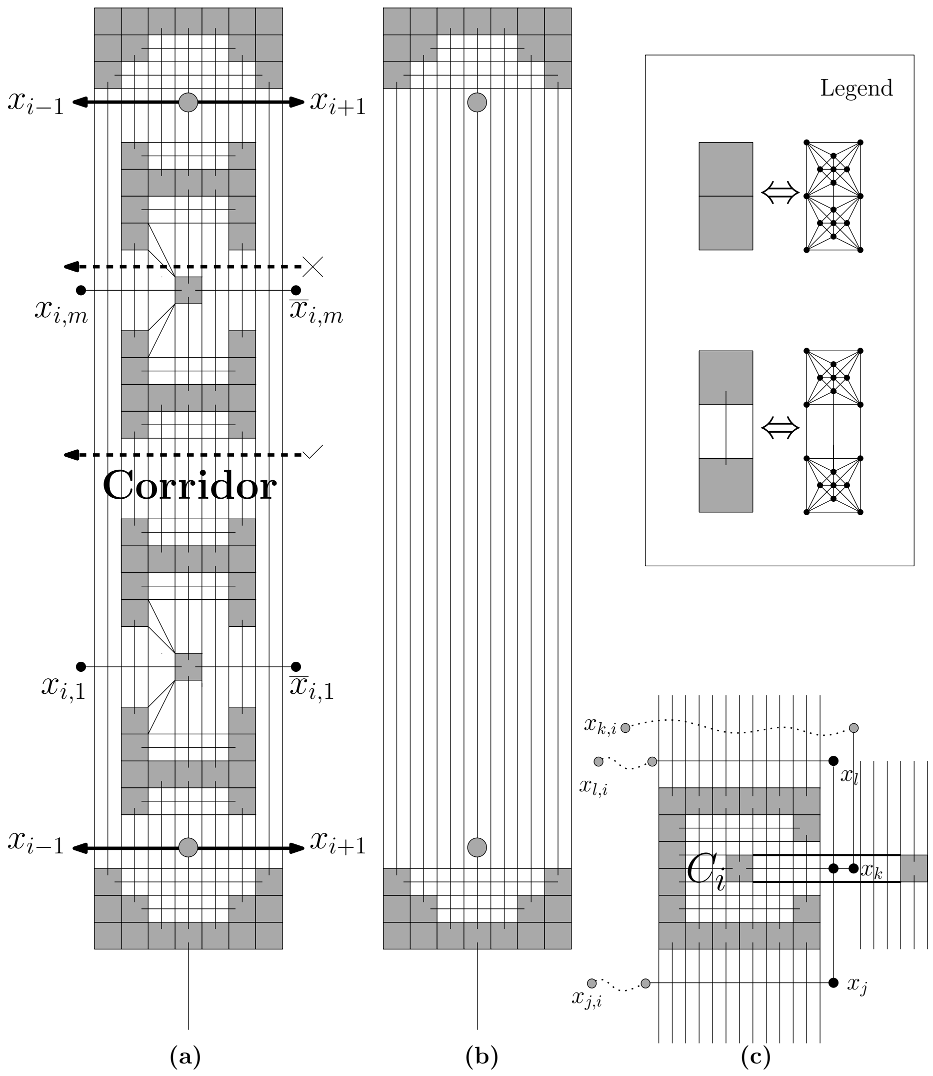


Fig. 3. Gadgets of our construction: (a) Variable gadget, (b) Dummy variable gadget, (c) Clause gadget

graphs, and, "horizontal" and "vertical" edges, which form a tower, whose RAC drawing has unique combinatorial embedding. One side of the tower accommodates multiple vertices that correspond to literal x_i, whereas its opposite side accommodates vertices that correspond to literal $\overline{x_i}$ (refer to vertices $x_{i,1}, \ldots, x_{i,m}$ and $\overline{x}_{i,1}, \ldots, \overline{x}_{i,m}$ in Figure 3a). These vertices are called *variable endpoints*. Then, based on whether on the final drawing the negated vertices will appear to the "left" or to the "right" side of the tower, we will assign a true or a false value to variable x_i, respectively. Pairs of consecutive endpoints $x_{i,j}$ and $x_{i,j+1}$ are separated by a *corridor* (see Figure 3a), which allows perpendicular edges to pass through it (see the bottommost dashed arrow of Figure 3a). Note that this is not possible through a "corridor" formed on a variable endpoint, since there exist four non-parallel edges that "block" any other edge passing through them (see the topmost dashed arrow of Figure 3a). The corridors can have variable height. In the variable gadget of variable x_i, there are also two vertices (they are drawn as gray circles in Figure 3a), which have degree four. These vertices serve as "*connectors*" among consecutive variable gadgets, i.e., these vertices should be connected to their corresponding vertices on the variable gadgets of variables x_{i-1} and x_{i+1}. Note that the connector vertices of the variable gadgets associated with variables x_1 and x_n are connected to connectors of the variable gadgets that correspond to variables x_2 and x_{n-1}, respectively, and to connectors of *dummy variable gadgets*.

Figure 3b illustrates a dummy variable gadget, which (similarly to the variable gadget) consists of a combination of augmented square antiprism graphs, and, "horizontal" and "vertical" edges, which form a tower. Any RAC drawing of this gadget has also unique combinatorial embedding. A dummy variable gadget does not support vertices that correspond to literals. However, it contains connector vertices (they are drawn as gray circles in Figure 3b). In our construction, we use exactly two dummy variable gadgets. The connector vertices of each dummy variable gadget should be connected to their corresponding connector vertices on the variable gadgets associated with variables x_1 and x_n, respectively.

The gadget that encodes the clauses of formula ϕ is illustrated in Figure 3c and resembles a valve. Let $C_i = (x_j \vee x_k \vee x_l)$ be a clause of ϕ. As illustrated in Figure 3c, the gadget which corresponds to clause C_i contains three vertices[1], say x_j, x_k, and x_l, such that: x_j has to be connected to $x_{j,i}$, x_k to $x_{k,i}$ and x_l to $x_{l,i}$ by paths of length two. These vertices (i.e., x_j, x_k, and x_l), referred to as the *clause endpoints*, encode the literals of each clause. Obviously, if a clause contains a negated literal, it should be connected to the negated endpoint of the corresponding variable gadget. The clause endpoints are incident to a vertex "trapped" within two parallel edges (refer to the bold drawn edges of Figure 3c). Therefore, in a RAC drawing of G_ϕ, only two of them can perpendicularly cross these edges, one from top (*top endpoint*; refer to clause endpoint x_l of Figure 3c) and one from bottom (*bottom endpoint*; refer to clause endpoint x_j of Figure 3c). The other one (*right endpoint*; refer to clause endpoint x_k of

[1] With slight abuse of notation, the same term is used to denote variables of ϕ and vertices of G_ϕ.

Figure 3c) should remain in the interior of the two parallel edges. The one that will remain "trapped" on the final drawing will correspond to the true literal of this clause (see Figure 3).

The gadgets, which correspond to variables and clauses of ϕ, are connected together by the skeleton of graph G_ϕ, which is depicted in Figure 4a. The skeleton consists of two main parts, i.e., one "horizontal" and one "vertical". The vertical part accommodates the clause gadgets (see Figure 4a). The horizontal part will be used in order to "plug" the variable gadgets. The long edges that perpendicularly cross (refer to the crossing edges slightly above the horizontal part in Figure 4a), imply that the vertical part should be perpendicular to the horizontal part. The horizontal part of the skeleton is separately illustrated in Figure 4b. Observe that it contains one set of horizontal lines.

Figure 5 shows how the variable gadgets are attached to the skeleton. More precisely, this is accomplished by a single edge, which should perpendicularly cross the set of the horizontal edges of the horizontal part. Therefore, each variable gadget is perpendicularly attached to the skeleton, as in Figure 5. Note that each variable gadget should be drawn completely above of these horizontal edges, since otherwise the connections among variable endpoints and clause endpoints would not be feasible. The connector vertices of the dummy variable

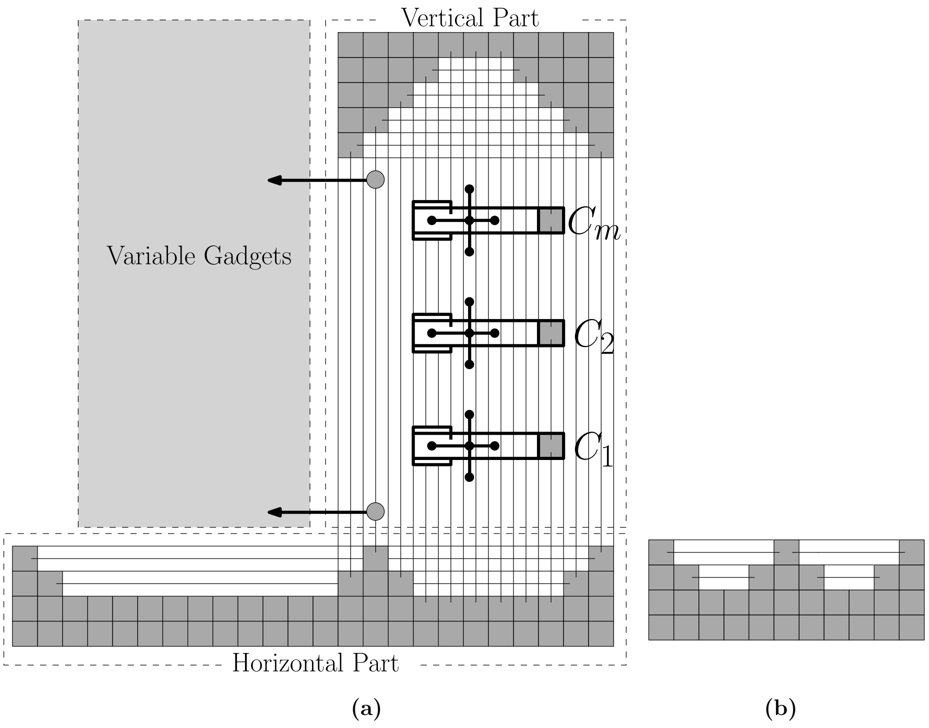
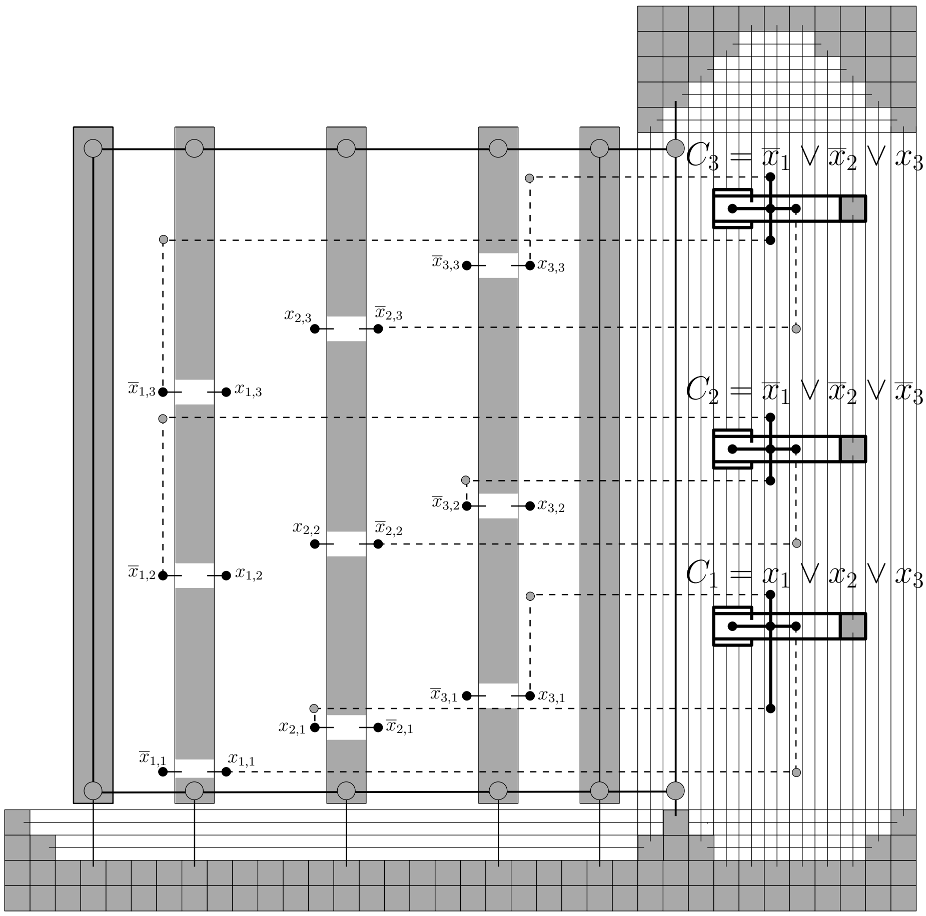


Fig. 4. Illustration of the skeleton of the construction



$C_3 = \overline{x}_1 \vee \overline{x}_2 \vee x_3$
$C_2 = \overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3$
$C_1 = x_1 \vee x_2 \vee x_3$
$\overline{x}_{3,3}$ $x_{3,3}$
$x_{2,3}$ $\overline{x}_{2,3}$
$\overline{x}_{1,3}$ $x_{1,3}$
$\overline{x}_{3,2}$ $x_{3,2}$
$x_{2,2}$ $\overline{x}_{2,2}$
$\overline{x}_{1,2}$ $x_{1,2}$
$\overline{x}_{3,1}$ $x_{3,1}$
$x_{2,1}$ $\overline{x}_{2,1}$
$\overline{x}_{1,1}$ $x_{1,1}$

Fig. 5. The reduction from 3-SAT to the straight-line RAC drawing problem. The input formula is $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee x_3)$. The drawing corresponds to the truth assignment $x_1{=}x_3{=}$true, $x_2{=}$false.

gadgets, the variable gadgets and the vertical part of the construction, ensure that the variable gadgets will be parallel to each other (i.e., they are not allowed to bend) and parallel to the vertical part of the construction.

We now proceed to investigate some properties of our construction. Any path of length two that emanates from a top- or bottom-clause endpoint can reach a variable endpoint either on the left or on the right side of its associated variable gadget. The first edge of this path should perpendicularly cross the vertical edges of the vertical part of the construction and pass through some corridors[2], whereas the second edge will be used to realize the “final” connection with the variable gadget endpoint (see Figure 5). However, the same doesn’t hold for the paths

[2] In Figure 5, the corridors are the gray-colored regions that reside at each variable gadget.

that emanate from a right-clause endpoint. These paths can only reach variable endpoints on the right side of their associated variable gadgets. More precisely, the first edge of the 2-length path should cross one of the two parallel edges (refer to the bold drawn edges of Figure 3c) that "trap" it, whereas the other one should be used to reach (passing through variable corridors) its variable endpoint (see Figure 5).

Our construction ensures that up to translations, rotations and stretchings any RAC drawing of G_ϕ resembles the one of Figure 4. Each tower corresponding to a non-dummy variable gadget of the construction contributes $O(m)$ time to the total construction time. The towers that correspond to dummy variable gadgets trivially contribute constant time to the total construction time. Therefore, we can construct all towers of variable gadgets in $O(nm)$ time. The horizontal part needs an extra $O(n)$ time, whereas the vertical part can be done in $O(m)$ time. Thus, our construction can be completed in $O(nm)$ time in total. Assume now that there is a RAC drawing $\Gamma(G_\phi)$ of G_ϕ. If the negated vertices of the variable gadget that corresponds to x_i, $i = 1, 2, \ldots, n$, lie to the "left" side in $\Gamma(G_\phi)$, then variable x_i is set to true, otherwise x_i is set to false. We argue that this assignment satisfies ϕ. To realize this, observe that there exist three paths that emanate from each clause gadget. The one that emanates from the right endpoint of each clause gadget can never reach a false value. Therefore, each clause of ϕ must contain at least one true literal, which implies that ϕ is satisfiable.

Conversely, suppose that there is a truth assignment that satisfies ϕ. We proceed to construct a RAC drawing $\Gamma(G_\phi)$ of G_ϕ, as follows: In the case where, in the truth assignment, variable x_i, $i = 1, 2, \ldots, n$ is set to true, we place the negated vertices of the variable gadget that corresponds to x_i, to its left side in $\Gamma(G_\phi)$, otherwise to its right side. Since each clause of ϕ contains at least one true literal, we choose this as the right endpoint of its corresponding clause gadget. As mentioned above, it is always feasible to be connected to its variable gadgets by paths of length two. This completes our proof. □

5 Conclusions

In this paper, we proved that it is $\mathcal{NP}$-hard to decide whether a graph admits a straight-line RAC drawing. Didimo et al. [8] proved that it is always feasible to construct a RAC drawing of a given graph with at most three bends per edge. If we permit two bends per edge, does the problem remain $\mathcal{NP}$-hard? It is also interesting to continue the study on the interplay between the number of edges and the required area in order to fill the gaps between the known upper and lower bounds.

References

1. Angelini, P., Cittadini, L., Di Battista, G., Didimo, W., Frati, F., Kaufmann, M., Symvonis, A.: On the perspectives opened by right angle crossing drawings. In: Eppstein, D., Gansner, E.R. (eds.) GD 2009. LNCS, vol. 5849, pp. 21–32. Springer, Heidelberg (2010)

2. Argyriou, E.N., Bekos, M.A., Symvonis, A.: Maximizing the total resolution of graphs. In: 18th International Symposium on Graph Drawing (2010) (to appear)
3. Argyriou, E.N., Bekos, M.A., Symvonis, A.: The straight-line RAC drawing problem is NP-hard. CoRR abs/1009.5227 (2010)
4. Arikushi, K., Fulek, R., Keszegh, B., Moric, F., Toth, C.: Drawing graphs with orthogonal crossings. In: 36th International Workshop on Graph Theoretic Concepts in Computer Science (2010) (to appear)
5. Battista, G.D., Eades, P., Tamassia, R., Tollis, I.G.: Graph Drawing: Algorithms for the Visualization of Graphs. Prentice-Hall, Englewood Cliffs (1999)
6. Bodlaender, H.L., Tel, G.: A note on rectilinearity and angular resolution. Journal of Graph Algorithms and Applications 8, 89–94 (2004)
7. Di Giacomo, E., Didimo, W., Liotta, G., Meijer, H.: Area, curve complexity, and crossing resolution of non-planar graph drawings. In: Eppstein, D., Gansner, E.R. (eds.) GD 2009. LNCS, vol. 5849, pp. 15–20. Springer, Heidelberg (2010)
8. Didimo, W., Eades, P., Liotta, G.: Drawing graphs with right angle crossings. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) Algorithms and Data Structures. LNCS, vol. 5664, pp. 206–217. Springer, Heidelberg (2009)
9. Didimo, W., Eades, P., Liotta, G.: A characterization of complete bipartite graphs. Information Processing Letters 110(16), 687–691 (2010)
10. Dujmovic, V., Gudmundsson, J., Morin, P., Wolle, T.: Notes on large angle crossing graphs. In: 16th Symposium on Computing: the Australasian Theory, pp. 19–24. Australian Computer Society (2010)
11. Formann, M., Hagerup, T., Haralambides, J., Kaufmann, M., Leighton, F., Symvonis, A., Welzl, E., Woeginger, G.: Drawing graphs in the plane with high resolution. SIAM Journal of Computing 22(5), 1035–1052 (1993)
12. Garey, M., Johnson, D.: Crossing number is NP-complete. SIAM Journal of Algebraic Discrete Methods 4, 312–316 (1983)
13. Garg, A., Tamassia, R.: Planar drawings and angular resolution: Algorithms and bounds (extended abstract). In: 2nd Annual European Symposium on Algorithms, pp. 12–23 (1994)
14. Gutwenger, C., Mutzel, P.: Planar polyline drawings with good angular resolution. In: Whitesides, S.H. (ed.) GD 1998. LNCS, vol. 1547, pp. 167–182. Springer, Heidelberg (1999)
15. Huang, W.: Using eye tracking to investigate graph layout effects. In: Asia-Pacific Symposium on Visualization, pp. 97–100 (2007)
16. Huang, W., Hong, S.-H., Eades, P.: Effects of crossing angles. In: IEEE Pacific Visualization Symposium, pp. 41–46. IEEE, Los Alamitos (2008)
17. Kaufmann, M., Wagner, D. (eds.): Drawing Graphs. LNCS, vol. 2025. Springer, Heidelberg (2001)
18. van Kreveld, M.: The quality ratio of RAC drawings and planar drawings of planar graphs. In: 18th International Symposium on Graph Drawing (2010) (to appear)
19. Malitz, S.M., Papakostas, A.: On the angular resolution of planar graphs. In: 24th Annual ACM Symposium on Theory of Computing, pp. 527–538. ACM, New York (1992)
20. Purchase, H.C., Carrington, D.A., Allder, J.-A.: Empirical evaluation of aesthetics-based graph layout. Empirical Software Engineering 7(3), 233–255 (2002)

Tracking the Evolution of Code Clones

Tibor Bakota

University of Szeged, Hungary,
bakotat@inf.u-szeged.hu

Abstract. It is believed by many academic and industrial experts, that source code cloning (copy&paste programming) represents a significant threat to maintainability in an evolving software system. The real threat does not lie in the existence of duplications, but the fears are in connection with their evolution. There exist an abundance of algorithms for finding code clones in one particular version of a software system, but eliminating or even evaluating these clones often seems hopeless, as there may exist several thousands of them. Tracking the evolution of individual clones might solve the problem, as it could help identifying the inconsistently changing duplications: the clones which are really dangerous at a particular moment. In this paper we present an approach for mapping code duplications from one particular version of the software to another one, based on a similarity distance function. For the suspicious evolution patterns we introduce the term of ”clone smells”. By defining the relevant categories of the possible evolution patterns, the proposed method also gives a clue about why the reported code fragments might be dangerous. In the case study, clone smells were extracted, evaluated, and manually categorized throughout many versions of the jEdit system. The findings suggest that roughly half of the reported smells refer to inconsistent changes in the code.

1 Introduction

During software development, when developers are under constant pressure of deadlines, it is a common practice to reuse source code by simply copying parts of it, and eventually performing smaller modifications on it (it has been estimated that both industrial and open source systems contain in average about 20% of duplicated code [1]). Although this approach can reduce software development time, the price in the long-term will usually be paid in the form of increased maintainability costs. One of the primary concerns is that if the original code segment needs to be corrected, all the copied parts need to be checked and changed accordingly as well. By inadvertently neglecting to change the related duplications, the programmers may leave bugs in the code and introduce inconsistencies. On the other hand, some researches [2,3] point out that there exist situations when duplicating code could even be beneficial, and clones should not always be considered as bad smells.

There exist an abundance of clone detection algorithms ranging from lexical (token-based) [4], through AST-based [5] to metric-based [6] approaches. These methods act on one particular version of the software and as a result they provide a detailed list of copied code segments, which may eventually contain several thousand items in case of real-size software. Despite the obvious fact that the eventual danger of using clones is in connection with their evolution, the field has only recently become a hot research area [7, 8, 9, 10]. There are basically three kinds of approaches that map clones between different versions of a software. Two of the techniques utilize single-version clone detection. The first category of approaches detects clones in a reference version and calculates those of following versions using change information from a version repository [11,12]. The second class of approaches detects clones for all versions of the program. Clones are then retroactively mapped using heuristics [13,8,14]. The third category of approaches uses incremental clone detection for finding clones in subsequent versions of a software.

In this paper we follow the second approach: a heuristics called *evolution mapping* is used to relate clone instances from different versions of a system. The mapping between the clones is trivial in some special cases, but generally, a sophisticated approach is needed, whose details will be presented in the following sections of this paper. By using the evolution mapping concept, we introduce the notion of *clone smells* which, similarly to bad code smells [15], refer to particular code parts that should be further inspected manually. The smells are defined based on the possible categories of clone evolution patterns. The advantage of this approach is that instead of focusing on a set of several thousand copied code segments and eliminating them (even those that will probably never be modified again), the developers can concentrate on those segments which may represent maintainability threats. In the paper we provide definitions of five different clone smells, which cover the possible clone evolution patterns on clone instance level. To evaluate the approach we executed the algorithm on several versions of the jEdit [16] text editor. The experiment resulted in a list of code fragments that was manually checked. It turned out that more than half of the reported smells occurred because of inconsistent changes in the code.

This paper is a continuation of our conference paper [14], and contains several new contributions:

1. The previously presented optimization approach was based on a greedy algorithm which does not necessarily find the global optimal solution (it may stuck in local optima) for the problem and strongly depends on the order in which the clone instances are considered. This drawback has been resolved here by replacing it with an also polynomial time complexity algorithm which always yields the global optimum.
2. The set of clone smells is different from the one proposed in our previous paper, and here it has also been extended to the clone class[1] level, which proved useful for identifying new types of clone evolution patterns.
3. We evaluated and manually checked the results on much more versions of an open source system.

The paper is organized as follows. Section 2 discusses several studies similar to ours. Afterwards, Section 3 describes our approach for creating an evolution mapping of clone instances between versions. Next, in Section 4 we give the definitions and explain the meaning of clone smells. In Section 5 we present the results obtained during the execution of the algorithm on jEdit. Finally, in the last section we round off with conclusions and suggestions for future work.

[1] Clone class is a set of source code segments which are considered to be copies.

2 Related Work

There are a number of papers that deal with various clone detection approaches. Starting from the algorithms which are based on a lexical comparison of the source lines or tokens [4], through the metric-based approaches [6] which use metric values of the code parts in order to identify similar fragments, to the more sophisticated AST-based approaches [5] which require a full syntactic analysis of the source code before the clone detection can take place. Algorithms for tracking clone instances across different versions of a software system has only recently become a hot research area.

Kim et al. [9] proposed a similar approach to ours. They defined the *cloning relationship* between two clone classes based on the lexical similarity of their representatives. In this way a directed acyclic graph was obtained (the nodes are the clone classes and the edges are represented by the evolution pattern relationship). *Clone Genealogy* is a connected component of the above graph where every clone group is connected by at least one evolution pattern. Clone genealogy was used to perform a study on two small Java systems from the viewpoint of the cloning habits of the developers.

Duala-Ekoko et al. [8] proposed a technique for tracking clones in evolving software. They proposed the notion of an abstract Clone Region Descriptor (*CRD*), which describes the clone instances within methods in such a way that it is independent of the exact text or their location in the code. A CRD is a lightweight and abstract description of the location of a clone region (i.e. clone instance) in the source code. Given a CRD and a code base, they identify the corresponding clone region through a series of automatic searches. The attributes used for constructing the CRD are similar to those applied by us for defining our similarity distance function. The subsequent versions of the system were searched thoroughly for code pieces having the same CRD.

Krinke [12] used a version control system of open source systems to identify changes applied to code duplications. They checked whether changes applied to cloned segments were generally consistent (repeated for all instances) or not. Their study revealed that clone classes were consistently changed in roughly half of the cases. Krinke also showed that classes that had been changed inconsistently earlier and were consistently changed later, were comparatively rare. This conclusion reinforces our view that identifying inconsistent changes, which may lead to unexpected behavior in the future, is an important maintenance-related challenge, which was our primary goal in the study.

Göde et al. [10] presented an incremental clone detection algorithm, which detects clones based on the results of the previous revision's analysis. Their algorithm creates a mapping between clones of one revision to the next, supplying information about the addition and deletion of clones. The incremental approach considerably speeds up clone detection itself, and makes it possible to track the changes on-the-fly (even while the developer is typing). They utilized general suffix trees in their approach built on token tables representing tokens of individual files. When a file is added, deleted or modified, the corresponding token table is either added, subtracted or modified accordingly. During the update procedure, the clones from consecutive versions are also related.

Thummalapenta et al. [11] proposed an automatic approach to classify the evolution of source code clone fragments. Their approach is different (and complementary) to this work for three main reasons:

1. Their approach aims at analyzing how clone fragments belonging to the same clone class are maintained, while our focus is to map clones detected in different releases of a software system.
2. They trace clones by starting from clone section pairs using a variant of a code tracing approach to analyze clones. Instead, we map clones based on a similarity distance function defined as a weighted sum of characteristic features such as lexical structure, file name, etc.
3. Our empirical study was performed by tracing clones between software system releases, while they used file revisions instead.

3 Approach

Our approach for finding clone smells consists of the following three steps:

1. Clone detection phase
2. Evolution mapping creation phase
3. Clone smell detection phase

Cloning defines an equivalence relation on the set of copied code segments. Two code segments correspond to each other if they are copies of each other (with respect to the underlying clone detection approach). The classes of the relation are called *clone classes*, and the members of the classes will be referred to as *clone instances* or just simply *clones*.

In the following, C_i, C_j will denote arbitrary clone instances, not necessarily from the same class. We shall use the notion of C_i^v, C_j^v, etc. to emphasize version v of the system from which the instances originate. In a similar way, CC_i^v, CC_j^v refer to clone classes of version v, while CI^v stands for the set of instances extracted from version v of the underlying system. We will use CC^v to denote the set of all clone classes in version v of the system.

3.1 Clone Detection

Clone detection is preceded by a syntactic and semantic analysis of the source code performed in the *Columbus* reverse engineering environment [17, 18].

In order to identify code clones, we apply the AST-based approach presented by Koschke et al. [5] implemented on the *Columbus Schema* [19] instance (which is basically an AST – Abstract Syntax Tree decorated with semantic edges). This method finds clone candidates that form syntactic units, i.e. there always exist a root AST node for each clone candidate (called *head node*). Two code parts are considered similar (and therefore fall into the same clone class) if they consist of the same AST node types (represented by the schema) in the same order. The algorithm used here is able to detect type 1 and type 2 clones[2].

[2] Type 1 and 2 clones can differ in whitespace, commentation, identifier names and constant values.

3.2 The Evolution Mapping

The *evolution mapping* is, in our context, a partial injective mapping of the clone instances of version v_1 to a version v_2 of the subject system:

$$e : G \subset CI^{v_1} \rightarrow CI^{v_2}$$

If a clone instance is a map of another instance, it means that the latter one has evolved from the first one. The mapping is *partial* (expressed by set G), as there might be some clone instances that vanished. *Injectivity* means that every clone instance from the newer version has at most one earlier corresponding instance. As the asymptotic number of possible mappings grows exponentially with the number of elements of the sets (due to partiality), it turns out that finding a good one (meaning of goodness will be specified later on) is hard. Not just the location of a clone instance may change, but also its syntactic structure, unique name, etc., which makes really hard to find the corresponding code fragments. Although, we used clone detector for type 1 and type 2 clones, the approach of constructing an evolution mapping does not rely on that; in fact the algorithm works on any two sets of source code fragments. Initially, every pair of instances from two subsequent versions should be considered as a potential pair of the mapping.

To find a good mapping, first we define a *similarity distance function*, which measures how similar the clone instances are. The measure is aggregated from the following elementary features:

F_1 : The lexical structure of the clone instance.
F_2 : Name of the file containing the clone instance.
F_3 : The unique name of the *head node*, or its nearest named ancestor.
F_4 : The relative position of the code segment inside its first named ancestor.

If a developer would need to decide if a code piece has evolved from another, first she would compare the lexical structures of the code fragments (F_1). If they do not differ too much, the developer would next check their location in the code. First, the names of the files would be compared (F_2). If they are two different, the probability that the code fragments are related would be smaller. Then the name of containing class/function would be compared (F_3). The same class/function could easily contain two very similar code pieces and nothing else but their relative position would distinguish them. Therefore, the relative positions of code fragments inside their class/function would be compared (F_4).

After every step the person becomes more or less confident in the final decision. If after any step the differences are already too significant, the developer would reject the possibility that the latter code has evolved from the earlier one. The goal is to use the above formula, to construct a mapping, which is in some sense *optimal*, i.e. it relates as many pairs as possible, but makes mistakes as rarely as possible. We will give a more formal description of optimality later in this section.

To formalize the concepts, let

$$D^* (C_i, C_j) = \sum_{k=1}^{4} \alpha_k D_k (C_i, C_j) \tag{1}$$

be defined for any two different C_i and C_j clone instances, where $D_k(C_i, C_j)$ is a similarity measure for the F_k feature, and α_k is the contribution factor of F_k to the overall composed similarity measure. If $D^*(C_i, C_j)$ is zero, then every $F_k(C_i, C_j)$ is zero for every feature, meaning that C_i and C_j neither differ in lexical structure, nor in location. In this case, we assume that C_j must be a map of C_i. Therefore, considering just the pairs with non-zero distances, we arrive at a so-called *assignment problem* [20], which is a fundamental combinatorial optimization problem. In its general form, the assignment problem is the following:

There are a number of agents and a number of tasks. Any agent can be assigned to perform any task, incurring some cost that may vary depending on the agent-task assignment. It is required that all the tasks be performed by assigning exactly one agent to each task in such a way that the total cost of the assignment is minimized. The number of agents and the number of tasks are equal.

The optimal solution of the above problem can be obtained by a polynomial time and space complexity algorithm called *Hungarian method* [21]. By considering the clone instances of the version v_s as agents, the instances of version v_t as tasks, and the similarity distances as costs, the original problem can be reduced to an assignment problem. If the number of instances are not the same (as it is expected), virtual nodes should be added to the version which has fewer instances. These virtual nodes are unlike every instance in the other version (the distances are infinite). The solution of the assignment problem is a bijection between the agents and tasks (i.e. clone instances), but a partial injective mapping would be needed. Partiality needs to be enforced, otherwise it could happen that an instance would be mapped onto a very dissimilar (i.e. distant) one, just because "something needs to be assigned to everything". To resolve this issue a threshold value β is introduced which serves to prevent mappings between instances being too dissimilar. In essence, the β value makes the mapping rather partial than bad. Thus equation 1 now has the following form:

$$D(C_i, C_j) = \begin{cases} D^*(C_i, C_j), & \text{if } D^*(C_i, C_j) \leq \beta \\ \infty, & \text{otherwise} \end{cases} \tag{2}$$

After the optimal mapping has been found, the edges having an infinite weight are deleted (it affects the pairs for which the similarity distance exceeds β and those in which one of the nodes is virtual). The remaining edges make up the optimal evolution mapping between the two versions.

When applying this procedure, two essential questions arise:

1. How should the similarity distance functions for the separate features be defined?
2. How should the weights and the cutting threshold value be determined?

Similarity distance functions. As the features F_1, F_2 and F_3 operate on lexical (string) values, we employed a modified *Levenshtein distance* [22] (also known as the *edit distance*) to measure their distances. In our case the distance between two strings is their edit distance divided by the length of the longer one. In this way we obtain a distance function for each of the above features lying between 0 and 1.

In the case of F_4 a different approach is required. This feature is responsible for measuring the displacement of the instances within their class or function. Let $L_{BI}(C_i)$ and $L_{IE}(C_i)$ be the number of AST nodes in the class/function before and after the clone instance respectively. Furthermore, let $L_I(C_i)$ be the length of the clone instance, measured by the number of AST nodes. Let

$$D_2\left(C_i, C_j\right) = \sqrt{\left(\frac{L_{BI}\left(C_i\right)}{L_{BI}\left(C_j\right)} - L_{ij}\right)^2 + \left(\frac{L_{IE}\left(C_i\right)}{L_{IE}\left(C_j\right)} - L_{ij}\right)^2}$$

where

$$L_{ij} = \left(L_{BI}(C_i) + L_I(C_i) + L_{IE}(C_i)\right) / \left(L_{BI}(C_j) + L_I(C_j) + L_{IE}(C_j)\right).$$

In this way D_2 measures the difference of the ratio of AST nodes before and after the clone instances.

Weights and threshold. As we would like to have an optimal mapping we need to define and to measure somehow the goodness of a particular mapping. A candidate mapping is good if it relates as many pairs as possible, while the probability of making mistakes (i.e. amount of false positives) is as low as possible at the same time. The number of pairs is measured by the number of edges of the mapping, while the overall similarity distance (every edge has a similarity distance value which contributes to the overall similarity distance of the mapping) must strictly correlate with the amount of false positives (if the overall weight is high – there are more pairs which are distant from each other, but they are still related – the probability of having false positives is higher).

For goodness, we decided to take a weighted ratio of these two numbers, formally:

$$\mathcal{C}\left(\boldsymbol{\alpha}, \beta\right) = \left(\sum_{C_i \in CI^{v_1}, C_j = e(C_i)} D\left(C_i, C_j\right)\right)^n / \|e\left(CI^{v_1}\right)\|$$

where $n > 0$, C_j is the map of C_i, $\|e\left(CI^{v_1}\right)\|$ is the number of edges of the mapping and the sum in the numerator expresses the overall weight of the edges involved in the evolution mapping. The smaller the $\mathcal{C}\left(\boldsymbol{\alpha}, \beta\right)$ function value is, the better the mapping is – at least with our interpretation of goodness. Simply taking the sum of the distances would not be enough, as the trivial mapping with no edges would outperform any other (this is possible as the mapping might be partial). The parameter n is used to express the importance of a similarity distance compared to the number of edges. The higher values of n make small changes of the similarity distance less significant compared to the number of edges. This is important for smaller values of n as the optimization algorithm may drop edges even for a small gain in distance. By increasing n, this should happen less frequently. Thus the proper value of this parameter needs to be determined empirically. Furthermore, we may assume that $\sum \alpha_i^2 = 1$, otherwise dividing all the α_i and β values by $\sum \alpha_i^2$ would result the some mapping. Without this assumption $\boldsymbol{\alpha}$ would tend to $\mathbf{0}$ making all the distances smaller and everything similar to everything.

Now, as the goodness of a mapping with respect to the weights can be measured, the need to apply an optimization algorithm naturally arises. The cost function $\mathcal{C}\left(\boldsymbol{\alpha}, \beta\right)$ is a step-function (not differentiable, not even continuous at every point of its domain), hence to find its minimum value, probably the simplest solution is to apply a simulated annealing algorithm [23]. At each step of the iteration, the α_i and β values are independently varied by choosing random values of a normal distribution, and the goodness of the resulting mapping is computed. Initially we assume that each feature is equally important, i.e. $\alpha_i = \frac{1}{2} = 0.5$ (as $\sum \alpha_i^2 = 1$). In the beginning β was also set to this value. The optimization was performed on 36 versions of *Mozilla Firefox*.

Initially, the overall similarity distance was equal to 141.31 and there were $34,747$ edges between the consecutive versions. After the optimization, $34,635$ edges remained, while the overall similarity distance dropped to 4.87 (the sum of squares of the weights was necessarily equal to one throughout the whole process). As the overall dissimilarity is minimized in this way, the probability of having false edges (edges between instances which are not evolutionary related) is also smaller.

Table 1. Initial and optimized weights of the model

Weights	Initial	Optimized
α_1	0.5	0.74
α_2	0.5	0.21
α_3	0.5	0.37
α_4	0.5	0.52
β	0.5	0.16

Table 1 shows the initial values of the weights and the values we got by applying this optimization procedure. The optimal weights lead to an evolution mapping e which will be referred to as the *optimal evolution mapping* in the following sections.

Evolution of clone classes. The optimal evolution mapping can naturally be extended to the level of clone classes: if more than the half of the instances of a clone class are mapped onto more than half of the instances of the other class, then the mapping can be extended to these two classes. It is easy to show that in this way one will necessarily have a well defined, partial and injective mapping for the clone classes.

4 Clone Smells

In the following, we will systematically summarize all of the possible evolution patterns and give a brief description of their meaning and consequences. Let us suppose that $v_1, v_2, \ldots, v_n$ are consecutive versions of the same software system.

Disappearing clone class (DCC)
Definition: if $e\left(CC_j^{v_{i-1}}\right) = \emptyset$ for some $CC_j^{v_{i-1}} \in CC^{v_{i-1}}$ clone class, i.e. the evolution mapping was unable to find a correspondence with any clone class from version v_i, then $CC_j^{v_{i-1}}$ is said to be a *disappearing clone class*. The most likely reason is that the clone instances of the class had changed inconsistently, making the clone class disappear.

Appearing clone class (ACC)
Definition: if $e^{-1}\left(CC_j^{v_i}\right) = \emptyset$ for some $CC_j^{v_i} \in CC^{v_i}$ clone class, i.e. the clone class is not a map of any other clone class from version v_{i-1}, then $CC_j^{v_i}$ is said to be an *appearing clone class*. It is very probable that the developers created a new type of duplication.

The next smells apply to just those clone instances which are not members of clone classes that had already been reported as **DCC** or **ACC** smells. This means that from now on it may be assumed that both the $e^{-1}\left(CC_j^{v_i}\right) \neq \emptyset$ and $e\left(CC_j^{v_i}\right) \neq \emptyset$ relations hold for the classes of the particular instances.

Disappearing clone instance (DCI)
Definition: $C_k \in CI^{v_{i-1}}$ is a *disappearing clone instance* if $e\left(C_k\right) = \emptyset$, i.e. C_k has not been mapped to any instance of the subsequent version. It means, that a clone instance had been modified, but at least two other copies still remained the same, as the developer might have forgotten for the other instances.

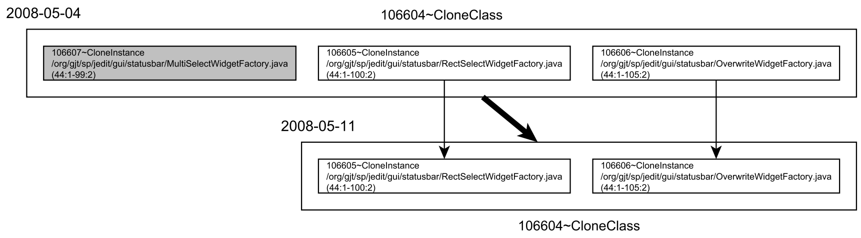

2008-05-04
106604~CloneClass
106607~CloneInstance
/org/gjt/sp/jedit/gui/statusbar/MultiSelectWidgetFactory.java
(44:1-99:2)
106605~CloneInstance
/org/gjt/sp/jedit/gui/statusbar/RectSelectWidgetFactory.java
(44:1-100:2)
106606~CloneInstance
/org/gjt/sp/jedit/gui/statusbar/OverwriteWidgetFactory.java
(44:1-105:2)
2008-05-11
106605~CloneInstance
/org/gjt/sp/jedit/gui/statusbar/RectSelectWidgetFactory.java
(44:1-100:2)
106606~CloneInstance
/org/gjt/sp/jedit/gui/statusbar/OverwriteWidgetFactory.java
(44:1-105:2)
106604~CloneClass

Fig. 1. Example of a DCI smell in the *jEdit* system

Figure 1 illustrates an example of a DCI smell which was found in the version of May 4^{th}, 2008 in the *jEdit* system. Please note, that the clone classes in the diagram are mapped onto each other. In this case, the clone instance colored in gray has disappeared as an execution branch possibly resulting in a NullPointerException has been fixed. The same bugfix was applied for the other two instances just two weeks later.

Appearing clone instance (ACI)
Definition: $C_k \in CI^{v_i}$ is an *appearing clone instance* if $e^{-1}\left(C_k\right) = \emptyset$, i.e. C_k is not a map of any instance from the previous version. It means that new duplication of an already copied code was created.

From this point on we will just consider those clone instances for which none of the above smells have yet been reported. Hence we shall assume that the relations $e^{-1}\left(C_k^{v_i}\right) \neq \emptyset$ and $e\left(C_k^{v_i}\right) \neq \emptyset$ both hold.

Moving clone instance (MCI)
Definition: Let $C_k \in CI^{v_i} \cap CC_j^{v_i}$ be a clone instance in version v_i. C_k is said to be a *moving clone instance* if $e^{-1}\left(C_k\right) \notin e^{-1}\left(CC_j^{v_i}\right)$, i.e. C_k comes from a different clone class as the other instances. It usually means that the developers had modified the clone instance in such a way that it became a clone instance of another class, while the other instances remained in the same clone class.

These five smells cover all the basic cases when just two consecutive versions of the system are considered.

5 Evaluation

We checked the usefulness of clone smells on the *jEdit* open source text editor [16] system, written in Java programming language. The clone detection was performed starting with the year 2008, taking weekly snapshots of the code. The evolution mapping over the 84 consecutive versions was computed by using the weights obtained in the optimization step (see Section 3.2). Table 2 lists the ranges in which the basic metric values of the system varied. (lLOC – logical Lines Of Code means the number of non-empty non-comment lines of code.)

Table 2. Basic metric values of jEdit

Metrics	jEdit
lLOC(logical Lines Of Code)	99,994 - 106,196
NCL (Number Of Classes)	905 - 979
CI (Clone Instances)	349 - 376
CCL (Clone Classes)	125 - 135
CC (Clone Coverage)	4.9% - 10%

The detected clone smells were manually evaluated in order to see if they are useful for finding suspicious evolution patterns. We tried to answer the following questions:

- Are the reported smells really duplications?
- Are the reported smells really clone smells of a particulary type?
- What is the root cause of their existence?

Table 3 summarizes the main results of our evaluation. As can be seen 50 smells were reported in 84 versions of *jEdit*. The most frequent smells are the **DCC** and the **ACC** smells. The table also shows that more than 80% of the hits were really smells (relative to the precision of the clone detector).

Table 3. Number of clone smells

	Hit	Clones?	Smells?
DCC	21	19 (90%)	17 (89%)
ACC	29	24 (83%)	22 (92%)
DCI	1	1 (100%)	1 (100%)
ACI	1	0	0
MCI	0	0	0
Overall	50	44 (88%)	40 (91%)

Table 4. Amount of clone smells found in *jEdit*

	Cause	DCC	ACC	DCI	Σ
Consistent changes	**C1:** Instances deleted	1			**1**
	C2: Intentional refactoring	3			**3**
	C3: Instances are newly created		10		**10**
	Σ	**4**	**10**		**14**
Inconsistent changes	**C4:** Some instances deleted	1			**1**
	C5: Inconsistent changes	12	8	1	**21**
	C6: New instances added		4		**4**
	Σ	**13**	**12**	**1**	**26**
Σ		**17**	**22**	**1**	**40**

Table 4 categorizes the clone smells found in *jEdit*. The categories are defined based on the types of the causes that resulted in reporting the particular smell. The first main category is the category of *Consistent changes* which includes 3 subcategories (**C1-C3**). 14 out of 40 evaluated smells fell into the *Consistent changes* group. The other main category is the category of *Inconsistent changes*, where not all the instances of a clone class are affected in the same way by the changes. In this category all the enumerated cases are suspicious, as they might have occurred because the developers were unaware of the existence of duplications. Most of the smells are caused by inconsistent changes (**C5**), which indicates unawareness of developers. To summarize the findings of experiment:

– More than the half (65%) of the reported smells refer to inconsistent code changes.
– **Inconsistent changes (C5)** are the major cause of the reported smells.
– Among inconsistent changes **DCC** is approximately as much frequent as **ACC**.

Although, the main goal of the smells is not in detecting coding issues, inconsistently changing duplications may uncover unintentionally leaved serious coding problems as well. Figure 2 illustrates an example. First, a bugfix eliminating the possibility of a NullPointerException was applied to the instance in the upper right corner. As there had been two other copies of it, which remained untouched, a **DCI** smell was reported. Two weeks later the same bugfix was applied to the other two instances, but in a syntactically different way which resulted the class to disappear (**DCC**) and new class appeared at the same time (**ACC**).

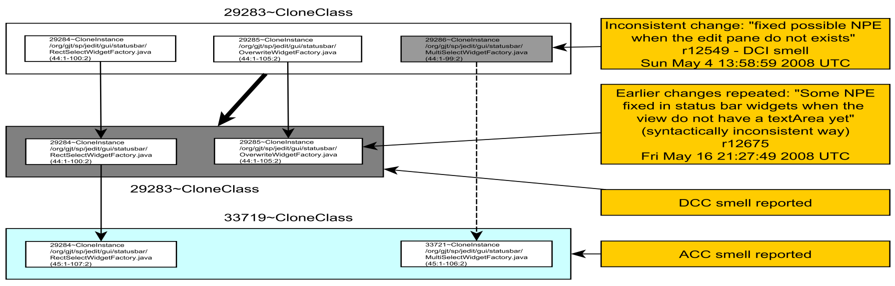

29283~CloneClass
29284~CloneInstance /org/gjt/sp/jedit/gui/statusbar/ RectSelectWidgetFactory.java (44:1-100:2)
29285~CloneInstance /org/gjt/sp/jedit/gui/statusbar/ OverwriteWidgetFactory.java (44:1-105:2)
29286~CloneInstance /org/gjt/sp/jedit/gui/statusbar/ MultiSelectWidgetFactory.java (44:1-99:2)
Inconsistent change: "fixed possible NPE when the edit pane do not exists" r12549 - DCI smell Sun May 4 13:58:59 2008 UTC
29284~CloneInstance /org/gjt/sp/jedit/gui/statusbar/ RectSelectWidgetFactory.java (44:1-100:2)
29285~CloneInstance /org/gjt/sp/jedit/gui/statusbar/ OverwriteWidgetFactory.java (44:1-105:2)
Earlier changes repeated: "Some NPE fixed in status bar widgets when the view do not have a textArea yet" (syntactically inconsistent way) r12675 Fri May 16 21:27:49 2008 UTC
29283~CloneClass
DCC smell reported
33719~CloneClass
29284~CloneInstance /org/gjt/sp/jedit/gui/statusbar/ RectSelectWidgetFactory.java (45:1-107:2)
33721~CloneInstance /org/gjt/sp/jedit/gui/statusbar/ MultiSelectWidgetFactory.java (45:1-106:2)
ACC smell reported

Fig. 2. Inconsistent changes uncover the possibility of a remaining NullPointerException

6 Conclusions and Future Work

In this paper we presented an approach for tracking clone instances across several consecutive versions of a system. We showed that by utilizing an appropriate optimization algorithm the code segments which have evolved from each other can be found. Our evaluation suggests that clone smells can be useful during software development because of the following observations:

1. The approach results in a comparatively short list of critical code segments.
2. The optimal evolution mapping yields a relatively high precision for clone smells.
3. More than the half of the reported smells is caused by inconsistent code changes.
4. Inconsistent changes could uncover unintentionally remaining bugs in the code.

In the approach presented here, we have not used any information originating from the underlying version control system (code repository). In the future we intend to make use of the configuration management as well. In this way, the additionally available pieces of process-related information about the source code should help improve the evolution mapping procedure.

References

1. Mayrand, J., Leblanc, C., Merlo, E.: Experiment on the automatic detection of function clones in a software system using metrics. In: Proceedings of the 1996 International Conference on Software Maintenance, ICSM 1996, p. 244. IEEE Computer Society, Washington (1996)
2. Kapser, C., Godfrey, M.W.: cloning considered harmful" considered harmful. In: Proceedings of the 13th Working Conference on Reverse Engineering, WCRE 2006, pp. 19–28. IEEE Computer Society, Washington (2006)
3. Rahman, F., Bird, C., Devanbu, P.: Clones: What is that smell? In: 7th IEEE Working Conference on Mining Software Repositories (MSR), pp. 72–81 (2010)
4. Kamiya, T., Kusumoto, S., Inoue, K.: CCFinder: a multilinguistic token-based code clone detection system for large scale source code. IEEE Transactions on Software Engineering 28, 654–670 (2002)
5. Koschke, R., Falke, R., Frenzel, P.: Clone detection using abstract syntax suffix trees. In: Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006), Washington, DC, USA, pp. 253–262. IEEE Computer Society, Los Alamitos (2006)
6. Lague, B., Proulx, D., Mayrand, J., Merlo, E.M., Hudepohl, J.: Assessing the benefits of incorporating function clone detection in a development process. In: Proceedings of the 13th International Conference on Software Maintenance (ICSM 1997), Washington, DC, USA, p. 314. IEEE Computer Society, Los Alamitos (1997)
7. Antoniol, G., Casazza, G., Di Penta, M., Merlo, E.: Modeling clones evolution through time series. In: Proceedings of the 17th International Conference on Software Maintenance (ICSM 2001), pp. 273–280. IEEE Computer Society, Los Alamitos (2001)
8. Duala-Ekoko, E., Robillard, M.P.: Tracking code clones in evolving software. In: ICSE 2007: Proceedings of the 29th International Conference on Software Engineering, Washington, DC, USA, pp. 158–167. IEEE Computer Society, Los Alamitos (2007)
9. Kim, M., Sazawal, V., Notkin, D., Murphy, G.: An empirical study of code clone genealogies. SIGSOFT Software Engineering Notes 30, 187–196 (2005)
10. Göde, N., Koschke, R.: Incremental clone detection. In: European Conference on Software Maintenance and Reengineering, pp. 219–228 (2009)
11. Thummalapenta, S., Cerulo, L., Aversano, L., Di Penta, M.: An empirical study on the maintenance of source code clones. Empirical Softw. Engg. 15, 1–34 (2010)
12. Krinke, J.: A study of consistent and inconsistent changes to code clones. In: Proceedings of the 14th Working Conference on Reverse Engineering, WCRE 2007, pp. 170–178. IEEE Computer Society, Washington (2007)
13. Canfora, G., Cerulo, L., Di Penta, M.: Identifying changed source code lines from version repositories. In: Proceedings of the Fourth International Workshop on Mining Software Repositories, MSR 2007, IEEE Computer Society, Washington (2007)
14. Bakota, T., Ferenc, R., Gyimothy, T.: Clone smells in software evolution. In: Proceedings of the 23rd International Conference on Software Maintenance (ICSM 2007), October 2-5, pp. 24–33 (2007)
15. Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional, Reading (1999)
16. jEdit Homepage, http://www.jedit.org
17. Ferenc, R., Beszédes, Á., Tarkiainen, M., Gyimóthy, T.: Columbus – Reverse Engineering Tool and Schema for C++. In: Proceedings of the 18th International Conference on Software Maintenance (ICSM 2002), pp. 172–181. IEEE Computer Society, Los Alamitos (2002)
18. FrontEndART Software Ltd, http://www.frontendart.com (2001)

19. Ferenc, R., Beszédes, Á.: Data Exchange with the Columbus Schema for C++. In: Proceedings of the 6th European Conference on Software Maintenance and Reengineering (CSMR 2002), pp. 59–66. IEEE Computer Society, Los Alamitos (2002)
20. Yudin, D., Gol'shtein, E.G.: Linear Programing Problems of Transportation. NAUKA (1969) (in Russian)
21. Kuhn, H.W.: The Hungarian method for the assignment problem. Naval Research Logistic Quarterly 2, 83–97 (1955)
22. Levenshtein distance, `http://en.wikipedia.org/wiki/Levenshtein_distance`
23. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220(4598), 671–680 (1983)

Liquidsoap: A High-Level Programming Language for Multimedia Streaming

David Baelde[1], Romain Beauxis[2], and Samuel Mimram[3]

[1] University of Minnesota, USA
[2] Department of Mathematics, Tulane University, USA
[3] CEA LIST – LMeASI, France

Abstract. Generating multimedia streams, such as in a netradio, is a task which is complex and difficult to adapt to every users' needs. We introduce a novel approach in order to achieve it, based on a dedicated high-level functional programming language, called *Liquidsoap*, for generating, manipulating and broadcasting multimedia streams. Unlike traditional approaches, which are based on configuration files or static graphical interfaces, it also allows the user to build complex and highly customized systems. This language is based on a model for streams and contains operators and constructions, which make it adapted to the generation of streams. The interpreter of the language also ensures many properties concerning the good execution of the stream generation.

The widespread adoption of broadband internet in the last decades has changed a lot our way of producing and consuming information. Classical devices from the analog era, such as television or radio broadcasting devices have been rapidly adapted to the digital world in order to benefit from the new technologies available. While analog devices were mostly based on hardware implementations, their digital counterparts often consist in software implementations, which potentially offers much more flexibility and modularity in their design. However, there is still much progress to be done to unleash this potential in many areas where software implementations remain pretty much as hard-wired as their digital counterparts.

The design of domain specific languages is a powerful way of addressing that challenge. It consists in identifying (or designing) relevant domain-specific abstractions (construct well-behaved objects equipped with enough operations) and make them available through a programming language. The possibility to manipulate rich high-level abstractions by means of a flexible language can often release creativity in unexpected ways. To achieve this, a domain-specific language should follow three fundamental principles. It should be

1. *adapted*: users should be able to perform the required tasks in the domain of application of the language;
2. *simple*: users should be able to perform the tasks in a simple way (this means that the language should be reasonably concise, but also understandable by users who might not be programming language experts);

3. *safe*: the language should perform automatic checks to prevent as many errors as possible, using static analysis when possible.

Balancing those requirements can be very difficult. This is perhaps the reason why domain specific languages are not seen more often. Another reason is that advanced concepts from the theory of programming language and type systems are often required to obtain a satisfying design.

In this paper, we are specifically interested in the generation of multimedia streams, notably containing audio and video. Our primary targets are netradios, which continuously broadcast audio streams to listeners over Internet. At first, generating such a stream might seem to simply consist in concatenating audio files. In practice, the needs of radio makers are much higher than this. For instance, a radio stream will often contain commercial or informative jingles, which may be scheduled at regular intervals, sometimes in between songs and sometimes mixed on top of them. Also, a radio program may be composed of various automatic playlists depending on the time of the day. Many radios also have live shows, based on a pre-established schedule or not; a good radio software is also expected to interrupt a live show when it becomes silent. Most radios want to control and process the data before broadcasting it to the public, performing tasks like volume normalization, compression, etc. Those examples, among many others, show the need for very flexible and modular solutions for creating and broadcasting multimedia data. Most of the currently available tools to broadcast multimedia data over the Internet (such as Darkice, Ezstream, VideoLAN, Rivendell or SAM Broadcaster) consist of straightforward adaptation of classical streaming technologies, based on predefined interfaces, such as a virtual mixing console or static file-based setups. Those tools perform very well a predefined task, but offer little flexibility and are hard to adapt to new situations.

In this paper, we present Liquidsoap, a domain-specific language for multimedia streaming. Liquidsoap has established itself as one of the major tools for audio stream generation. The language approach has proved successful: beyond the obvious goal of allowing the flexible combination of common features, unsuspected possibilities have often been revealed through clever scripts. Finally, the modular design of Liquidsoap has helped its development and maintenance, enabling the introduction of several exclusive features over time. Liquidsoap has been developed since 2004 as part of the Savonet project [2]. It is implemented in OCaml, and we have thus also worked on interfacing many C libraries for OCaml. The code contains approximatively 20K lines of OCaml code and 10K lines of C code and runs on all major operating systems. The software along with its documentation is freely available [2] under an open-source license.

Instead of concentrating on detailing fully the abstractions manipulated in Liquidsoap (streams and sources) or formally presenting the language and its type system, this paper provides an overview of the two, focusing on some key aspects of their integration. We first give a broad overview of the language and its underlying model in Section 1. We then describe two recent extensions of that basic setup. In Section 2 we illustrate how various type system features are combined to control the combination of stream of various content types. Section 3 motivates the interest of having multiple time flows (clocks) in a streaming system, and presents how this feature is integrated in Liquidsoap. We finally discuss related systems in Section 4 before concluding in Section 5.

1 Liquidsoap

1.1 Streaming Model

A stream can be understood as a timed sequence of data. In digital signal processing, it will simply be an infinite sequence of samples – floating point values for audio, images for video. However, multimedia streaming also involves more high-level notions. A *stream*, in Liquidsoap, is a succession of *tracks*, annotated with *metadata*. Tracks may be finite or infinite, and can be thought of as individual songs on a musical radio show. Metadata packets are punctual and can occur at any instant in the stream. They are used to store various information about the stream, such as the title or artist of the current track, how loud the track should be played, or any other custom information. Finally, tracks contain multimedia data (audio, video or MIDI), which we discuss in Section 2.

Streams are generated on the fly and interactively by *sources*. The behavior of sources may be affected by various parameters, internal (*e.g.*, metadata) or external (*e.g.*, execution of commands made available via a server). Some sources purely produce a stream, getting it from an external device (such as a file, a sound card or network) or are synthesizing it. Many other sources are actually operating on other sources in the sense that they produce a stream based on input streams given by other sources. Abstractly, the program describing the generation of a stream can thus be represented by a directed acyclic graph, whose nodes are the sources and whose edges indicate dependencies between sources (an example is given in Figure 1).

Some sources have a particular status: not only do they compute a stream like any other source, but they also perform some observable tasks, typically outputting their stream somewhere. These are called *active* sources. Stream generation is performed "on demand": active sources actively attempt to produce their stream, obtaining data from their input sources which in turn obtain data from their dependent sources, and so on. An important consequence of this is the fact that *sources do not constantly stream*: if a source would produce a stream which is not needed by any active source then it is actually frozen in time. This avoids useless computations, but is also crucial to obtain the expected expressiveness. For example, a `rotation` operator will play alternatively several sources, but should only rotate at the end of tracks, and its unused sources should not keep streaming, otherwise we might find them in the middle of a track when we come back to playing them. *Sources are also allowed to fail* after the end of a track, *i.e.*, refuse to stream, momentarily or not. This is needed, for example, for a queue of user requests which might often be empty, or a playlist which may take too long to prepare a file for streaming. Failure is handled by various operators, the most common being the *fallback*, which takes a list of sources and replays the stream of the first available source, failing when all of them failed.

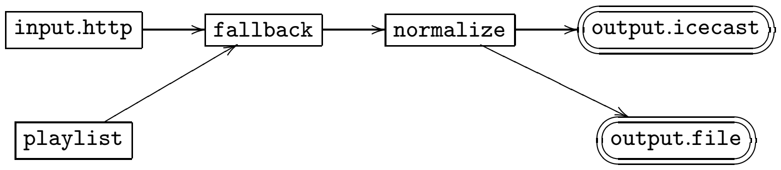


Fig. 1. A streaming system with sharing

1.2 A Language for Building Streaming Systems

Based on the streaming model presented above, Liquidsoap provides the user with a convenient high-level language for describing streaming systems. Although our language borrows from other functional programming languages, it is has been designed from scratch in order to be able to have a dedicated static typing discipline together a very user-friendly language.

One of the main goals which has motivated the design of the Liquidsoap language is that it should be very accessible, even to non-programmers. It turned out that having a functional programming language is very natural (cf. Section 1.3). The built-in functions of the language often have a large number of parameters, many of which have reasonable default values, and it would be very cumbersome to have to write them all each time, in the right order. In order to address this, we have designed a new extension of λ-calculus with labeled arguments and multi-abstractions which makes it comfortable to use the scripting API [3]. Having designed our own language also allowed us to integrate a few domain-specific extensions, to display helpful error messages and to generate a browsable documentation of the scripting API. In practice, many of the users of Liquidsoap are motivated by creating a radio and not very familiar with programming, so it can be considered that the design of the language was a success from this point of view.

An other motivation was to ensure some safety properties of the stream generation. A script in Liquidsoap describes a system that is intended to run for months, some parts of whose rarely triggered, and it would be very disappointing to notice a typo or a basic type error only after a particular part of the code is used for an important event. In order to ensure essential safety properties, the language is statically and strongly typed. We want to put as much static analysis as possible, as long as it doesn't put the burden on the user, *i.e.*, all types should be inferred. As we shall see, Liquidsoap also provides a few useful dynamic analysis.

The current paper can be read without a prior understanding of the language and its typing system, a detailed presentation can however be found in [3]. A basic knowledge of programming languages should be enough to understand the few examples presented in this paper, which construct sources using built-in operators of our language. For example, the following script defines two elementary sources, respectively reading from an HTTP stream and a playlist of files, composed in a fallback and filtered through a volume normalizer. The resulting stream is sent to an Icecast server which broadcasts the stream to listeners, and saved in a local backup file:

```
s = normalize(fallback([input.http("http://other.net/radio"),
                        playlist("listing.txt")]))
output.icecast(%vorbis,mount="myradio",s)
output.file(%vorbis,"backup.mp3",s)
```

The graph underlying the system resulting from the execution of that script is shown in Figure 1. Note that the two output functions build a new source: these sources generate the same stream as `s`, but are active and have the side effect of encoding and sending the stream, respectively to an Icecast server and a file. A few remarks on the syntax: the notation `[...]` denotes a list, `mount` is a label (the name of an argument of the function `output.icecast`) and `%vorbis` is an encoding format parameter whose meaning is explained in Section 2 (recall that Vorbis is a compressed format for audio, similar to MP3).

1.3 Functional Transitions

Liquidsoap is a *functional* programming language and a particularly interesting application of this is the case of *transitions*. Instead of simply sequencing tracks, one may want a smoother transition. For example, a *crossfade* consists in mixing the end of the old source, whose volume is faded out, with the beginning of the new one, whose volume is faded up (see Figure 2). But there is a wide variety of other possible transitions: a delay might be added, jingles may be inserted, etc.

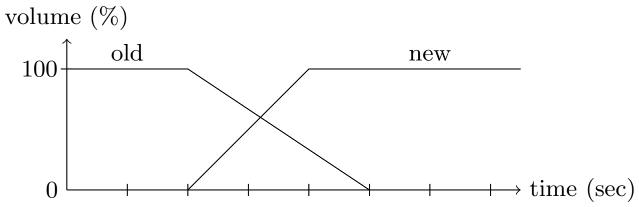


Fig. 2. A crossfade transition between two tracks

A solution that is both simple and flexible is to allow the user to specify a transition as a function that combines two sources representing the old and new tracks. We illustrate this feature with an example involving transitions used when switching from one source to another in a `fallback`. This is particularly useful when the fallback is track insensitive, *i.e.*, performs switching as soon as possible, without waiting for the end of a track. The following code defines a fallback source which performs a crossfade when switching from one source to another:

```
def crossfade(old,new) =
  add([fade.initial(duration=2.,new),fade.final(duration=3.,old)])
end
t = [crossfade,crossfade]
f = fallback(track_sensitive=false,transitions=t,[r,s])
```

Because any function can be used to define a transition, the possibilities are numerous. A striking example from the standard library of Liquidsoap scripts is the operator `smooth_add`, which takes as argument a main source (*e.g.*, musical tracks) and a special interruption source (*e.g.*, news items). When a new interruption is available, `smooth_add` gradually reduces the volume of the main source to put it in the background, and superposes the interruption. The reverse is performed at the end of the interruption. This very appreciated effect is programmed using the same building blocks as in the previous example.

1.4 Efficient Implementation

An important aspect of the implementation is efficiency concerning both CPU and memory usage. The streams manipulated can have high data rates (a typical video stream needs 30Mo/s) and avoiding multiple copies of stream data is crucial.

In Liquidsoap, streams are computed using *frames*, which are data buffers representing a portion of stream portion of fixed duration. Abstractly, sources produce a stream by producing a sequence of frames. However, in the implementation a source is *passed* a frame that it has to fill. Thus, we avoid unnecessary copies and memory allocations. Active sources, which are the initiators of streaming, initially allocate one frame, and keep re-using it to get stream data from their input source. Then, most sources do not need to allocate their own frame, they simply pass frames along and modify their content in place. However, this simple mechanism does not work when a source is *shared*, *i.e.*, it is the input source of several sources. This is the case of the `normalize` node in the graph of Figure 1 (which happens to be shared by active sources). In that case, we use a *caching* mechanism: the source will have its own *cache* frame for storing its current output. The first time that the source is asked to fill a frame, it fills its internal cache and copies the data from it; in subsequent calls it simply fills the frame with the data computed during the first call. Once all the filling operations have been done, the sources are informed that the stream has moved on to the next frame and can forget their cache.

With this system, frames are created once for all, one for each active source plus one for each source that is shared and should thus perform caching — of course, some sources might also need another frame depending on their behavior. Sharing is detected automatically when the source is initialized for streaming. We do not detail this analysis, but note that the dynamic reconfigurations of the streaming system (notably caused by transitions) make it non-trivial to anticipate all possible sharing situations without over-approximating too much.

2 Heterogeneous Stream Contents

In Liquidsoap, streams can contain data of various nature. The typical example is the case of video streams which usually contain both images and audio samples. We also support MIDI streams (which contain musical notes) and it would be easy to add other kinds of content. It is desirable to allow sources of different content kinds within a streaming system, which makes it necessary to introduce a typing discipline in order to ensure the consistency of stream contents across sources.

The nature of data in streams is described by its *content type*, which is a triple of natural numbers indicating the number of audio, video and midi channels. A stream may not always contain data of the same type. For instance, the `playlist` operator might rely on decoding files of heterogeneous content, *e.g.*, mono and stereo audio files. In order to specify how content types are allowed to change over time in a stream, we use *arities*, which are essentially natural numbers extended with a special symbol $\star$:

$$a \quad ::= \quad \star \mid 0 \mid S(a)$$

An arity is *variable* if it contains $\star$, otherwise it is an usual natural number, and is *fixed*. A *content kind* is a triple of arities, and specifies which content types are acceptable. For example, $(S(S(0)), S(\star), \star)$ is the content kind meaning “2 audio channels, at least one video channel and any number of MIDI channels”. This is formalized through the subtyping relation defined in Figure 3: $T <: K$ means that the content kind T is allowed by K. More generally, $K <: K'$ expresses that K is more permissive than K', which implies that a source of content kind K can safely be seen as one of content kind K'.

$$\frac{}{0 <: 0} \qquad \frac{A <: A'}{S(A) <: S(A')} \qquad \frac{}{\star <: \star} \qquad \frac{}{0 <: \star} \qquad \frac{A <: \star}{S(A) <: \star}$$

$$\frac{A <: A' \quad B <: B' \quad C <: C'}{(A, B, C) <: (A', B', C')}$$

Fig. 3. Subtyping relation on arities

When created, sources are given their expected content kind. Of course, some assignments are invalid. For example, a pure audio source cannot accept a content kind which requires video channels, and many operators cannot produce a stream of an other kind than that of their input source. Also, some sources have to operate on input streams that have a fixed kind – a kind is said to be fixed when all of its components are. This is the case of the `echo` operator which produces echo on sound and has a internal buffer of a fixed format for storing past sound, or sound card inputs/outputs which have to initialize the sound card for a specific number of channels. Also note that passing the expected content kind is important because some sources behave differently depending on their kind, as shown with the previous example.

Integration in the language. To ensure that streaming systems built from user scripts will never encounter situations where a source receives data that it cannot handle, we leverage various features of our type system. By doing so, we guarantee statically that content type mismatches never happen. The content kinds are reflected into types, and used as parameters of the `source` type. In order to express the types of our various operators, we use a couple features of type systems (see [5] for extensive details). As expected, the above subtyping relation is integrated into the subtyping on arbitrary Liquidsoap types. We illustrate various content kinds in the examples of Figure 4:

– The operator `swap` exchanges the two channels of a stereo audio stream. Its type is quite straightforward: it operates on streams with exactly two audio channels.
– Liquidsoap supports polymorphism *à la* ML. We use it in combination with constraints to allow arbitrary arities. The notation `'*a` stands for a universal variable (denoted by `'a`) to which a type constraint is attached, expressing that it should only be instantiated with arities. For example, the operator `on_metadata` does not rely at all on the content of the stream, since it is simply in charge of calling a handler on each of its metadata packets – in the figure, `handler` is a shortcut for `([string*string]) -> unit`.
– When an operator, such as `echo`, requires a fixed content type, we use another type constraint. The resulting constrained universal variable is denoted by `'#a` and can only be instantiated with fixed arities.
– The case of the `greyscale` operator, which converts a color video into greyscale, shows how we can require at least one video channel in types. Here, `'*b+1` is simply a notation for `S('*b)`.
– Finally, the case of `output.file` (as well as several other outputs which encode their data before sending it to various media) is quite interesting. Here, the expected content kind depends on the format the stream is being encoded to, which is given as first argument of the operator. Since typing the functions generating formats would require dependent types (the number of channels would be given as argument) and break type inference, we have introduced particular constants for type formats with syntactic sugar for them to appear like functions – similar ideas are for example used to type the `printf` function in OCaml. For example, `output.file(%vorbis,"stereo.ogg",s)` requires that `s` has type `source(2,0,0)` because `%vorbis` alone has type `format(2,0,0)`, but `output.file(%vorbis(channels=1),"mono.ogg",s)` requires that there is only one audio channel; we also have video formats such as `%theora`.

These advanced features of the type system are statically inferred, which means that the gain in safety does not add any burden on users. As said above, content

```
       swap : (source(2,0,0)) -> source(2,0,0)
on_metadata : (handler,source('*a,'*b,'*c)) -> source('*a,'*b,'*c)
       echo : (delay:float,source('#a,0,0)) -> source('#a,0,0)
  greyscale : (source('*a,'*b+1,'*c)) -> source('*a,'*b+1,'*c)
output.file : (format('*a,'*b,'*c),string,source('*a,'*b,'*c))->
              source('*a,'*b,'*c)
```

Fig. 4. Types for some operators

kinds have an influence on the behavior of sources — polymorphism is said to be *non-parametric*. In practice, this means that static types must be maintained throughout the execution of a script. This rather unusual aspect serves us as an overloading mechanism: the only way to remove content kinds from execution would be to duplicate our current collection of operators with a different one for each possible type instantiation.

Ideally, we would like to add some more properties to be statically checked by typing. But it is sometimes difficult to enrich the type system while keeping a natural syntax and the ability to infer types. For example, Liquidsoap checks that active sources are *infallible*, *i.e.*, always have data available in their input stream, and this check is currently done by a flow analysis on instantiated sources and not typing. Another example is clocks which are described next section.

3 Clocks

Up to now, we have only described streaming systems where there is a unique global clock. In such systems, time flows at the same rate for all sources. By default, this rate corresponds to the wallclock time, which is appropriate for a live broadcast, but it does not need to be so. For example, when producing a file from other files, one might want the time rate to be as fast as the CPU allows.

While having a global clock suffices in many situations, there are a couple of reasons why a streaming system might involve multiple clocks or time flows. The first reason is external to liquidsoap: there is simply not a unique notion of time in the real world. A computer's internal clock indicates a slightly different time than your watch or another computer's clock. Moreover, when communicating with a remote computer, network latency causes a perceived time distortion. Even within a single computer there are several clocks: notably, each soundcard has its own clock, which will tick at a slightly different rate than the main clock of the computer. Since liquidsoap communicates with soundcards and remote computers, it has to take those mismatches into account.

There are also some reasons that are purely internal to liquidsoap: in order to produce a stream at a given speed, a source might need to obtain data from another source at a different rate. This is obvious for an operator that speeds up or slows down audio, but is also needed in more subtle cases such as a crossfading operator. A variant of the operator described in Section 1.3 might combine a portion of the end of a track with the beginning of the next track *of the same source* to create a transition between tracks. During the lapse of time where the operator combines data from an end of track with the beginning of the other other, the crossing operator needs to read both the stream data of the current track and the data of the next track, thus reading twice as much stream data as in normal time. After ten tracks, with a crossing duration of six seconds, one more minute will have passed for the source compared to the time of the crossing operator.

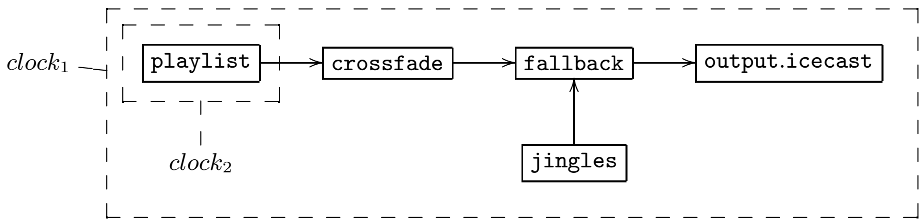


Fig. 5. A streaming system with two clocks

3.1 Model

In order to avoid inconsistencies caused by time differences, while maintaining a simple and efficient execution model for its sources, liquidsoap works under the restriction that one source belongs to a unique clock, fixed once for all when the source is created. Sources from different clocks cannot communicate using the normal streaming protocol, since it is organized around clock cycles: each clock is responsible for animating its own active sources and has full control on how it does it.

In the graphical representation of streaming systems, clocks induce a partition of sources represented by a notion of locality or box, and clock dependencies are represented by nesting. For example, the graph shown in Figure 5 corresponds to the stream generators built by the following script:

```
output.icecast(%vorbis,mount="myradio",
  fallback([crossfade(playlist("some.txt")),jingles]))
```

There, *clock*$_2$ was created specifically for the crossfading operator; the rate of that clock is controlled by that operator, which can hence accelerate it around track changes without any risk of inconsistency. *clock*$_1$ is simply a wallclock, so that the main stream is produced following the real time rate.

A clock is *active* if it ticks by itself, therefore running its sources constantly; this is the case of wallclocks or soundcard clocks. We say that a clock depends on another one if its animation (and thus time rate) depends on it. Active sources do not depend on other sources, and dependencies must be acyclic. In the above example, the ticking of *clock*$_2$ is provoked by that of *clock*$_1$, and freezes when the fallback is playing jingles. Although nothing forces it in the model, it makes more sense if each passive source depends (possibly indirectly) on an active one, and all sources without dependencies are active. Those assumptions are in fact guaranteed to hold for the systems built using the Liquidsoap language.

From an implementation viewpoint, each active clock launches its own streaming thread. Hence, clocks provide a way to split the generation of one or several streams across several threads, and hence multiple CPU cores.This powerful possibility is made available to the user through the intuitive notion of clock. As we shall see in the next section, the script writer never needs to specify clocks unless he explicitly wants a particular setup, and Liquidsoap automatically checks that clock assignements are correct.

3.2 Clock Assignment

Clocks are not represented in the type of Liquidsoap sources. Although it would be nice to statically check clock assignment, type inference would not be possible without technical annotations from the user. Instead, clocks are assigned upon source creation. Some sources require to belong to a particular, definite clock, such as the wallclock, or the clock corresponding to a sound card. Most sources simply require that their clock is the same as their input sources. Since clocks often cannot be inferred bottom-up, we use a notion of clock variable that can be left undefined. Clock variables reflect the required clock dependencies, which are maintained during the inference process.

Two errors can occur during this phase. Although they are runtime errors that could be raised in the middle of streaming when new sources are created (*e.g.*, by means of a transition), this usually only happens during the initial construction. The first error is raised when two different known clocks need to be unified. For example, in the following script, the ALSA input is required to belong to the ALSA clock and `crossfade`'s internal clock at the same time:

```
output.file(%vorbis,"record.ogg",crossfade(input.alsa()))
```

The other possible error happens when unifying two unknown clock variables if one depends on the other – in unification terminology, this is an *occurs-check* failure. A simple example of that situation is the script `add([s,crossfade(s)])` where the two mixed sources respectively have clocks c and X_c where c is the clock created for the crossfading operator and X_c is the variable representing the clock to which the crossfading belongs, on which c depends.

After this inference phase, it is possible that some clocks are still unknown. Remaining variables are thus forcibly assigned to the default wallclock, before that all new sources are prepared for streaming by their respective clocks.

4 Related Work

Liquidsoap is obviously different from classical tools such as Ices or Darkice in that it offers the user the freedom to assemble a stream for a variety of operators, through a scripting language rather than traditional configuration files.

Liquidsoap has more similarities with multimedia streaming libraries and digital signal processing (DSP) languages. The GStreamer library [6] defines a model of stream, and its API can be used to define streaming systems in various programming languages (primarily coded in C, the library has also been ported to many other languages). Faust [4] provides a high-level functional programming language for describing stream processing devices, and compiles this language down to C++, which enables an integration with various other systems. It is also worth mentioning Chuck [7], a DSP programming language with an emphasis on live coding (dynamic code update). Besides a different approach and target application, Liquidsoap differs more deeply from these tools. The notion of source provides a richer way of generating streams, providing and relying on the additional notions of tracks and metadata; also recall the ability to momentarily stop streaming, and the possibility to dynamically create or destroy sources. It would be very interesting to interface Liquidsoap with the above mentionned tools, or import some of their techniques. This could certainly be done for simple operators such as DSP, and would allow us to program them efficiently and declaratively from the scripting language rather than in OCaml.

5 Conclusion

We have presented the main ideas behind the design of Liquidsoap, a tool used by many netradios worldwide as well as in some academic work [1]. We believe that Liquidsoap demonstrates the potential of building applications as domain-specific languages. It also shows that very rich type systems can be put to work usefully even in tools not designed for programmers: although most Liquidsoap users have a limited understanding of our type system, they are able to fix their mistakes when an error is reported — errors might be difficult to read but they have the merit of signaling real problems.

Of course, there are many other reasons behind the success of Liquidsoap, including a wide variety of features plugged onto the basic organization described here. Some of the future work on Liquidsoap lies there: integration with other tools, graphical interfaces, documentation, etc. But we are also planning some improvements of the language. One of the goals is to make it possible to express more operators directly in Liquidsoap instead of OCaml, bringing more customizability to the users. Also, Liquidsoap offers a server through which many sources offer various services. An interesting way to structure more this very useful system would be to consider sources as objects whose methods are services, and type them accordingly.

References

1. Baccigalupo, C., Plaza, E.: A case-based song scheduler for group customised radio. In: Weber, R.O., Richter, M.M. (eds.) ICCBR 2007. LNCS (LNAI), vol. 4626, pp. 433–448. Springer, Heidelberg (2007)
2. Baelde, D., Beauxis, R., Mimram, S., et al.: Liquidsoap, `http://savonet.sf.net/`
3. Baelde, D., Mimram, S.: De la webradio lambda à la λ-webradio. In: Journées Francophones des Langages Applicatifs (JFLA 2008), pp. 47–62 (2008)
4. Orlarey, Y., Fober, D., Letz, S.: FAUST: an Efficient Functional Approach to DSP Programming. In: New Computational Paradigms for Computer Music (2009)
5. Pierce, B.C.: Types and Programming Languages. MIT Press, Cambridge (2002)
6. Taymans, W., Baker, S., Wingo, A., Bultje, R., Kost, S.: GStreamer application development manual
7. Wang, G., Cook, P., et al.: ChucK: A concurrent, on-the-fly audio programming language. In: Proc. of Int. Comp. Music Conf., pp. 219–226 (2003)

Combining Traditional Map Labeling with Boundary Labeling[⋆]

Michael A. Bekos[1], Michael Kaufmann[2], Dimitrios Papadopoulos[1], and Antonios Symvonis[1]

[1] School of Applied Mathematical & Physical Sciences,
National Technical University of Athens, Greece
{mikebekos,dpapadopoulos,symvonis}@math.ntua.gr

[2] University of Tübingen, Institute for Informatics, Germany
mk@informatik.uni-tuebingen.de

Abstract. The traditional map labeling problems are mostly $\mathcal{NP}$-hard. Hence, effective heuristics and approximations have been developed in the past. Recently, efficient algorithms for the so-called boundary labeling model have been introduced which assumes that the labels are placed on the boundary of the map and connected by polygonal leaders to their corresponding sites. Internal labels have been forbidden. In this paper, we allow both. Since clearly internal labels should be preferred, we consider several maximization problems for the number of internal labels and we show that they can be obtained efficiently or in quasi-polynomial time.

1 Motivation

In information visualization, geographic information systems, and cartography, map labeling constitutes an important task, which is concerned with the placement of extra information –in the form of textual labels– next to features of interest of an illustration. In order to ensure readability, it is suggested that the labels: i) do not overlap with each other, and ii) are close to (if possible, next to) the features they are associated with. Unfortunately, due to these constraints, the majority of the map labeling problems turns out to be computationally hard [9]. A detailed bibliography on map labeling can be found in [19].

A great amount of research on map labeling has been devoted to labeling site-features of a map. However, in the presence of large labels or dense point sets, producing a labeling with no overlaps is most of the times impossible. To address this problem, a commonly used approach is to place the labels next to the map, and to connect them to their sites by polylines, called *leaders*. This labeling approach is called *boundary labeling*; in the past, several papers have appeared that show that certain types of boundary labeling problems can be solved efficiently in theory and practice [2,3,4,5,6,12,15,16].

⋆ The work is funded by the National Technical University of Athens research program ΠEBE 2008.

In this paper, we introduce and study a *mixed labeling* model, according to which we place the labels next to the sites and use boundary labeling in cases where this is not feasible due to overlaps.

As a first step towards solving this labeling problem, we investigate simple settings of the problem. Our motivation rests on the work of Fowler et al. [10], who proved that the number maximization problem (i.e., the problem of determining the maximum number of labels that can be placed on a set of sites without overlaps) is $\mathcal{NP}$-hard, even if the labels are of uniform size, each site has only one label candidate position and leaders are not permitted. In contrast to the negative result of Fowler et al., there are variants of the proposed model which admit an algorithm that maximizes the number of site-labels and guarantees the placement of all labels (either immediately next to the sites or on the boundary of the underlying drawing through leaders). Our variants, where we give quasi-polynomial time algorithms, suggest that using boundary labeling, the problem becomes easier as no $\mathcal{NP}$-hard problem is known which has a quasi-polynomial algorithm. More sophisticated variants are still to be considered. The proposed model is well suited when labeling technical or medical maps where it is common to explain certain features of the map with label on its boundary, since in the case where the map contains several features to be labeled, a great number of leaders will unavoidably occur, and subsequently clutter the illustration.

This rest of this paper is structured as follows: In Section 2, we formally define the mixed map labeling model. In Sections 3, 4 and 5, we study several variations of the mixed labeling problem. In Section 6, we provide an experimental evaluation of our algorithms. We conclude in Section 7 with open problems.

2 Problem Definition

Typically, a map labeling problem consists of a set $P = \{s_1, s_2, \ldots s_n\}$ of n sites to be labeled. Each site $s_i = (x_i, y_i)$ is associated with an axis-parallel, rectangular label l_i of dimensions $w_i \times h_i$. We also assume that the site set is enclosed in an axis-parallel rectangle $R = [0, W] \times [0, H]$ (*enclosing rectangle*). According to our model, there exist two alternatives to label each site: i) either through a label "close" to the site (*internal labels*), or ii) through a label on the boundary of R and a leader which realizes the connection (*external labels*).

For each site which is to be labeled with an internal label, the input specifies a *model*, which indicates how the site's label can be placed w.r.t. the actual position of the site. The most important models studied in the literature are: i) the *fixed position model*, where each site has a finite set of *label candidates* (i.e., feasible label positions), and ii) the *slider model*, where each label can be placed at any position, so that it touches its site. Fig.1 depicts some commonly used variants of the both models. For a more formal definition refer to [13].

The external labels are usually attached to one, two or all four sides of R. Several types of leaders have been proposed, among them *straight-line* [5], *rectilinear* [5] and *octilinear* [6]. In this paper, we focus on rectilinear leaders of type-*opo*, that consist of three line segments, where the first and third ones are

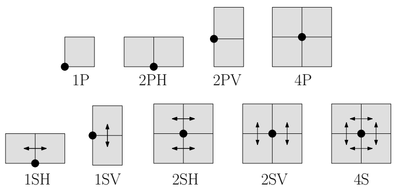


Fig. 1. Each model has an abbreviation of the form xMD, where $M \in \{P, S\}$ stands for fixed-position (P) or slider model (S), $x \in \{1, 2, 4\}$ refers to the number of fixed positions or sliding directions, and $D \in \{\emptyset, H, V\}$ indicates the horizontal or vertical direction in which fixed-position labels are arranged or labels slide

orthogonal (o) to the side of R containing the label, and the second one is parallel (p) to that side. We further assume that the sites are in *general position*, i.e., no two sites share the same y or x coordinate. For more details, refer to [5].

Conventionally, a mixed labeling model (m, k, t) is identified as a type-m traditional labeling model supported by a k-sided boundary labeling model with type-t leaders, where $m \in \{1P, 2PH, 2PV, 4P, 1SH, 1SV, 2SH, 2SV, 4S\}$, $k \in \{\text{Left-Sided}, \text{Right-Sided}, \text{Two-Sided}, \text{Four-Sided}\}$ and $t \in \{s, po, opo, do, od, pd\}$[1], in the case where the internal (external) labels are placed according to the rules of model m (k) and the leaders are of type t. Our intension is to obtain *legal labelings* in which no internal labels overlap and no leader intersect or overlap, and, simultaneously maximize the number of internal labels. We further assume that one of the rectangle's sides can always accommodate all labels.

3 The (1P, Right-Sided, opo) Mixed Labeling Model

We first consider the mixed model (1P, right-sided, *opo*) and present an algorithm which produces labelings with maximum number of internal labels. Our algorithm performs in three steps. Initially, it computes a legal labeling purely consisting of external labels. This is feasible since we have assumed that one of the enclosing rectangle's sides can accommodate all labels. As shown in [5], in a legal solution of the pure boundary labeling problem (i.e., no internal labels are allowed), the vertical order of the sites is identical to the vertical order of their corresponding labels and it can be computed in $O(n \log n)$ time. The leaders are considered to be preliminary and might either be removed or labeled permanent. In the second step, the algorithm identifies sites whose preliminary leaders cannot be replaced by internal labels. Crucial is the following. For two sites with overlapping internal labels, the lower one must be labeled through a leader (see the right part of Figure 2).

Based on this observation, we determine in a second step for each site s, if its inner label $l(s)$ is intersected by a label $l(s')$ such that s' lies above of s. If such a label $l(s')$ exists, we mark the leader of s to be permanent. In the final step, we perform a top-down plane sweep, where we stop at each leader that

[1] Abbreviations s, po, opo, do, od and pd refer to different types of leaders (see [2] and [5]).

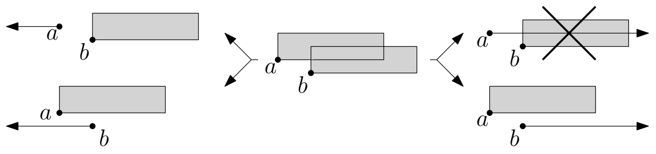


Fig. 2. The internal labels of sites a and b are involved in an overlap. In the case where the leaders are routed to the right only one feasible case exists. However, in the case where the leaders are routed to the left, two feasible solutions exist.

we reach. If the leader, say for site s, is already permanent and it intersects other internal labels $l(s')$, we mark the leaders for s' also to be permanent. If the leader of site s is still preliminary, we know that for site s there is no other intersecting internal label and also no intersecting leader. Hence we can replace it by an internal label.

Theorem 1. *The (*$1P$*, right-sided, opo) problem can be solved in* $O(n \log^2 n)$ *time.*

Sketch of proof. As already stated, the first step of our algorithm needs $O(n \log n)$ time. By employing a dynamic data structure that maintains a set of rectangles (that changes under insertions and deletions) and given a query rectangle q reports whether q is involved in an overlap with other rectangle stored in the data structure in $O(\log^2 n)$ time [8], the second step of our algorithm can be implemented in a total of $O(n \log^2 n)$ time. The third step of our algorithm needs an extra $O(n \log n)$ time by employing a dynamic data-structure that maintains a set of points Q that changes under insertions, and supports visibility queries of the form in $O(\log n)$ time: "Given a threshold value x_0 and a query range (y_{bottom}, y_{top}), return the point of Q (if any) with the largest y-coordinate that is located within the rectangle $(0, x_0) \times (y_{bottom}, y_{top})$" [17, pp. 209]. □

4 The (1P, Left-Sided, opo) Mixed Labeling Model

In this section, we adopt the scenario of the previous section assuming that the external labels occupy the left side of R. Again, our goal is to obtain a labeling with maximum number of internal labels. In contrast to the case where the external labels are to the right of R, in this case when two labels overlap it is not necessary that the lower label is always external in a feasible solution and this makes the problem more complicated (see the left part of Figure 2).

4.1 A Quasi-polynomial Time Solution

We first present a quasi-polynomial time algorithm, assuming that the labels are of uniform height. Our recursive algorithm splits the problem into a particular number of half-sized subproblems by fixing an internal label each time, then recursively solves the subproblems and derives a solution of the initial problem by keeping the solution which implies the maximum number of internal labels.

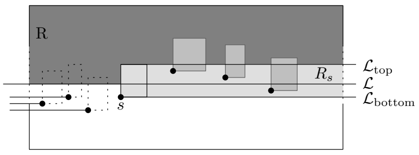


Fig. 3. An illustration of our recursive algorithm

Consider an arbitrary line $\mathcal{L}$ that is parallel to x-axis and intersects several internal labels and let l_s be any of these internal labels. Let also s be its corresponding site. In the final optimal solution, site s will be labeled either by an internal or by an external label. Consider a solution in which site s is the first site (from left to right) out of those intersected by line $\mathcal{L}$ to be labeled by an internal label (see Fig. 3). The sites to the left of s with internal labels intersecting $\mathcal{L}$ must all be routed through leaders. Let $\mathcal{L}_{\text{top}}$ ($\mathcal{L}_{\text{bottom}}$) be the horizontal half-line that emanates from the top-left (bottom-left) corner of l_s and coincides with the top (bottom) boundary edge of l_s. Also, let R_s be the rectangle defined by the left side of l_s, $\mathcal{L}_{\text{top}}$, $\mathcal{L}_{\text{bottom}}$ and the right side of R (see the light-gray rectangle of Fig. 3). Then, all sites inside R_s must be labeled by internal labels.

First, we have to check whether all sites inside R_s can be labeled by internal labels, which needs $O(n \log^2 n)$ time using range queries [17]. Next, we turn our attention to the solution of the internal label maximization problem restricted to instances where l_s is the first internal label (from left to right) out of those that intersect $\mathcal{L}$. The maximum number of internal labels can be determined by the solution of two independent problems, each consisting of a set of sites to be labeled and a set of already labeled sites through internal or external labels. For the first (top) subproblem, the set of sites consists of all unlabeled sites above line $\mathcal{L}$ and half-line $\mathcal{L}_{\text{top}}$ (within the dark-gray polygonal region of Fig.3), while the set of fixed labels consists of the internal labels for the sites in R_s, plus l_s (plus, any previously fixed label placements). For the second (bottom) subproblem, the set of sites consists of the remaining unlabeled sites, while the set of fixed sites consists of all sites in R_s, plus all sites to the left of s which are labeled by leaders, plus s (plus, any previously fixed label placements).

The global optimal solution can then be easily obtained by considering each of the internal labels crossed by $\mathcal{L}$ as the first internal label. If line $\mathcal{L}$ is carefully selected so that it has half of the unlabeled sites above and below it, then the time needed to calculate the optimal solution is given by the following formula:

$$T(k) = \begin{cases} 2kT(k/2) + n^2 \log^2 n, & k > 1 \\ O(1), & k = 1 \end{cases}$$

The $n^2 \log^2 n$ term is needed in order to check whether the labeling of the sites in each rectangle R_s is legal. The solution of this formula leads to a time complexity of $O(n^{\log n + 3})$. So, we can state the following theorem.

Theorem 2. *The (1P, left-sided, opo) problem with labels of uniform height can be solved in $O(n^{\log n+3})$ time.*

4.2 A 2-Approximation Algorithm

In the following, we present a more efficient, factor 2-approximation algorithm, which is based on a reduction to a variant of the 2-SAT problem. Reductions to the 2-SAT problem have been previously used in the map labeling literature [9,18]. In our approach, each site s_i in the labeling instance is identified by a boolean variable z_i, $i = 1, 2, \ldots, n$. The true or false value of each variable z_i specifies whether s_i is labeled through an external or an internal label, respectively. We proceed to construct a set of clauses based on the following cases:

1. **Avoidance of internal label overlaps:** Let $\overline{z_i}$ and $\overline{z_j}$ be two internal label candidates that overlap and assume that neither site s_i nor s_j is contained in $\overline{z_j}$ and $\overline{z_i}$, respectively. Then, $\overline{z_i}$ and $\overline{z_j}$ cannot simultaneously appear in a solution, which is ensured by clause: $z_i \vee z_j$.
2. **Avoidance of site and internal label overlaps:** Let $\overline{z_i}$ and $\overline{z_j}$ be two internal label candidates that overlap and assume that s_j is contained in $\overline{z_i}$. Then, $\overline{z_i}$ cannot appear in a solution, which is ensured by clause: z_i.
3. **Avoidance of leader and internal label intersections:** Let s_i and s_j be a pair of sites, such that the leader of s_j crosses the internal label of s_i. Then, z_j and $\overline{z_i}$ cannot simultaneously appear in a solution. This is ensured by clause: $z_i \vee \overline{z_j}$.

Having formulated our problem as a 2-SAT clausal problem, we need a satisfying truth assignment which simultaneously minimizes the number of true variables. Gusfield and Pitt [11] proved that given a satisfiable 2-SAT formula, it is $\mathcal{NP}$-hard to find a satisfying assignment that contains a minimum number of true variables and proposed an approximation that results in an assignment with at most twice as many true variables as necessary in $O(NM)$ time, where N and M are the number of variables and clauses, respectively. Hence, we can state the following theorem:

Theorem 3. *The* (1*P, left-sided, opo) problem admits a factor* 2 *approximation algorithm which needs* $O(n^3)$ *time.*

4.3 An ILP Formulation

In this subsection, we provide an alternative solution of the (1P, left-sided, *opo*) labeling problem, that is, a formulation as an integer linear program. ILP formulations have also been previously used in the map labeling literature [7,20]. In our formulation, each site s_i is associated with two variables z_i and w_i, each of which has value 0 or 1 indicating the absence or presence of an internal label and a leader, respectively. Then, one set of constraints expresses the requirement that each site should be labeled exactly once, either through an internal label or through a leader. Therefore, $z_i + w_i = 1$, for each $i = 1, 2, \ldots, n$. A second set of constraints expresses the requirement that no two internal label candidates overlap. More precisely, if s_i and s_j is a pair of sites, whose internal labels overlap then a constraint of the form $z_i + z_j \leq 1$ should be present. Finally, a last set of constraints is needed in order to avoid crossings among leaders and internal labels. So, the case where the leader emanating from a site s_i crosses the internal label of another site s_j, can be expressed by a constraint of the form $w_i + z_j \leq 1$. Since we seek to maximize the number of internal labels, the objective function of our integer linear program is $\sum_{i=1}^{n} z_i$. Clearly, the set of constraints can be constructed in $O(n^2)$. Therefore, we can state the following theorem.

Theorem 4. *An instance of the (1P, left-sided, opo) problem can be transformed into an integer linear program in $O(n^2)$ time.*

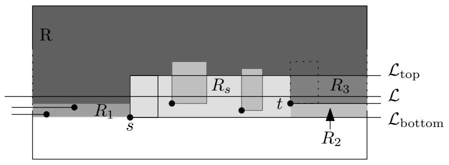
Note that the formulation given above with small modifications can be used to solve any type-*opo* mixed labeling problem that involves a fixed position model. However, the ILP presented above has an extra property; it contains two variables per constraint. Bar-Yehuda and Rawitz [1] proved that such integer programs can be approximated by a factor of 2 in $O(NMU)$ time, where N, M and U are the number of variables, constraints and the maximum variable range, respectively. This implies that our problem admits another 2-approximation algorithm that also needs $O(n^3)$ time.

5 The (1P, Two-Sided, *opo*) Mixed Labeling Model

In this section, we adopt the 1P point-labeling model supported by the two opposite-sided type-*opo* boundary labeling model as alternative, assuming that the labels are of uniform height and the external labels are allowed to be placed along the left and the right side of R. The algorithm we describe follows the lines of the recursive algorithm for the left sided case, presented in Section 4. However, due to the fact that for each site we have three possible labeling alternatives (i.e., leader to the left, internal label, leader to the right), the splitting into subproblems is slightly more complicated.

Let h be the height of each label and λ be the maximum number of sites within any strip of height h. Consider some legal labeling and as before, consider an arbitrary line $\mathcal{L}$ that is parallel to the x-axis and intersects several internal labels. Let again l_s be the leftmost of these labels and s be its corresponding site. Define as before lines $\mathcal{L}_{\text{top}}$ and $\mathcal{L}_{\text{bottom}}$. Let t be the leftmost site out of those to the right of label l_s that is labeled by a leader towards the right side of R. Then, the following must hold:

- Let R_1 be a rectangle (of maximum height h) whose bottom-right corner coincides with site s, whereas its top-left corner coincides with the intersection of line $\mathcal{L}$ and the left side of the enclosing rectangle R (see Fig.4a). Then, all sites in R_1 are labeled through leaders to the left side of R.
- Let R_s be the rectangle that has the sites s and t on its left and right sides, respectively, and is defined by the half-lines $\mathcal{L}_{\text{top}}$ and $\mathcal{L}_{\text{bottom}}$ (see Fig.4a). Then, all sites in R_s must be labeled through internal labels.



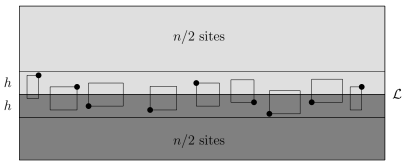


Fig. 4. Illustrations of our recursive algorithms

- Let R_2 be the rectangle having site t as its top-left corner and the intersection of $\mathcal{L}_{\text{bottom}}$ with the right side of R as its bottom-right corner. Then, all sites in R_2 must be labeled through leaders to the right side of R.
- Let R_3 be the rectangle with site t as its bottom-left corner and the intersection of $\mathcal{L}_{\text{top}}$ with the right side of R as its top-right corner. Then, all sites in R_3 must be labeled by internal labels or through leaders to the right side of R.

As in the previous case, the maximum number of internal labels can be determined by the solution of two independent problems, each consisting of a set of sites to be labeled and a set of already labeled sites through internal or external labels. For the first (top) subproblem, the set of sites consists of all unlabeled sites that lie above line $\mathcal{L}$, half-line $\mathcal{L}_{\text{top}}$ and in the interior of R_3 (within the dark-gray polygonal region of Fig.4a), while the set of fixed labeled sites consists of the sites in R_s (which are labeled by internal labels), plus, s and t (plus, any previously fixed label placements). For the second (bottom) subproblem, the set of sites consists of the remaining unlabeled sites, while the set of fixed labeled sites consists of all sites in $R_1 \cup R_2$ (which are labeled by leaders), plus, the sites in R_s (which are labeled by internal labels), plus, s and t (plus, any previously fixed label placements).

Attempting to do an analysis as in the previous case, we realize that there are at most $2\lambda^2$ subproblems, each defined by a pair of sites s and t (as defined above). If line $\mathcal{L}$ equally-splits the sites, observe that the top subproblem might have at most $n/2$ sites plus the sites of R_3 that are below line $\mathcal{L}$. Since each slice of height h contains at most λ sites, the top subproblem contains at most $n/2 + \lambda$ sites. Then, the bottom subproblem contains at least $n/2 - \lambda$ sites. So, in the i-th recursion-level, each subproblem has size at most $n/2^i + \sum_{k=0}^{i} \lambda/2^k$, which approximately equals $n/2^i + 2\lambda$. Note that parameter λ is not halfed at each recursion level, the subproblems however do.

We choose to stop the recursion when the size of a subproblem is less than 3λ to facilitate our analysis. Observe that a problem of size 3λ can be solved in $O(3^{3\lambda}\lambda \log^2 \lambda)$ time. For each site, we have three choices, i.e., leader to the left, internal label, leader to the right. Thus, the time complexity of our algorithm is given by the following formula.

$$T(n) \leq \begin{cases} \lambda^2(2T(n/2+\lambda) + n\log^2 n), & n > 3\lambda \\ 3^{3\lambda}\lambda\log^2\lambda, & n \leq 3\lambda \end{cases}$$

The above formula, for $n > 3\lambda$ means that we have $\log(n/3\lambda)$ recursion levels, giving a total time of $O(n(n/(3\lambda))^{2\log\lambda}\log^2 n + 2\lambda(n/(3\lambda))^{2\log(2\lambda)}\log^2 n)$, while the second term gives for each subproblem of size λ an amount of at most $3^{3\lambda}\lambda\log^2\lambda$. On level i, we have $2^i\lambda^{2i}$ subproblems, hence in total the recursion ends with $(n/(3\lambda))^{2\log\lambda+1}$ subproblems. In total this gives a seemingly complicated formula for the time complexity: $O(n(n/(3\lambda))^{2\log\lambda}\log^2 n + 2\lambda(n/(3\lambda))^{2\log(2\lambda)} + (n/(3\lambda))^{2\log\lambda+1}\cdot 3^{3\lambda}\lambda\log^2\lambda)$. We conclude by the following theorem. If λ is constant, the following theorem gives a polynomial time bound.

Theorem 5. *An instance of the (*$1P$*, 2-sided, opo) labeling problem can be solved in time* $(n/\lambda)^{O(\log\lambda)}\cdot 3^{O(\lambda)}$.

5.1 Generalizations

In the following, we present a general scheme for the mixed labeling model (m, k, opo), where $m \in \{2PH, 2PV, 4P\}$, $k \in \{\emptyset$, Left-Sided, Right-Sided, Two-Sided$\}$. The proposed scheme also supports labelings without external labels. It resembles to one of the first map labeling algorithms by Kucera et al [14].

Let κ and μ be two variables defined as follows. Variable κ equals to (a) zero, if $k = \emptyset$, (b) one, if $k \in \{$Left-Sided, Right-Sided$\}$, and, (c) two, if $k =$ Two-Sided. Similarly, variable μ equals to (a) two, if $m \in \{2PH, 2PV\}$, and, (b) four, if $m = 4P$. Let also $\mathcal{P}$ be a labeling problem with labels of uniform height h and λ-height restricted, i.e., in each horizontal strip of height h there are at most λ sites. Let $\mathcal{L}$ be a horizontal line bisecting the rectangle into two subproblems $\mathcal{P}_{top}$ and $\mathcal{P}_{bot}$, such that the subproblems have at most $n/2$ sites each. Because of the height restriction, at most λ sites above $\mathcal{L}$ might have a bottom-label which intersects $\mathcal{L}$, and analogously at most λ sites below $\mathcal{L}$ might have a top-label which intersects $\mathcal{L}$ (see Fig.4b). It is clear that any solution of $\mathcal{P}$ consists of a configuration $\mathcal{C}$ of labels intersecting $\mathcal{L}$ and the solution of $\mathcal{P}_{top}(\mathcal{C})$ and $\mathcal{P}_{bot}(\mathcal{C})$, where the two subproblems are modified according to the configuration $\mathcal{C}$. For each configuration $\mathcal{C}$, we solve the corresponding subproblems $\mathcal{P}_{top}(\mathcal{C})$ and $\mathcal{P}_{bot}(\mathcal{C})$, construct the solution $\mathcal{P}(\mathcal{C})$ and optimize over all configurations $\mathcal{C}$.

For each configuration $\mathcal{C}$, labels of the sites close to line $\mathcal{L}$ might intersect $\mathcal{L}$ or not. The sites above and below might intersect the strip only when they use their lower or upper internal labels, or they do not intersect at all, hence we have $\kappa/2+1$ possibilities for the upper sites and $\kappa/2+1$ possibilities for the sites below. Hence, we have at most $(\kappa/2+1)^\lambda\cdot(\kappa/2+1)^\lambda$ different configurations for the middle line $\mathcal{L}$. If we fix one of those configurations, then the two subproblems arising on top or below the strip are independent of each other, and can be solved recursively. They only depend on the fixed label configuration of the middle strip and they have size at most $n/2$ each. Each configuration can be checked, whether it is legal or not, in time $O(n\log^2 n)$, using the methods described in Section 4.

Note that the recursion stops when $n \le \lambda$, after $\log(n/\lambda)$ recursion levels. Each time, we have to check the legality of the solution for the subproblem. This can be done in time $O(\lambda\log^2\lambda)$ by a series of $O(\lambda)$ range queries concerning at most $O(\lambda)$ rectangles. Then, $T(\lambda) \le O((\kappa+\mu)^\lambda\lambda\log^2\lambda)$. Hence, we get a similar recursion as before, which can be summarized by the following theorem.

Theorem 6. *A λ-height restricted instance of the (m, k, opo) problem, where $m \in \{2PH, 2PV, 4P\}$, $k \in \{\emptyset$, Left-Sided, Right-Sided, Two-Sided$\}$ can be solved in time $(n/\lambda)^{O(\lambda \log \kappa)} \cdot (\kappa + \mu)^{\lambda+1}$.*

More concretely, if $m \in \{2PH, 2PV\}$ and $k \in \{$Two-Sided$\}$, a λ-height restricted instance of this problem can be solved in $O((n/\lambda)^{1+2\lambda} \cdot (4^{\lambda}\lambda \log^2 \lambda + n \log^2 n)$. If $m \in \{4P\}$ and $k \in \{$Left-Sided, Right-Sided$\}$, we have 5 choices for each site (either through a leader or through an internal label utilizing one of the four available internal label candidates). For the sites directly above and below the strip, we have 3 choices, since two of the four available internal label candidates cannot be utilized. Hence, there are $3^{\lambda} \cdot 3^{\lambda}$ configurations for the middle line. Therefore, a λ-height-restricted instance of this model can be solved in $O((n/\lambda)^{1+\lambda \log 9} \cdot (5^{\lambda}\lambda \log^2 \lambda + n \log^2 n)$. When λ is constant or logarithmic in n, we obtain polynomial or quasi-polynomial algorithms, which do not contradict the $\mathcal{NP}$-hardness of the general problem if λ is large ($\lambda \geq n^{\epsilon}, \epsilon > 0$).

6 Experimental Results

In this section, we present the results of the experimental evaluation of the algorithms presented in Sections 3, 4.1 and 5. The experiment was performed on a Linux machine with 2.60 GHz CPU and 2GB RAM. A parameter that was taken into account in the experiment's setup was the *density* of each input point set, that is the ratio of the total area of all labels to the area of R. The labels had a predefined, fixed size. Hence, a specific density value together with a given value for the total number of point sites, also specifies the dimensions of the enclosing rectangle. The density values vary from 1/10 (low density) to 1/5 (high density), whereas the point sets were randomly generated with 5 up to 100 sites. Given a density value and a number of point sites, the experiment generated 100 random point sets, which were given as input to the three algorithms.

As expected the one-sided, polynomial time algorithm of Section 3 is slightly faster than the corresponding one of Section 4.1 that runs in quasi-polynomial time. Both algorithms need less than half a second to perform the labeling, regardless of the density of the labeling or the size of the input point set. On the other hand, the two-sided, quasi-polynomial time algorithm of Section 5 needs noticeably more time, especially in large and dense point sets. For input point sets consisting of less than 40 points the algorithm needs less than two and a half seconds. For input point sets with 50 up to 60 point sites less than a minute. However, for more dense point sets consisting of more than 70 points, there exist instances which need more than ten minutes. Note that in our experiment we didn't bound the parameter λ.

Regarding the quality of the produced labelings, which is measured in terms of the number internal labels, the two-sided, quasi-polynomial time algorithm of Section 5 produces labelings that have significantly more internal labels than the other two algorithms, especially in large and dense point sets. On the other hand, the one-sided, quasi-polynomial time algorithm of Section 4.1 and the one-sided, polynomial time algorithm of Section 3 have roughly the same performance.

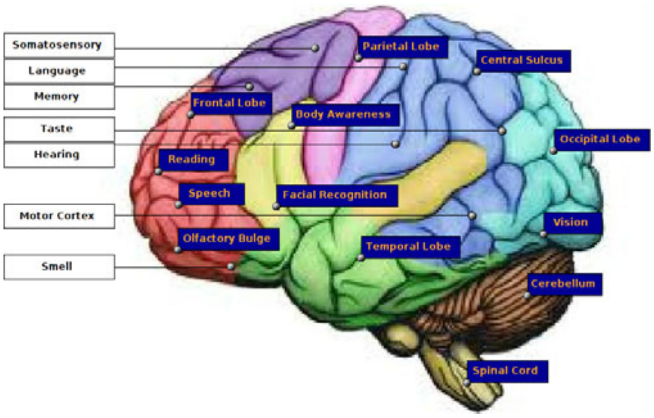


Fig. 5. The anatomy of a human brain produced by the (1P, left-sided, *opo*) model

Figure 5 depicts a medical map that describes the anatomy of a human brain. The labeling has been produced by our one-sided quasi-polynomial time algorithm of Section 4.1, in which we have also performed an additional post-process step, in order to minimize the number of leader-bends (using an algorithm given in [5]). Since the external labels are few, most of the leaders are eventually drawn as straight-lines, which drastically improves the quality of the final labeling.

7 Conclusions

Our experimental results indicate that our algorithms have practical impact, even if they are mostly subexponential (except from the first one). Of course, there is large space for improvements. The proposed model is not appropriate for every instance of map labeling problems. We evaluated our algorithms in terms of time complexity and number of internal labels. It should also be evaluated whether the criterion of maximizing the number of internal labels yields aesthetically valuable labelings. The extension from the type-*opo* boundary labeling model to the more appealing type-*po* (or even to octilinear models [2]) is still open and seems nontrivial. Other variants to be considered might be to use leaders only when they are bend-less, to incorporate length-restrictions on the leaders or more general penalty functions for intersections between internal labels and leaders.

References

1. Bar-Yehuda, R., Rawitz, D.: Efficient algorithms for integer programs with two variables per constraint (extended abstract). Algorithmica 29(44), 595–609 (2001)
2. Bekos, M.A., Kaufmann, M., Nöllenburg, M., Symvonis, A.: Boundary Labeling with Octilinear Leaders. Algorithmica 57(3), 436–461 (2009)

3. Bekos, M.A., Kaufmann, M., Potika, K., Symvonis, A.: Multi-Stack Boundary Labeling Problems. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 81–92. Springer, Heidelberg (2006)
4. Bekos, M.A., Kaufmann, M., Potika, K., Symvonis, A.: Polygons Labelling of Minimum Leader Length. In: Kazuo, M., Kozo, S., Jiro, T. (eds.) Proc. 5th Asia Pacific Symposium on Information Visualisation (APVIS206). CRPIT, vol. 60, pp. 15–21. ACS Inc. (2006)
5. Bekos, M.A., Kaufmann, M., Symvonis, A., Wolff, A.: Boundary Labeling: Models and Efficient Algorithms for Rectangular Maps. Computational Geometry: Theory and Applications 36, 215–236 (2007)
6. Benkert, M., Haverkort, H., Kroll, M., Nöllenburg, M.: Algorithms for multi-criteria one-sided boundary labeling. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 243–254. Springer, Heidelberg (2008)
7. Cromley, R.G.: A spatial allocation analysis of the point annotation problem. In: Proc. 2nd International Symposium on Spatial Data Handling (SDH 1986), pp. 38–49 (1986)
8. Edelsbrunner, H.: Dynamic data structures for orthogonal intersection queries. Tech. Rep. F59, Tech. Univ. Graz, Institute für Informationsverarbeitung (1980)
9. Formann, M., Wagner, F.: A packing problem with applications to lettering of maps. In: Proc. 7th Annual ACM Symposium on Computational Geometry, pp. 281–288 (1991)
10. Fowler, R.J., Paterson, M., Tanimoto, S.L.: Optimal Packing and Covering in the Plane are NP-Complete. Information Processing Letters 12(3), 133–137 (1981)
11. Gusfield, D., Pitt, L.: A bounded approximation for the minimum cost 2-Sat problem. Algorithmica 8(2), 103–117 (1992)
12. Kao, H.-J., Lin, C.-C., Yen, H.-C.: Many-to-one boundary labeling. In: Hong, S.-H., Ma, K.-L. (eds.) Proc. 6th International Asia-Pacific Symposium on Visualization (APVIS 2007), pp. 65–72. IEEE, Los Alamitos (2007)
13. van Kreveld, M., Strijk, T., Wolff, A.: Point labeling with sliding labels. Computational Geomentry: Theory and Applications 13, 21–47 (1999)
14. Kucera, L., Mehlhorn, K., Preis, B., Schwarzenecker, E.: Exact algorithms for a geometric packing problem. In: Enjalbert, P., Wagner, K.W., Finkel, A. (eds.) STACS 1993. LNCS, vol. 665, pp. 317–322. Springer, Heidelberg (1993)
15. Lin, C.-C.: Crossing-free many-to-one boundary labeling with hyperleaders. In: 3rd IEEE Pacific Visualization Symposium (PacificVis 2010), pp. 185–192. IEEE Press, Los Alamitos (2010)
16. Lin, C.-C., Wu, H.-Y., Yen, H.-C.: Boundary labeling in text annotation. In: 13th International Conference on Information Visualisation (IV2009), pp. 110–115. IEEE, Los Alamitos (2009)
17. Mehlhorn, K.: Data structures and algorithms 3: multi-dimensional searching and computational geometry. Springer-Verlag New York, Inc., Heidelberg (1984)
18. Poon, C.K., Zhu, B., Chin, F.: A polynomial time solution for labeling a rectilinear map. In: Proc. 13th Annual ACM Symposium on Comp. Geom (SoCG 1997), pp. 451–453 (1997)
19. Wolff, A., Strijk, T.: The Map-Labeling Bibliography, maintained since (1996), `http://i11www.ira.uka.de/map-labeling/bibliography`
20. Zoraster, S.: Integer programming applied to the map label placement problem. Cartographica 23(3), 16–27 (1986)

On Making a Distinguished Vertex Minimum Degree by Vertex Deletion

Nadja Betzler[1,*], Robert Bredereck[2,*], Rolf Niedermeier[2], and Johannes Uhlmann[1,**]

[1] Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany
[2] Institut für Softwaretechnik und Theoretische Informatik, Technische Universität Berlin, Franklinstr. 28/29, D-10587 Berlin, Germany
{nadja.betzler,robert.bredereck,johannes.uhlmann}@uni-jena.de, rolf.niedermeier@tu-berlin.de

Abstract. For directed and undirected graphs, we study the problem to make a distinguished vertex the unique minimum-(in)degree vertex through deletion of a minimum number of vertices. The corresponding NP-hard optimization problems are motivated by applications concerning control in elections and social network analysis. Continuing previous work for the directed case, we show that the problem is W[2]-hard when parameterized by the graph's feedback arc set number, whereas it becomes fixed-parameter tractable when combining the parameters "feedback vertex set number" and "number of vertices to delete". For the so far unstudied undirected case, we show that the problem is NP-hard and W[1]-hard when parameterized by the "number of vertices to delete". On the positive side, we show fixed-parameter tractability for several parameterizations measuring tree-likeness, including a vertex-linear problem kernel with respect to the parameter "feedback edge set number". On the contrary, we show a non-existence result concerning polynomial-size problem kernels for the combined parameter "vertex cover number and number of vertices to delete", implying corresponding nonexistence results when replacing vertex cover number by treewidth or feedback vertex set number.

1 Introduction

Making a distinguished vertex minimum degree by vertex deletion is a natural though widely unexplored graph problem. We contribute new insights into the algorithmic complexity of the corresponding computational problems, providing intractability as well as fixed-parameter tractability results.

Formally, we studied the following two decision problems.

* Supported by the DFG, research project PAWS, NI 369/10.
** Supported by the DFG, research project PABI, NI 369/7.

Min-Indegree Deletion (MID)
Given: A directed graph $D = (W, A)$, a distinguished vertex $w_c \in W$, and an integer $k \geq 1$.
Question: Is there a subset $W' \subseteq W \setminus \{w_c\}$ of size at most k such that w_c is the only vertex that has minimum indegree in $D[W \setminus W']$?

Whereas MID has been studied in one previous paper [2], its undirected counterpart seems completely unexplored:

Min-Degree Deletion (MDD)
Given: An undirected graph $G = (V, E)$, a distinguished vertex $w_c \in V$, and an integer $k \geq 1$.
Question: Is there a subset $V' \subseteq V \setminus \{w_c\}$ of size at most k such that w_c is the only vertex that has minimum degree in $G[V \setminus V']$?

MID directly emerges from a problem concerning electoral control with respect to so-called "Llull voting" [2,9], one of the well-known voting systems based on pairwise comparision of candidates. Concerning MDD, in undirected social networks the degree of a vertex relates to its popularity or influence [18, pages 178–180]. Then, making a distinguished vertex minimum degree (equivalently, making it maximum degree in the complement graph) corresponds to activities or campaigns where a single agent shall be transformed to the least or most important agent in its community. Minimum vertex deletion, hence, can be interpreted as making "competing agents" disappear at minimum cost. A problem related to MDD is Bounded Degree Deletion (BDD) and its dual problem (considering the complement graph) Maximum s-plex. For BDD the goal is to bound the maximum vertex degree by a prespecified value d (the case $d = 0$ is equivalent to the well-known Vertex Cover problem) using a minimum number of vertex deletions. Other than MDD, BDD and its dual Maximum s-plex have been studied quite intensively in recent years [1,10,14] which is due to their applications in social and biological network analysis.

Although both MID and MDD are simple and natural graph problems, we only know one previous publication concerning these problems. MID has been shown W[2]-complete for parameter solution size k even when restricted to tournament graphs and it is polynomial-time solvable on directed acyclic graphs [2].

We initiate a thorough theoretical analysis of MID and MDD mainly focussing on "tree-likeness" parameterizations. We employ several basic structural parameters measuring the tree-likeness of graphs. The most famous parameter is the *treewidth* tw of the input graph, which comes along with the concept of tree decompositions of graphs.[1] The *feedback vertex set number* s_v of a graph is the minimum number of vertices to delete from a graph to make it acyclic. Correspondingly, the *feedback edge set number* s_e and the the *feedback arc set number* s_a, respectively, denote the minimum number of edges or arcs to delete from an undirected or directed graph to make it acyclic. While the computation of tw, s_v and s_a leads to NP-hard problems, s_e can be quickly determined by a

[1] We omit any details because we will not need the formal definition in this work.

Table 1. Overview of the parameterized complexity of MIN-INDEGREE DELETION and MIN-DEGREE DELETION. The considered parameters are tw := "treewidth of the input graph", s_v := "size of a feedback vertex set", s_a := "size of a feedback arc set", s_v^* := "size of a feedback vertex set not containing w_c", s_e := "size of a feedback edge set", k := "number of vertices to delete", and d := "maximum degree". New results are in boldface. The remaining results are from [2].

parameter	MIN-INDEGREE DELETION	MIN-DEGREE DELETION
tw	—	**FPT, no polynomial kernel**
s_v	**W[2]-hard**	**FPT, no polynomial kernel**
s_v^*	**W[2]-hard**	$\mathbf{O((2s_v^* + 4)^{s_v^*} \cdot n^6)}$**, no polynomial kernel**
s_a, s_e	**W[2]-hard**	$\mathbf{O(2^{s_e} \cdot n^3)}$**, vertex-linear kernel**
k	W[2]-complete	**W[1]-hard**
d	FPT	FPT
(s_v, k)	$\mathbf{O(s_v \cdot (k+1)^{s_v} \cdot n^2)}$**, no polynomial kernel**	

spanning tree computation. Note that a small value of s_e means that the network is very sparse—however, there are several sparse social networks [13,16,17].

Table 1 summarizes our results. We extend the previous results for MID [2] by showing that MID is W[2]-hard even when parameterized by s_a whereas it turns fixed-parameter tractable for the combined parameter (k, s_v). Note that this also implies fixed-parameter tractability with respect to the combined parameter (k, s_a) since s_a is a *weaker* parameter than s_v in the sense that $s_v \leq s_a$. As to MDD, we show that it is NP-complete as well as W[1]-hard with respect to the parameter k, devising a parameterized many-one reduction from the INDEPENDENT SET problem. In addition, we show that MDD is fixed-parameter tractable for each of the tree-likeness parameters treewidth, size s_v^* of a feedback vertex set not containing the distinguished vertex, and feedback edge set number s_e. Herein, our fixed-parameter tractability result for treewidth is of purely theoretical interest whereas the one for the feedback edge set number comes along with a $2s_e$-vertex problem kernel and a size-$O(2^{s_e})$ search tree. The result for s_v^* relies on dynamic programming and bears a combinatorial explosion of $O((2s_v^* + 4)^{s_v^*})$. Finally, building on a recent framework for proving non-existence of polynomial-size problem kernels [3], we also show that there is presumably no polynomial-size problem kernel for MDD even for the combined parameter (k, s_c^*), where s_c^* denotes the size of a vertex cover *not* containing the distinguished vertex. This directly implies the non-existence of polynomial-size problem kernels for the parameters feedback vertex set number and treewidth. Due to the lack of space, several details are deferred to a full version of this paper.

Preliminaries. Parameterized complexity is a two-dimensional framework for studying the computational complexity of problems [8,11,15]. One dimension is the input size n (as in classical complexity theory), and the other one is the *parameter* k (usually a positive integer). A problem is called *fixed-parameter tractable* (fpt) if it can be solved in $f(k) \cdot n^{O(1)}$ time, where f is a computable function only depending on k. The complexity class consisting of all fpt problems is denoted by FPT. A core tool in the development of fixed-parameter algorithms is polynomial-time preprocessing by *data reduction* [4,12]. Here, the goal is for a given problem instance x with parameter k, to transform it into a new instance x' with parameter k' such that the size of x' is upper-bounded by some function only depending on k, the instance (x, k) is a yes-instance if and only if (x', k') is a yes-instance, and $k' \leq k$. The reduced instance, which must be computable in polynomial time, is called a *problem kernel*, and the whole process is called *reduction to a problem kernel* or *kernelization*.

Downey and Fellows [8] developed a formal framework for showing *fixed-parameter intractability* by means of *parameterized reductions*. A parameterized reduction from a parameterized problem P to another parameterized problem P' is a function that, given an instance (x, k), computes in $f(k) \cdot n^{O(1)}$ time an instance (x', k') (with k' only depending on k) such that (x, k) is a yes-instance of problem P if and only if (x', k') is a yes-instance of problem P'. The basic complexity class for fixed-parameter intractability is called $W[1]$. There is good reason to believe that $W[1]$-hard problems are not fpt [8,11,15]. In this sense, $W[1]$-hardness is the parameterized complexity analog of NP-hardness. The class W[2] means the next higher degree of parameterized intractability.

We assume familiarity with basic graph-theoretic concepts. Let $G = (V, E)$ be an undirected graph. Unless stated otherwise, let $n := |V|$ and $m := |E|$. For $V' \subseteq V$ we denote the subgraph induced by V' as $G[V']$. Furthermore, we write $G - V'$ for $G[V \setminus V']$. The open neighborhood of a vertex v is denoted by $N(v)$ and the *degree* of v is $\deg(v) := |N(v)|$. We use analogous terms for directed graphs and differentiate between in- and out-(degree, neighborhood, etc.) by a subscript in the notation (e.g., $\deg_{\text{in}}(v)$ denotes the indegree of v).

2 Min-Degree Deletion

In this section, we study parameterizations of Min-Degree Deletion by the solution size, that is, the number of vertices to delete, and structural graph parameters measuring the tree-likeness. By devising a parameterized reduction from the W[1]-complete Independent Set problem we obtain the following.

Theorem 1. Min-Degree Deletion *is NP-complete and W[1]-hard for the parameter "number of vertices to delete".*

For the "tree-likeness" parameterizations, we show fixed-parameter tractability results (Subsection 2.1) and refute the existence of some polynomial-size problem kernels (Subsection 2.2).

2.1 Fixed-Parameter Tractability Results

In the following, all structural graph parameters are related to measuring the tree-likeness of the underlying graph. More specifically, we provide results for the treewidth tw, the size s_v^* of a feedback vertex set not containing the distinguished vertex, and the feedback edge set number s_e. By definition, $(\text{tw}+1) \leq s_v^* \leq s_e$. Hence, our fixed-parameter tractability result for MDD for the parameter tw implies fixed-parameter tractability for the parameters s_v^* and s_e. However, since our corresponding result for tw only provides a classification and not an efficient fixed-parameter algorithm, we subsequently present specific fixed-parameter tractability results for each parameterization.

Parameter treewidth. For treewidth, the "strongest" tree-likeness parameter we consider in this section, we obtain the following.

Theorem 2. Min-Degree Deletion *is fixed-parameter tractable for the parameter treewidth.*

The proof of Theorem 2 relies on expressing MDD by a monadic second-order logic (MSO) sentence and making use of Courcelle's famous theorem [6]. Due to the huge constants coming along with Courcelle's machinery this result is of purely theoretical interest. The following observation is crucial to obtain Theorem 2 as well as for some of our other results.

Observation 1. *Let $G = (V, E)$ be a graph of treewidth* tw *and let M^* be any solution set for* MDD*. Then, w_c has degree at most* $\text{tw} - 1$ *in $G - M^*$.*

Parameter distinguished feedback vertex set number. We investigate the parameter *distinguished feedback vertex set number* s_v^* denoting the "size of a feedback vertex set not containing the distinguished vertex w_c". Since for a graph with treewidth tw it holds that $s_v^* \geq s_v \geq (\text{tw}+1)$, Theorem 2 implies that MDD is fixed-parameter tractable with respect to s_v^*. However, Theorem 2 does not come with a direct combinatorial algorithm and hence such an algorithm with running time $O((2s_v^*+4)^{s_v^*} \cdot n^4 \cdot \deg(w_c)^2)$ will be provided in the following.

Let $(G = (V, E), w_c, k)$ be an MDD-instance and let V_f be a feedback vertex set that does not contain w_c. Our algorithm basically branches into all possible subsets V_f^* of V_f and investigates whether there is a solution containing all vertices from V_f^* and not containing any vertex from $V_f \setminus V_f^*$. Furthermore, the algorithm iterates over the "final" degree that w_c might assume in the graph G after deleting a set of "solution vertices". Additionally applying some simple branching and preprocessing steps it remains to solve the following problem.

Annotated Min-Degree Deletion (AMDD)
Given: An undirected graph $G = (V, E)$, a distinguished vertex w_c, a feedback vertex set V_f of G with $V_f \subseteq V \setminus \{w_c\}$, and two non-negative integers k and i.
Question: Is there a subset $M \subseteq V \setminus (V_f \cup \{w_c\})$ of size at most k such that, in $G - M$, $\deg(w_c) = i$ and every other vertex has degree at least $i + 1$?

The branching and the preprocessing giving an AMDD-instance can be carried out in $O(2^{|V_f|} \cdot n^2)$ time. Moreover, due to the preprocessing, in the following we can assume that every vertex in $V \setminus \{w_c\}$ has degree at least $i+1$ and w_c has at most i neighbors in V_f. Now, for an AMDD-instance $(G=(V,E), w_c, V_f, k, i)$, the algorithm makes use of the following property of $V_S := V \setminus (V_f \cup \{w_c\})$, the set consisting of all vertices that can be part of the solution.

Observation 2. *Let $n_1, \dots, n_d$ denote the neighbors of w_c in $G - V_f$. In the graph $G[V_S]$, every vertex n_x belongs to a connected component $T(x)$ such that $T(x)$ is a tree and does not contain any vertex n_y with $n_x \neq n_y$.*

Observation 2 can be seen as follows. Consider two neighbors n_x and n_y of w_c. First, assume that there would be a path from n_x and n_y that does not contain w_c. Adding w_c to this path would induce a cycle and hence V_f would not be a feedback vertex set of G. Hence, every connected component can contain at most one neighbor of w_c. Second, a cycle within a connected component would also violate that V_f is a feedback vertex set. Hence, all connected components induce trees.[2]

Now, we take a look at an arbitrary solution set M of our MDD-instance. Since the final degree of w_c is i, M must contain $\deg_G(w_c) - i$ neighbors of w_c. Putting a vertex $x \in N(w_c) \setminus V_f$ into the solution may decrease the degree of other vertices from $T(x)$ so that they also must be part of the solution. The set $A(x)$ of *affected* vertices that need to be deleted when x is deleted can be computed iteratively as follows. Start with $A(x) := \{x\}$. While there is vertex v with degree at most i in $T(x) - A(x)$, add v to $A(x)$. Since we have to put all vertices of $A(x)$ into a solution when choosing x into the solution, we define the *cost* of x as $\text{cost}(x) := |A(x)|$. Moreover, we will make use of the following easy-to-verify observation.

Observation 3. *A vertex $v \in V_s \setminus (\bigcup_{x \in N(w_c) \setminus V_f} A(x))$ cannot be part of any minimal solution.*

For the graph not containing vertices from the feedback vertex set V_f, a solution could easily be computed by choosing a set of $\deg(w_c) - i$ neighbors of w_c such that the sum of the corresponding costs is minimal. The critical point is that putting a vertex x into the solution set may also decrease the degree of vertices from V_f. By definition, we cannot remove any vertex from V_f. Thus, we must ensure that we "keep" enough vertices from V_s such that the final degree of every vertex from V_f is at least $i+1$. For every vertex $v \in V_f$, we can easily compute the number $n_{\text{fixed}}(v)$ of neighbors which it has "for sure" in every minimal solution. More specifically, $n_{\text{fixed}}(v)$ is the number of neighbors of v in V_f and in $V_s \setminus (\bigcup_{x \in N(w_c)} A(x))$ (see Observation 3).

We introduce some notation measuring how many neighbors of a vertex from V_f must be kept under the assumption that a certain subset $V_r \subseteq V_s$ is *not* part of a solution. More specifically, for a vertex $v \in V_f$, let $n_{V_r}(v)$ be the number of neighbors of v in V_r. Then, the number of additional neighbors that are not

[2] Observation 2 does not hold for a feedback vertex set containing the distinguished vertex. Hence, the following approach cannot be transferred to this more general case.

allowed to be deleted is defined as $s(v, V_r) := i + 1 - n_{\text{fixed}}(v) - n_{V_r}(v)$. This can be generalized as follows.

Definition 1. *For $V_f = \{v_1, \dots, v_{|V_f|}\}$, the* remain-tuple *with respect to $V_r \subseteq V_S$ is $S = (s_1, \dots, s_{|V_f|})$ where $s_j := s(v_j, V_r)$.*

Recall that the task is to search for a set $N \subseteq N(w_c) \setminus V_f$ of $\deg(w_c) - i$ neighbors of w_c of minimum cost such that the degree of every vertex from V_f is at least $i + 1$. Now, for every subset $N' \subseteq N(w_c) \setminus V_f$, the effect of choosing that N' is *not* part of a solution can be described by a remain-tuple. More specifically, a subset $N' \subseteq N(w_c) \setminus V_f$ *realizes* a remain-tuple $(s'_1, \dots, s'_{|V_f|})$ when, for every $v \in V_f$, the number of neighbors of v in $\bigcup_{x \in N'} A(x)$ is at least $i + 1 - n_{\text{fixed}}(v) - s'_i$. Then, a cost-$k$ set $N \subseteq N(w_c)$ containing $\deg(w_c) - i$ neighbors of w_c such that set $N(w_c) \setminus N$ realizes the remain-tuple $(0, \dots, 0)$ corresponds to a solution.

Dynamic programming table. Based on the previous definitions, the dynamic programming table is defined as $D(x, z, S')$ with $x \in \{1, \dots, d\}$ where $d := |N(w_c) \cap V_s|$, $z \leq \min\{x, d - i\}$, and $S' \subseteq \mathcal{S} := \{(s'_1, \dots, s'_{|V_f|}) \mid 0 \leq s'_j \leq i + 1\}$. The entry $d(x, z, S')$ contains the minimum cost of deleting a size-z subset $N' \subseteq \{n_i \in N(w_c) \mid i \leq x\}$ such that $N'_r := N(w_c) \setminus N'$ "realizes" the remain-tuple S'. It follows that $D(\deg(w_c), \deg(w_c) - i, (0, \dots, 0)) \leq k$ if and only if (G, V_f, w_c, k, i) is a yes-instance of AMDD. It is easy to verify that the size of the D is bounded by $\deg(w_c)^2 \cdot (s^*_v + 2)^{s^*_v}$ (see also Observation 1).

One can show that the initialization and update step per entry can be accomplished in $O((s^*_v + 2)^{s^*_v} \cdot n^2 \cdot \deg(w_c)^2)$ time. Hence, together with the running time for the overall branching into all subsets of a feedback vertex set, one ends up with the following.

Theorem 3. Min-Degree Deletion *can be solved in $O((2s^*_v + 4)^{s^*_v} \cdot n^4 \cdot \deg(w_c)^2)$ time with s^*_v being the size of a feedback vertex set not containing w_c.*

Parameter feedback edge set number. As discussed in the beginning of this section, the feedback edge set number is the weakest of our parameters measuring the tree-likeness of graphs. Hence, not surprisingly, we achieve our best running time bounds here, based on kernelization and a simple search tree.

Our problem kernel result relies on the following "low-degree removal" procedure. Let $G = (V, E)$ be an undirected graph and k be a positive integer. Denote by $\text{RLD}(G, k)$ the graph resulting from the following data reduction: If deleting all or all but one neighbors from w_c leads to a solution (by iteratively deleting all further vertices with degree at least zero/one), then return "yes". Otherwise, w_c has degree at least two for every solution. Hence, iteratively remove every vertex with degree at most two and decrease k accordingly. It is easy to verify that $\text{RLD}(G, k)$ is sound and can be executed in $O(n^2 \cdot k)$ time. Note that every vertex different from w_c in $\text{RLD}(G, k)$ has degree at least three.

Theorem 4. *Parameterized by the feedback edge set number s_e,* Min-Degree Deletion *admits a $2s_e$-vertex problem kernel which can be computed in $O(n^2 \cdot k)$ time.*

Proof. Let $(G', k') := \text{RLD}(G, k)$, and let E_d be a size-s_e-feedback edge set for G. The graph $G - E_d$ is a forest. Since each vertex in G' has degree at least three, each leaf in $G' - E_d$ must be incident to at least two edges in E_d. It follows that $G' - E_d$ contains $l \leq s_e$ leaves because each leaf must be incident to two edges of the feedback edge set and each edge of the feedback edge set can be incident to at most two leaves. Furthermore, the number of incidences of the edges in E_d is bounded from above by $2s_e$. Each inner vertex of degree two in $G' - E_d$ must be incident to at least one edge in E_d. Since there are l leaves in $G' - E_d$, only $2s_e - 2l$ incidences are left over. Hence, $G' - E_d$ contains at most $2s_e - 2l$ inner vertices with degree two. Moreover, all remaining vertices must have degree at least three and a tree with l leaves can clearly have at most l such vertices. Altogether, G' consists of at most $l + 2s_e - 2l + l = 2s_e$ vertices. □

Finally, we complement Theorem 4 by a simple search tree algorithm which can be interleaved with the data reduction procedure $\text{RLD}(G, k)$. This yields the following theorem.

Theorem 5. Min-Degree Deletion *can be solved in $O(2^{s_e} \cdot {s_e}^3 + n^2 \cdot k)$ time, where s_e is the feedback edge set number of the input graph.*

2.2 Non-existence of a Polynomial Kernel

We show that, unless coNP ⊆ NP / poly, there is no polynomial kernel for MDD with respect to the parameter $s_c^* :=$"size of a vertex cover that does not contain w_c". Since the treewidth t_w and the feedback vertex set number s_v of a graph are bounded from above by s_c^*, this non-kernelizable result carries over to these two parameterizations.

Theorem 6. MDD *does not admit a polynomial kernel with respect to the combined parameter (s_c^*, k), with s_c^* being the size of a vertex cover not containing w_c and k being the solution size, unless* coNP ⊆ NP / poly.

Proof. Our proof relies on a reduction from Hitting Set (HS) defined as follows. Given a set family $\mathcal{S} := \{S_1^*, \ldots, S_m^*\}$ over a universe $U := \{u_1^*, \ldots, u_d^*\}$ and an integer $k' \geq 0$, HS asks for a subset $U' \subseteq U$ with $|U'| \leq k'$ such that $S_i^* \cap U' \neq \emptyset$ for every i, $1 \leq i \leq m$. Herein, U' is called a *hitting set.*

Dom et al. [7] have shown that HS does not admit a problem kernel of size $(d + k')^{O(1)}$, unless coNP ⊆ NP / poly. Since HS and MDD are NP-complete, it directly follows from a result of Bodlaender et al. [5] that if there is a polynomial-time reduction from HS to MDD such that $(s_c^* + k) \leq (d + k')^{O(1)}$, then MDD does not admit a polynomial kernel with respect to (s_c^*, k) unless coNP ⊆ NP / poly. In the following, we provide such a reduction.

Let (U, S, k') be an HS-instance. We construct an undirected graph $G = (V, E)$ with a distinguished vertex w_c as follows. The vertex set V is the disjoint union of the sets $\{w_c\}$, V_U, $V_{\mathcal{S}}$, C, and L. Herein, $V_U := \{u_i \mid u_i^* \in U\}$, $V_{\mathcal{S}} := \{s_j \mid S_j^* \in \mathcal{S}\}$, $C := \{c_1, \ldots, c_{k'+1}\}$, and $L := \{l_1, \ldots, l_d\}$. The edge set is constructed as follows. There is an edge between u_i and s_j if and only if $u_i^* \in S_j^*$. Moreover, the following edges are added to adjust the degree of w_c to d and, for each other vertex, to at least $k' + 1$. First, w_c is made adjacent to every vertex in V_U. Furthermore, C is transformed into a clique, and each l_i, $1 \leq i \leq d$, is made adjacent to each vertex in C. Finally, each vertex $x \in V_U \cup V_{\mathcal{S}}$ is made adjacent to k' arbitrarily chosen vertices of C. This completes the construction.

Now, observe that each edge of G is incident to a vertex in $C \cup V_U$. Hence, G has a vertex cover of size $k' + 1 + d$ which does not contain w_c. For the correctness of the reduction it remains to show that $(U, \mathcal{S}, k')$ is a yes–instance of HS if and only if $(G, w_c, d - k')$ is a yes–instance of MDD.

"$\Rightarrow$": Let $U' \subseteq U$ with $|U'| = k'$ denote a hitting set of $\mathcal{S}$. We show that $M := \{u_j \mid u_j^* \in U \setminus U'\}$ is a solution for $(G, w_c, d - k')$. First, observe that w_c has degree k' in $G - M$. Moreover, since U' is a hitting set, every vertex in $V_{\mathcal{S}}$ has at least one neighbor in $V_U \setminus M$, and, hence, degree at least $k' + 1$ in $G - M$. For this reason and since we do not delete a neighbor of $L \cup C$, each vertex in $V \setminus \{w_c\}$ has degree at least $k' + 1$. Hence, $(G, w_c, d - k')$ is a yes-instance of MDD.

"$\Leftarrow$": Let $M \subseteq V$ with $|M| \leq d - k'$ denote a solution for $(G, w_c, d - k')$. First, we argue that w_c has degree k' in $G - M$. Clearly, w_c cannot have degree smaller than k'. Furthermore, w_c cannot have degree more than k' in $G - M$; otherwise, since w_c is the only vertex with minimum degree in $G - M$ and each vertex in L has degree $k' + 1$, M must contain every vertex in L. However, $|L| = d > d - k'$. Thus, $\deg_{G-M}(w_c) = k'$ and, as a consequence, $M \subseteq V_U$ and $|M| = d - k'$.

Next, we show that $U' := \{u_i^* \in U \mid u_i \in V_U \setminus M\}$ is a hitting set of size k'. By the observation above, $|U'| = k'$. Assume towards a contradiction that there is a set S_j^*, $1 \leq j \leq m$, with $S_j^* \cap U' = \emptyset$. Thus, for each element $u_i^* \in S_j^*$ the corresponding vertex u_i is in M. Due to the construction of G, vertex s_j has degree k' in $G - M$; since $\deg_{G-M}(w_c) = k'$ this contradicts the fact that w_c is the only vertex with minimum degree. $\square$

Since the treewidth and the feedback vertex set of a graph are bounded from above by s_c^*, we arrive at the following.

Corollary 1. MDD *has no polynomial problem kernel with respect to the parameters feedback vertex set and treewidth, respectively, unless* coNP $\subseteq$ NP / poly.

3 Min-Indegree Deletion

In this section, we show that MID is W[2]-hard with respect to the parameter feedback arc set number s_a. We provide a parameterized reduction from the W[2]-complete Dominating Set (DS) problem [8]. Given an undirected graph and an integer k, DS asks whether there is a size-k subset $V' \subseteq V$ such that every vertex from V is contained in V' or has a neighbor in V'. A corresponding subset is denoted as *dominating set*.

Theorem 7. Min-Indegree Deletion *is W[2]-hard with respect to the feedback arc set number s_a.*

Proof. Given a DS-instance $(G^* = (V^*, E^*), k)$ with $V^* = \{v_1^*, v_2^*, \dots, v_n^*\}$, we construct a directed graph $G = (W, E)$ with feedback arc set number at most $(k+1)^2$ such that $(G, w_c, n-k)$ is a yes-instance of MID if and only if (G^*, k) is a yes-instance of DS.

The vertex set W of G consists of w_c and the union of the following disjoint vertex sets. The sets $V := \{v_i \mid v_i^* \in V^*\}$ and $D := \{d_i \mid v_i^* \in V^*\}$ representing the vertices of G and four sets of auxiliary vertices, namely a set S containing n vertices and three sets X, Y, and Z, each containing $k+1$ vertices. The arcs of G are as follows.

- One arc from v_i to d_j if and only if $v_j^* \in N[v_i^*]$.
- One arc from each vertex in V to w_c.
- One arc from each vertex in X to each vertex in Y, from each vertex in Y to each vertex in Z, and from each vertex in Z to each vertex in X.
- One arc from each of k arbitrarily chosen vertices in Y to each vertex in D.
- One arc from each vertex in Y to each vertex in V.
- One arc from each vertex in X to each vertex in S.

It follows directly from the construction that the distinguished vertex w_c has indegree n and each vertex in $V \cup X \cup Y \cup Z \cup S$ has indegree $k+1$. Since each vertex d_i has one ingoing arc from v_i and k ingoing neighbors from Y, the vertices in D have indegree at least $k+1$.

Furthermore, it is easy to verify that $(W, E \setminus (X \times Y))$ is acyclic and, since $|X| = |Y| = k+1$, the feedback arc set number s_a is at most $(k+1)^2$. This finishes the description of the construction. It remains to prove the correctness.

Claim: (G^*, k) is a yes-instance of DS if and only if $(G, w_c, n-k)$ is a yes-instance of MID.

"$\Rightarrow$": Let $V_d^* \subseteq V^*$ be a size-k dominating of G^*. We show that $M_d := \{v_i \in V \mid v_i^* \notin V_d^*\}$ is a solution for MID. Since $|M_d| = n-k$ and w_c has indegree n in G, w_c has indegree k in $G - M_d$. We show that all other vertices have degree at least $k+1$. By construction, every vertex in G has indegree at least $k+1$. Since from the vertices in V_d^* there are only arcs to $D \cup \{w_c\}$, only vertices from $D \cup \{w_c\}$ can have smaller indegrees in $G - M_d$ than in G. Because V_d^* is a dominating set, every d_i has at least one in-neighbor within $V \setminus M_d$. Moreover, every d_i has k further in-neighbors in Y. Hence, each vertex in D has indegree at least $k+1$. Thus, $(G, w_c, n-k)$ is a yes-instance of MID.

"$\Leftarrow$": Consider a yes-instance $(G, w_c, n-k)$ of MID with solution M_d. We show that $V_d^* := \{v_i^* \in V^* \mid v_i \in V \setminus M_d\}$ is a size-k dominating set of G^*.

We first prove that V_d^* has cardinality k. To this end, we show by contradiction that the indegree of w_c in $G - M_d$ is k and hence M_d contains only vertices from V. Assume that w_c has indegree at least $k+1$ in $G - M_d$. Then, every other vertex must have indegree greater than $k+1$ in $G - M_d$. Since every vertex in S has indegree exactly $k+1$, it follows that $S \subseteq M_d$ and hence $|M_d| \geq n$; a contradiction. Consequently, $|V \cap M_d| = n-k$ and, hence, V_d^* has cardinality k.

It remains to show that V_d^* is a dominating set. Assume that there is a vertex $v_i^* \in V^*$ not dominated by any vertex in V_d^*. This implies that d_i has no in-neighbor from V in $G - M_d$. Moreover, by construction, d_i has only k in-neighbors in $G - V$. As argued above, d_i is not in M_d since M_d contains only vertices from V. Hence, d_i and w_c have indegree k in $G - M_d$; a contradiction.

In the remainder of this section, we show fixed-parameter tractability of MID with respect to the combined parameter feedback vertex set number s_v and number k of vertices to be deleted. The corresponding branching algorithm relies on the following lemma.

Lemma 1. *For a yes-instance $(G = (V, E), w_c, k)$ of* MID*, the indegree of w_c in G is at most $k + s_v$, where s_v denotes the feedback vertex set number of G.*

Proof. The proof is by contradiction. Let $V_f \subseteq V$ be a feedback vertex set of size s_v. Assume that $\deg_{\text{in}}(w_c) > s_v + k$. For every subgraph G' of G obtained by deleting k vertices from $G - \{w_c\}$, one can make the following two observations. First, since $G' - V_f$ is acyclic, there must be a vertex v with indegree zero in $G' - V_f$. Hence, the indegree of v in G' is at most s_v (in case that v has one ingoing arc from every vertex in V_f). Second, since $\deg_{\text{in}}(w_c) > s_v + k$ in G, it follows that $\deg_{\text{in}}(w_c) > s_v$ in G'. Consequently, there is no size-k subset of vertices that can be deleted from G such that w_c is a vertex with minimum indegree; a contradiction to the fact that $(G = (V, E), w_c, k)$ is a yes-instance. $\square$

Now, by applying an algorithm branching on all up-to-size-k subsets of the in-neighborhood of w_c and checking whether a corresponding subset can be extended to a solution, one arrives at the following theorem.

Theorem 8. Min-Indegree Deletion *can be solved in $O((k + 1)^{s_v} \cdot s_v \cdot n^2)$ time.*

4 Conclusion

We introduced the NP-hard vertex deletion problem Min-Degree Deletion on undirected graphs. For Min-Degree Deletion and its directed counterpart Min-Indegree Deletion we provided several results concerning their fixed-parameter tractability with respect to the parameter solution size and several parameters measuring the input graph's tree-likeness (see Table 1 in the introductory section for an overview). There remain numerous opportunities for future research. For example, the fixed-parameter tractability results for Min-Degree Deletion for the parameter treewidth as well as for the parameter feedback vertex set are far from any practical relevance. For these parameterizations it would be interesting to complement our classification results by direct combinatorial algorithms. Moreover, we are not aware of studies concerning the polynomial-time approximability of both problems.

References

1. Balasundaram, B., Butenko, S., Hicks, I.V.: Clique relaxations in social network analysis: The maximum k-plex problem. Oper. Res. (2010) (to appear)
2. Betzler, N., Uhlmann, J.: Parameterized complexity of candidate control in elections and related digraph problems. Theor. Comput. Sci. 410(52), 5425–5442 (2009)
3. Bodlaender, H., Downey, R., Fellows, M., Hermelin, D.: On problems without polynomial kernels. J. Comput. System Sci. 75(8), 423–434 (2009)
4. Bodlaender, H.L.: Kernelization: New upper and lower bound techniques. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 17–37. Springer, Heidelberg (2009)
5. Bodlaender, H.L., Thomassé, S., Yeo, A.: Analysis of data reduction: Transformations give evidence for non-existence of polynomial kernels. Technical Report UU-CS-2008-030, Department of Information and Computing Sciences, Utrecht University (2008)
6. Courcelle, B.: Graph rewriting: An algebraic and logic approach. In: Handbook of Theoretical Computer Science. Formal Models and Sematics (B), vol. B, pp. 193–242 (1990)
7. Dom, M., Lokshtanov, D., Saurabh, S.: Incompressibility through colors and IDs. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 378–389. Springer, Heidelberg (2009)
8. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
9. Faliszewski, P., Hemaspaandra, E., Hemaspaandra, L.A., Rothe, J.: Llull and Copeland voting computationally resist bribery and constructive control. J. Artifical Intelligence Res. 35(1), 275–341 (2009)
10. Fellows, M.R., Guo, J., Moser, H., Niedermeier, R.: A generalization of Nemhauser and Trotter's local optimization theorem. In: Proc. 26th STACS, pp. 409–420. IBFI Dagstuhl, Germany (2009)
11. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
12. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. ACM SIGACT News 38(1), 31–45 (2007)
13. Huberman, B.A., Romero, D.M., Wu, F.: Social networks that matter: Twitter under the microscope. First Monday 14(1) (2009)
14. Moser, H., Niedermeier, R., Sorge, M.: Algorithms and experiments for clique relaxations—finding maximum s-plexes. In: Proc. 8th SEA. LNCS, vol. 5526, pp. 233–244. Springer, Heidelberg (2009)
15. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
16. Potterat, J., Phillips-Plummer, L., Muth, S., Rothenberg, R., Woodhouse, D., Maldonado-Long, T., Zimmerman, H., Muth, J.: Risk network structure in the early epidemic phase of HIV transmission in Colorado Springs. Sexually Transmitted Infections 78(suppl. 1), 159–163 (2002)
17. Romm-Livermore, C., Setzekorn, K.: Social Networking Communities and E-Dating Services: Concepts and Implications. Information Science Reference (2008)
18. Wasserman, S., Faust, K.: Social Network Analysis: Methods and Applications. Cambridge University Press, Cambridge (1994)

Randomized OBDDs for the Most Significant Bit of Multiplication Need Exponential Size

Beate Bollig* and Marc Gillé

LS2 Informatik, TU Dortmund,
44221 Dortmund, Germany

Abstract. Integer multiplication as one of the basic arithmetic functions has been in the focus of several complexity theoretical investigations and ordered binary decision diagrams (OBDDs) are one of the most common dynamic data structures for Boolean functions. Only two years ago, the question whether the deterministic OBDD complexity of the most significant bit of integer multiplication is exponential has been answered affirmatively. Since probabilistic methods have turned out to be useful in almost all areas of computer science, one may ask whether randomization can help to represent the most significant bit of integer multiplication in smaller size. Here, it is proved that the randomized OBDD complexity is also exponential.

Keywords: computational complexity, integer multiplication, lower bounds, ordered binary decision diagrams, randomized one-round communication complexity.

1 Introduction

Integer multiplication is one of the most important functions in computer science and a lot of effort has been spent in designing good algorithms and small circuits and in determining its complexity. For some computation models it is a quite simple function. It is contained in NC^1 (polynomial-size $\{\vee, \wedge, \neg\}$-circuits of fan-in 2 and logarithmic depth) and in $TC^{0,3}$ (polynomial-size threshold circuits of depth 3) but neither in AC^0 (polynomial-size $\{\vee, \wedge, \neg\}$-circuits of unbounded fan-in and constant depth) nor in $TC^{0,2}$ [14]. For more than 35 years the algorithm of Schönhage-Strassen [22] has been the fastest method for integer multiplication running in time $O(n \log n \log \log n)$. Recently algorithms running in time $n \log n \cdot 2^{O(\log^* n)}$ have been presented [12,13]. Until now it is open whether there exist algorithms with running time $O(n \log n)$ for integer multiplication.

Ordered binary decision diagrams (OBDDs), introduced by Bryant [10], are one of the most common dynamic data structures for Boolean functions. Although many exponential lower bounds on the OBDD size of Boolean functions are known and the lower bound methods are quite simple, it is often a more difficult task to prove large lower bounds for some predefined and interesting

* Supported in part by DFG BO 2755/1-1.

functions. Although an exponential lower bound on the OBDD size of the so-called middle bit of integer multiplication has been known since 1991 [11], only two years ago Wegener's question [25], whether the OBDD complexity of the most significant bit of integer multiplication is also exponential, could be answered affirmatively [4].

The complexity theoretical analysis of randomized OBDDs has been launched in [1] by presenting a function representable by randomized OBDDs of polynomial size with small one-sided error (for the formal definition of the model see Section 2) but exponential size for deterministic OBDDs. Independently, in [2] and [20] the first lower bounds for randomized OBDDs have been proven. In [3] it has been shown that the size of randomized ordered binary decision diagrams representing the middle bit of integer multiplication with two-sided bounded error is at least $2^{\Omega(n/\log n)}$. In this paper we show that the size of randomized ordered binary decision diagrams for the most significant bit of integer multiplication is $2^{\Omega(n)}$ using a simple proof. As a by-product the lower bound on the size of randomized ordered binary decision diagrams for the middle bit of integer multiplication can also be improved up to $2^{\Omega(n)}$.

1.1 Ordered Binary Decision Diagrams

Boolean circuits, formulae, and binary decision diagrams, in complexity theory often called branching programs, are standard representations for Boolean functions. (For a history of results on binary decision diagrams see, e.g., the monograph of Wegener [25]). Besides the complexity theoretical viewpoint people have used restricted binary decision diagrams in applications and ordered binary decision diagrams are nowadays one of the most popular data structures for Boolean functions. Among the many areas of application are verification, model checking, computer-aided design, relational algebra, and symbolic graph algorithms.

Definition 1. *A* binary decision diagram (BDD) *on the variable set $X_n = \{x_1, \dots, x_n\}$ is a directed acyclic graph with one source and two sinks labeled by the constants 0 and 1. Each non-sink node (or decision node) is labeled by a Boolean variable and has two outgoing edges, one labeled by 0 and the other by 1. An input $b \in \{0,1\}^n$ activates all edges consistent with b, i.e., the edges labeled by b_i which leave nodes labeled by x_i. A* computation path *for an input b in a* BDD *G is a path of edges activated by the input b which leads from the source to a sink. A computation path for an input b which leads to the 1-sink is called* accepting path *for b. The* BDD *G represents a function $f \in B_n$ for which $f(b) = 1$ iff there exists an accepting path for the input b. The* size *of a binary decision diagram G is the number of its nodes. The* binary decision diagram size *of a Boolean function f is the size of the smallest* BDD *representing f.*

Definition 2. *Let $X_n = \{x_1, \dots, x_n\}$ be a set of Boolean variables. A variable ordering π on X_n is a permutation on $\{1, \dots, n\}$ leading to the ordered list $x_{\pi(1)}, \dots, x_{\pi(n)}$ of the variables. An* ordered binary decision diagram (OBDD) *is a binary decision diagram with a* variable ordering*. On each path from the source* *to the sinks, the variables at the nodes have to appear in the order prescribed by* π (*where some variables may be left out*). *A* π-OBDD *is an* OBDD *ordered according to* π. *The* π-OBDD *size of* f (*denoted by* $\pi\text{-OBDD}(f)$) *is the size of the smallest* π-OBDD *representing* f. *The* OBDD complexity *of* f (*denoted by* $\text{OBDD}(f)$) *is the minimum of all* $\pi\text{-OBDD}(f)$.

It is well known that the size of an OBDD representing a function f, that is defined on n Boolean variables and depends essentially on all of them (a function g depends essentially on a variable z if $g_{|z=0} \neq g_{|z=1}$), depends on the chosen variable ordering and may vary between linear and exponential size.

Randomized binary decision diagram models can be defined like randomized algorithms for decision problems.

Definition 3. *A* randomized binary decision diagram G *is a directed acyclic graph with decision nodes for Boolean variables and randomized nodes. A randomized node is an unlabeled node with two outgoing edges. Sinks can be labeled by* 0, 1, *or ?. The* random computation path *for an input* b *is defined as follows. At decision nodes labeled by* x_i, *the outgoing* b_i*-edge is chosen. At randomized nodes each outgoing edge is chosen independently from all other random decisions with probability* 1/2. *The* acceptance probability $\text{acc}_G(b)$ *or* $Prob(G(b) = 1)$ *of* G *on* b *is the probability that the random computation path reaches the* 1*-sink. The* rejection probability $\text{rej}_G(b)$ *or* $Prob(G(b) = 0)$ *of* g *on* b *is the probability that the random computation path reaches the* 0*-sink.*

Randomized variants of restricted binary decision diagrams can be defined similarly. Another approach to define randomized binary decision diagrams is the introduction of probabilistic variables in addition to the usual Boolean variables (see [1]). The input is an assignment to the usual variables where the probabilistic variables independently take the values 0 and 1 with probability 1/2. If each probabilistic variables is tested at most once on each path from the source to the sinks, we obtain an equivalent definition of randomized binary decision diagrams.

Definition 4. *i) G represents a function* $f \in B_n$ *with* unbounded error *if* $Prob(G(b) = f(b)) > 1/2$ *for all inputs* b.

ii) G represents a function $f \in B_n$ *with* two-sided ϵ-bounded error, $0 \leq \epsilon < 1/2$, *if* $Prob(G(b) \neq f(b)) \leq \epsilon$ *for all inputs* b.

iii) G represents a function $f \in B_n$ *with* one-sided ϵ-bounded error, $0 \leq \epsilon < 1$, *if* $Prob(G(b) \neq f(b)) \leq \epsilon$ *for all inputs* $b \in f^{-1}(1)$ *and* $Prob(G(b) = 0) = 1$ *for all inputs* $b \in f^{-1}(0)$.

iv) G represents a function $f \in B_n$ *with* zero error *and* ϵ-failure, $0 \leq \epsilon < 1$, *if* $Prob(G(b) = \overline{f}(b)) = 0$ *and* $Prob(G(b) =?) \leq \epsilon$ *for all inputs* b.

1.2 Integer Multiplication and Ordered Binary Decision Diagrams

Lower bounds for integer multiplication are motivated by the general interest in the complexity of important arithmetic functions.

Definition 5. *The Boolean function* $\mathrm{MUL}_{i,n} \in B_{2n}$ *maps two* n*-bit integers* $x = x_{n-1} \ldots x_0$ *and* $y = y_{n-1} \ldots y_0$ *to the* i*th bit of their product , i.e.,* $\mathrm{MUL}_{i,n}(x, y) = z_i$*, where* $x \cdot y = z_{2n-1} \ldots z_0$ *and* x_0, y_0, z_0 *denote the least significant bits.*

The first exponential lower bound on the OBDD complexity for an output bit of integer multiplication has been proved for $\mathrm{MUL}_{n-1,n}$. Bryant [11] has presented a lower bound of $2^{n/8}$. Progress in the analysis of the middle bit of integer multiplication has been achieved by an approach using universal hashing and as a result Woelfel [26] has improved Bryant's lower bound up to $\Omega(2^{n/2})$. In the meantime exponential lower bounds for the middle bit of multiplication have also been proved for more general binary decision diagram models (see, e.g., [9,8], [19], and [21]). As mentioned above, in [3] it has been shown that the size of randomized ordered binary decision diagrams representing $\mathrm{MUL}_{n-1,n}$ with ϵ-bounded error, where $\epsilon < 1/2$ is a constant, is bounded below by $2^{\Omega(n/\log n)}$. In [21] a lower bound for so-called read-k-times branching programs representing $\mathrm{MUL}_{n-1,n}$ has been shown which is superpolynomial as long as the error probability is superpolynomially small. (For more about integer multiplication and the size of BDDs see e.g. [5].)

$\mathrm{MUL}_{2n-1,n}$ computes the most important bit of integer multiplication in the following sense. Let $(z_{2n-1}, \ldots, z_0)$ be the binary representation of the integer z, i.e., $z = \sum_{i=0}^{2n-1} z_i \cdot 2^i$. Since the bit z_{2n-1} has the highest significance, for the approximation of the value of the product of two n-bit numbers x and y it is the most important one. Moreover, for space bounded models of computation the most significant bit of integer multiplication is interesting since lower bounds of order $s(n)$ can be transferred to lower bounds of order $s(i/2)$ for any other bit z_i, $2n - 1 > i \geq 0$. As mentioned before, only in 2008 it has been shown that the OBDD complexity of the most significant bit of multiplication is exponential [4]. (For larger lower bounds see also [6,7].)

Our result can be summarized as follows.

Theorem 1. *The size of randomized* OBDDs *representing the most significant bit of integer multiplication* $\mathrm{MUL}_{2n-1,n}$ *with bounded two-sided error is at least* $2^{\Omega(n)}$.

In Section 2 we start with some notation. Since results on randomized one-round communication complexity provide useful tools to prove lower bounds on the size of randomized OBDDs, we present afterwards some basics concerning communication complexity. Our lower bound proof on the size of randomized OBDDs for the most significant bit of integer multiplication appeals to known results for the communication complexity of the greater than function GT_n, therefore we take a look at the round-elimination technique and the lower bounds for GT_n. Finally, Section 3 contains the main result of the paper, the proof of Theorem 1.

2 Preliminaries

2.1 Notation

In the rest of the paper we use the following notation.

Let $[x]^l_r$, $n-1 \geq l \geq r \geq 0$, denote the bits $x_l \ldots x_r$ of a binary number $x = (x_{n-1}, \ldots, x_0)$. For the ease of description we use the notation $[x]^l_r = z$ if $(x_l, \ldots, x_r)$ is the binary representation of the integer $z \in \{0, \ldots, 2^{l-r+1}-1\}$. Sometimes, we identify $[x]^l_r$ with z if the meaning is clear from the context. We use the notation $(z)^l_r$ for an integer z to identify the bits at position $l, \ldots, r$ in the binary representation of z.

Let $\ell \in \{0, \ldots, 2^m - 1\}$, then $\overline{\ell}$ denotes the number $(2^m - 1) - \ell$.

2.2 Communication Complexity

In order to obtain lower bounds on the size of OBDDs one-round communication complexity has become a standard technique (see [15] and [17] for the theory of communication complexity). Loosely speaking, the known lower bound proofs on the size of OBDDs are based on the fact that a large amount of information has to be exchanged across a suitably chosen cut in the OBDD in order to evaluate the represented function. Results from communication complexity are more or less used to get lower bounds on the necessary amount of information.

The main subject is the analysis of the following (restricted) communication game. Consider a Boolean function $f \in B_n$ which is defined on the variables in $X_n = \{x_1, \ldots, x_n\}$, and let $\Pi = (X_A, X_B)$ be a partition of X_n. Assume that Alice has only access to the input variables in X_A and Bob has only access to the input variables in X_B. In a one-round communication protocol, upon a given input x, Alice is allowed to send a single message (depending on the input variables in X_A) to Bob who must then be able to compute the answer $f(x)$. The *one-round communication complexity* of the function f denoted by $C(f)$ is the worst case number of bits of communication which need to be transmitted by such a protocol that computes f. It is easy to see that an OBDD G with respect to a variable ordering where the variables in X_A are tested before the variables in X_B can be transformed into a communication protocol and $C(f) \leq \lceil \log |G| \rceil$. Therefore, linear lower bounds on the communication complexity of a function $f : \{0,1\}^{|X_A|} \times \{0,1\}^{|X_B|} \rightarrow \{0,1\}$ lead to exponential lower bounds on the size of π-OBDDs where the X_A-variables are before the X_B-variables in π. Similarly, randomized communication complexity (or more precisely communication with randomized protocols) provide the main tool to prove lower bounds on the size of randomized OBDDs. (For a nice paper on randomized one-round communication complexity see [16]). A randomized OBDD G' can be transformed into a randomized communication protocol with the same error or failure guarantee and $R(f) \leq \lceil \log |G'| \rceil$ for the randomized communication complexity $R(f)$ of the given type of one-round protocol.

Usually we cannot directly analyze the communication complexity of a function. One way out is the identification of hard subproblems. Rectangular reductions allow to apply known results on (randomized) communication complexity to prove results on the size of randomized OBDDs in a simple way.

Definition 6. *Let $f(x,y) : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}$ and $g(x,y) : \{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}$. A pair (φ_A, φ_B) of functions $\varphi_A : \{0,1\}^n \rightarrow \{0,1\}^k$ and $\varphi_B : \{0,1\}^m \rightarrow \{0,1\}^l$ is called* rectangular reduction *from f to g if $f(a,b) = g(\varphi_A(a), \varphi_B(b))$ for all $(a,b) \in \{0,1\}^n \times \{0,1\}^m$.*

The obvious and important property of a rectangular reduction from a function f to a function g is that the communication complexity of f is bounded above by the communication complexity of g and this holds for each type of protocol (deterministic and randomized).

Next, we look at the randomized communication complexity of some functions and the ideas for the lower bound proofs of the size of (randomized) OBDDs for the output bits of integer multiplication. Let the greater than function $\mathrm{GT}_n : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}$ be defined by $\mathrm{GT}_n(a,b) = 1$ iff $[a]_1^n > [b]_1^n$. In [18,23,24] rounds versus communication tradeoffs for GT_n have been studied. An r-round protocol is a protocol with the restriction that Alice and Bob may exchange at most r messages, where Alice must send the first message and only one of the players need output the answer. $R^r(f)$ is the number of bits in the largest message, minimized over all r-round protocols that output f correctly with the given bounded error or failure probability. A lower bound of $R^r(\mathrm{GT}_n) = \Omega(n^{1/r}c^{-r})$ for some fixed constant c and a nearly matching upper bound of $O(n^{1/r}\log n)$ are known [18,23] which leads to linear randomized one-round communication complexity for the greater than function. Intuitively speaking, the round elimination technique pioneered in [18] and refined in [23] enables Alice and Bob to remove the first round of communication in a protocol and to solve a somewhat smaller instance of the same problem. Here, a problem $\mathrm{GT}_{n/k}^{k,A}(a^1, \ldots, a^k, i, b)$ is defined, where Alice is given $a^1, \ldots, a^k \in \{0,1\}^{n/k}$ and Bob is given i, $i \in \{1, \ldots, k\}$, $b \in \{0,1\}^{n/k}$ and copies of $a^1, \ldots, a^{i-1}$, and they have to communicate and decide whether $a_i > b$. To reduce $\mathrm{GT}_{n/k}^{k,A}$ to GT_n, Alice concatenates $a^1, \ldots, a^k$ to a and Bob concatenates $a^1, \ldots, a^{i-1}, b, 1 \ldots 1$ to $b' \in \{0,1\}^n$. Now, using the round elimination lemma the argumentation is the following. If there exists an one-round δ-bounded error (public coin) randomized protocol for the modified greater than function with communication complexity c', then a zero-round protocol for the greater than function $\mathrm{GT}_{n/k}$ can be obtained with error probability $\delta + 1/2(2c' \ln 2/n)^{1/2}$. Since $n/k > 1$, $\mathrm{GT}_{n/k}$ is defined on a nontrivial domain, the function cannot be solved by a zero-round protocol with error probability less than $1/2$. Therefore, $c' = \Omega(n)$. It is easy to prove that the same lower bound for a modification of GT_n, where Alice gets exactly one of the variables a_i and b_i, $1 \leq i \leq n$, can be obtained using a rectangular reduction.

Binary addition $\mathrm{ADD}_{i,n} : \{0,1\}^{2n} \rightarrow \{0,1\}$ maps two n-bit integers $x = x_{n-1} \ldots x_0$ and $y = y_{n-1} \ldots y_0$ to the ith bit of their sum, i.e., $\mathrm{ADD}_{i,n}(x,y) = s_i$, where $s_n \ldots s_0$ is the sum of x and y and x_0, y_0, z_0 denote the least significant bits. It is easy to see that the communication complexity of $\mathrm{ADD}_{n,n}$ is linear if for each i, $0 \leq i \leq n-1$, Alice gets exactly one of the variables x_i and y_i. $\mathrm{ADD}_{n,n}(x,y) = 0$ iff $[x]_1^n < 2^n - [y]_1^n$. Therefore, the functions $\mathrm{ADD}_{n,n}$ and GT_n are closely related, to be more precise, there is a rectangular reduction from GT_n to $\mathrm{ADD}_{n,n}$ and vice versa. The idea of Bryant's lower bound proof on the OBDD size of $\mathrm{MUL}_{n-1,n}$ [11] is the following. For each variable ordering there is a subfunction of $\mathrm{MUL}_{n-1,n}$ which is equal to $\mathrm{ADD}_{m,m}$, where $m \geq n/8$. The variable ordering is bad in the sense that among Alice's m variables is exactly one of the variables x_i and y_i. As a result we can conclude that the size of randomized OBDDs representing $\mathrm{MUL}_{n-1,n}$ is at least $2^{\Omega(n)}$. For the proof of the lower bound on the size of randomized OBDDs for $\mathrm{MUL}_{2n-1,n}$ in the next section, we use some of the ideas for the exponential lower bound on the size of deterministic OBDDs for the most significant bit of integer multiplication [4,7].

Definition 7. *Let $f : \{0,1\}^n \rightarrow \{0,1\}$ be an arbitrary function defined on the variable set $X_n = \{x_1, \ldots, x_n\}$. Let $\pi : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$ be a variable ordering on X_n, $1 \leq p \leq n-1$, and $L := \{x_{\pi(1)}, \ldots, x_{\pi(p)}\}$, $R := \{x_{\pi(p+1)}, \ldots, x_{\pi(n)}\}$. The function $f^{\pi,p} : \{0,1\}^p \times \{0,1\}^{n-p} \rightarrow \{0,1\}$ is defined on assignments l to the variables in L and r to R by $f^{\pi,p}(l,r) := f(l+r)$, where $l+r$ denotes the joint assignment to X_n obtained from l and r.*

Our aim is to prove that there exists for every variable ordering π a rectangular reduction from the variant of $\mathrm{GT}_{n'}$, where Alice gets exactly one variable for each significance and $n' = \Theta(n)$, or the negation of this function to $\mathrm{MUL}_{2n-1,n}^{\pi,p}$ for p suitably chosen.

3 An Exponential Lower Bound on the Randomized OBDD Complexity of the Most Significant Bit of Integer Multiplication

In this section we prove Theorem 1 and determine the lower bound of $2^{\Omega(n)}$ on the size of randomized OBDDs for the representation of the most significant bit of integer multiplication. We use some of the ideas presented in [7] but we have to apply them differently to obtain a lower bound on the size of randomized OBDDs.

Let π be an arbitrary but fixed variable ordering in the rest of the sequel. We start with a useful observation.

Fact 1. *For an integer $2^{n-1} + \ell 2^{n/3}$ the corresponding smallest integer such that the product of the two numbers is at least 2^{2n-1} is*

$$2^n - \ell 2^{n/3+1} + \left\lceil \ell^2 2^{-n/3+2} - \frac{4\ell^3}{2^{n-1} + \ell 2^{n/3}} \right\rceil .$$

Next, we investigate requirements that have to be fulfilled for inputs x and y, where $\mathrm{MUL}_{2n-1,n}(x,y) = 1$. If x represents an integer $2^{n-1} + \ell 2^{n/3}$, $1 \leq \ell < 2^{n/3-1}$, and $[y]_0^{n/3} = 0$, the upper part of y has to represent an integer greater

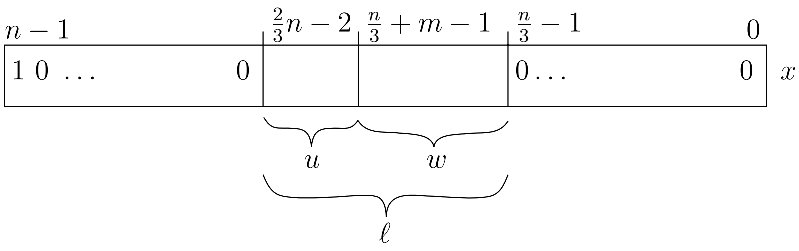


Fig. 1. The composition of the input x

than $2^{(2/3)n} - 2\ell$, i.e., $[y]_{n/3}^{n-1} > 2^{(2/3)n} - 2\ell$. If $[x]_0^{n-1} = 2^{n-1} + \ell 2^{n/3}$ and $[y]_{n/3}^{n-1} = 2^{(2/3)n} - 2\ell$, the lower part of y has to represent an integer of at least $\left\lceil \ell^2 2^{-n/3+2} - \frac{4\ell^3}{2^{n-1}+\ell 2^{n/3}} \right\rceil$.

In the following we decompose ℓ into two parts:

$$\ell := u \cdot 2^m + w,$$

$w < 2^m$ and $u < 2^{(7/8)m}$ and $m := (8/45)n - 8/15$. (In the sequel for the sake of simplicity we do not apply floor or ceiling functions to numbers even when they need to be integers whenever this is clear from the context and has no bearing on the essence of the proof.) Then $\ell^2 = u^2 2^{2m} + uw2^{m+1} + w^2$. In our lower bound proof we choose u and w in such a way that no carry bit is generated by adding w^2 and $uw2^{m+1}$. For our rectangular reduction the most decisive part is $uw2^{m+1}$. If we choose u as a power of 2, the product $u \cdot w$ is equal to w shifted according to u. Moreover, one key idea of our lower bound proof is that

$$\begin{aligned}(\ell^2)_{2m+1-n/3+2}^{(5/2)m-n/3+2} &= \left(\left\lceil \ell^2 2^{-n/3+2} - \frac{4\ell^3}{2^{n-1}+\ell 2^{n/3}} \right\rceil\right)_{2m+1-n/3+2}^{(5/2)m-n/3+2} \\ &= \left(u^2 2^{-1} + u \cdot w\right)_m^{(3/2)m-1}\end{aligned}$$

if we fix u and choose the assignments for w suitably. Therefore, $[y]_0^{n/3-1} \geq \left\lceil \ell^2 2^{-n/3+2} - \frac{4\ell^3}{2^{n-1}+\ell 2^{n/3}} \right\rceil$ if $[y]_{2m+1-n/3+2}^{(5/2)m-n/3+2} \geq \left(u^2 2^{-1} + u \cdot w\right)_m^{(3/2)m-1}$. (See Figure 1 for the decomposition of our inputs x in the lower bound proof.)

Let $S := \{x_{n/3+m/2}, \ldots, x_{n/3+m-1}, y_{(5/2)m-n/3+2}, \ldots, y_{2m+1-n/3+2}\}$ and L be the set of the first $|L|$ variables according to π where there are $m/2$ variables from S and R be the set of the remaining variables. Let $X_{S,L}$ be the x-variables in $S \cap L$, $X_{S,R}$ the x-variables in $S \cap R$. Similar the sets $Y_{S,L}$ and $Y_{S,R}$ are defined. Using simple counting arguments we can prove that there exists a distance parameter d such that there are at least $m/8$ pairs $(x_{n/3+i}, y_{m+1+i+d-n/3+2})$ in $X_{S,L} \times Y_{S,R} \cup X_{S,R} \times Y_{S,L}$ (for a similar proof see, e.g., [11]). Let I be the set of indices, where $x_{n/3+i}$ belongs to such a pair.

Next, we have to use a case inspection. A pair $(x_{n/3+i}, y_{n/3+i+1})$, $i \in I$, is called (x, y)-pair. A pair (x_i, y_{i+1}) is called separated with respect to L iff $x_i \in L$ and $y_{i+1} \notin L$ or vice versa.

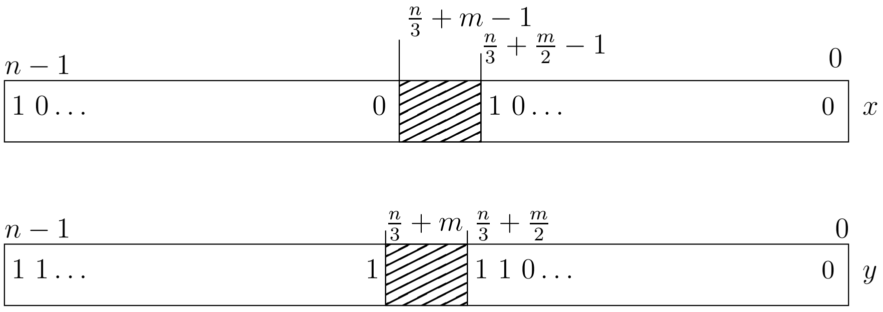


Fig. 2. Some replacements of the x- and y-variables in Case 1. The free x-and y-variables are in the shaded areas.

Case 1: There are at least $|I|/2$ separated (x, y)-pairs with respect to L.

Let $I' \subseteq I$ be the set of indices i such that $x_{n/3+i}$ belongs to a separated (x, y)-pair with respect to L. The variables $x_{n/3+i}, y_{n/3+i+1}$, $i \in I'$, are called free x- and y-variables. We replace the variables that are not free in the following way (see Figure 2 for some of the replacements mentioned below):

– x_{n-1} and $x_{n/3+m/2-1}$ are set to 1, the remaining x-variables that are not free are set to 0,
– $y_{n/3+m/2}$ and $y_{n/3+m/2-1}$ are set to 1, $y_{n/3+m/2-2}, \ldots, y_0$ are set to 0, the remaining y-variables that are not free are set to 1.

The effect of these replacements is that the function value of the corresponding subfunction of the most significant bit of integer multiplication is 1 if the vector that consists of the free x-variables is at least as large than the vector that consists of the negated free y-variables. With other words there is a rectangular reduction from the variant of the function $\overline{\mathrm{GT}}_{|I'|}$, where Alice has exactly one bit for each significance, to $\mathrm{MUL}^{\pi,|L|}_{2n-1,n}$.

Case 2: There are less than $|I|/2$ separated (x, y)-pairs with respect to L.

Let $I' \subseteq I$ be the set of indices i such that $x_{n/3+i}$ belongs to a separated (x, y)-pair with respect to L and $I'' := I \setminus I'$. The variables $x_{n/3+i}, y_{n/3+i+1}$, $y_{m+1+i+d-n/3+2}$, $i \in I''$, are called free x- and y-variables. We replace some of the variables that are not free in the following way (see Figure 1 for the composition of the input x and Figure 3 for some of the replacements mentioned below):

– x_{n-1}, $x_{n/3+m+d}$, and $x_{n/3}$ are set to 1, the remaining x-variables that are not free are set to 0,
– $y_{n/3+m+d+1}$ and $y_{n/3}$ are set to 0, the remaining variables in $\{y_{n-1}, \ldots, y_{n/3}\}$ that are not free are set to 1,
– $y_{2m+2d-n/3+2}$ and $y_{2m-n/3+2}, \ldots, y_0$ are set to 1, the remaining y-variables that are not free are set to 0.

Now, it is not difficult to see that there is a rectangular reduction from the variant of the function $\overline{\mathrm{GT}}_{|I''|}(a, b)$, where Alice has exactly one bit for each

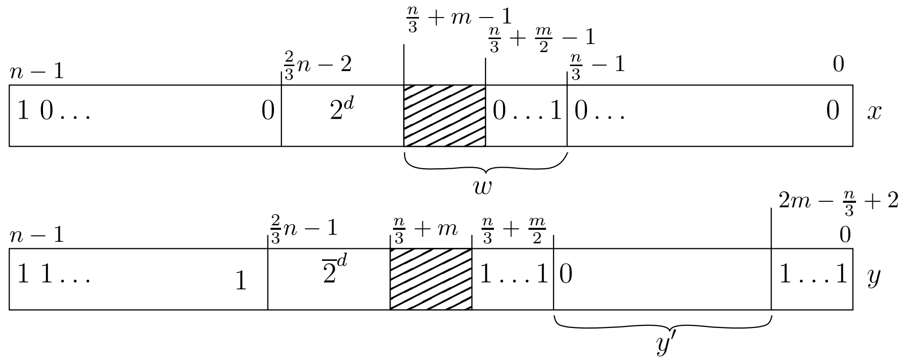


Fig. 3. Some replacements of the x- and y-variables in Case 2. The function value of the corresponding subfunction of $\mathrm{MUL}_{2n-1,n}(x,y)$ is 1 iff the subvector y' represents an integer that is at least $u^2 \cdot 2^{-1} + (u \cdot w)$ div $2^m = 2^{2d-1} + (2^d \cdot w)$ div 2^m (if $x_{n/3+i} = \overline{y}_{n/3+i+1}$, for all $i \in I''$).

significance, to $\mathrm{MUL}^{\pi,|L|}_{2n-1,n}$: assignments to the a-variables are set to the same assignments to the variables $x_{n/3+i}$, $i \in I''$, the variables $y_{n/3+i+1}$ are set to the negated assignments to $x_{n/3+i}$, and the assignments to the b-variables are set to the same assignments to the variables $y_{m+1+i+d-n/3+2}$.

4 Concluding Remarks

The complexity of integer multiplication is a fascinating subject, although we have already learned in primary school how to multiply integers. For the middle bit of multiplication exponential lower bounds for more general (non-oblivious) binary decision diagram models are known for a long time ([19]) but nothing is known for the most significant bit. Intuitively $\mathrm{MUL}_{2n-1,n}$ seems to be easier to compute than $\mathrm{MUL}_{n-1,n}$ but does there really exists a (non-oblivious) binary decision diagram model for which the computational complexity of the two functions is exponentially different?

References

1. Ablayev, F.: Randomization and nondeterminism are incomparable for ordered read-once branching programs. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 195–202. Springer, Heidelberg (1997)
2. Ablayev, F., Karpinski, M.: On the power of randomized ordered branching programs. In: Meyer auf der Heide, F., Monien, B. (eds.) ICALP 1996. LNCS, vol. 1099, pp. 348–356. Springer, Heidelberg (1996)
3. Ablayev, F., Karpinski, M.: A lower bound for integer multiplication on randomized ordered read-once branching programs. Information and Computation 186(1), 78–89 (2003)
4. Bollig, B.: On the OBDD complexity of the most significant bit of integer multiplication. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 306–317. Springer, Heidelberg (2008)

5. Bollig, B.: Integer multiplication and the complexity of binary decision diagrams. EATCS Bulletin 98, 78–106 (2009)
6. Bollig, B.: Larger lower bounds on the OBDD complexity of integer multiplication. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 212–223. Springer, Heidelberg (2009)
7. Bollig, B.: A larger lower bound on the OBDD complexity of the most significant bit of multiplication. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 255–266. Springer, Heidelberg (2010)
8. Bollig, B., Waack, S., Woelfel, P.: Parity graph-driven read-once branching programs and an exponential lower bound for integer multiplication. Theoretical Computer Science 362, 86–99 (2006)
9. Bollig, B., Woelfel, P.: A read-once branching program lower bound of $\Omega(2^{n/4})$ for integer multiplication using universal hashing. In: Proc. of 33rd STOC, pp. 419–424 (2001)
10. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. on Computers 35, 677–691 (1986)
11. Bryant, R.E.: On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. IEEE Trans. on Computers 40, 205–213 (1991)
12. De, A., Kurur, P., Saha, C., Saptharishi, R.: Fast integer multiplication using modular arithmetic. In: Proc.of 40th STOC, pp. 499–506 (2008)
13. Fürer, M.: Faster integer multiplication. In: Proc. of 39th STOC, pp. 57–66 (2007)
14. Hajnal, A., Maass, W., Pudlák, P., Szegedy, M., Turán, G.: Threshold circuits of bounded depth. In: Proc. 28th FOCS 1987, pp. 99–110 (1987)
15. Hromkovič, J.: Communication Complexity and Parallel Computing. Springer, Heidelberg (1997)
16. Kremer, I., Nisan, N., Ron, D.: On Randomized One-Round Communication Complexity. Computational Complexity 8(1), 21–49 (1999)
17. Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press, Cambridge (1997)
18. Miltersen, P.B., Nisan, N., Safra, S., Wigderson, A.: On data structures and asymmetric communication complexity. Journal of Computer and System Science 57(1), 37–49 (1998)
19. Ponzio, S.: A lower bound for integer multiplication with read-once branching programs. SIAM Journal on Computing 28, 798–815 (1998)
20. Sauerhoff, M.: Lower bounds for randomized read-k-times branching programs. In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 103–115. Springer, Heidelberg (1998)
21. Sauerhoff, M., Woelfel, P.: Time-space trade-off lower bounds for integer multiplication and graphs of arithmetic functions. In: Proc. of 35th STOC, pp. 186–195 (2003)
22. Schönhage, A., Strassen, V.: Schnelle Multiplikation großer Zahlen. Computing 7, 281–292 (1971)
23. Sen, P., Venkatesh, S.: Lower bounds for predecessor searching in the cell probe model. Journal of Computer and System Sciences 74(3), 364–385 (2008)
24. Smirnov, D.: Shannon's information methods for lower bounds for probabilistic communication complexity. Master's thesis, Moskow University (1988)
25. Wegener, I.: Branching Programs and Binary Decision Diagrams - Theory and Applications. SIAM Monographs on Discrete Mathematics and Applications (2000)
26. Woelfel, P.: New bounds on the OBDD-size of integer multiplication via universal hashing. Journal of Computer and System Science 71(4), 520–534 (2005)

GreedyMAX-type Algorithms for the Maximum Independent Set Problem

Piotr Borowiecki[1] and Frank Göring[2]

[1] Department of Algorithms and System Modeling,
Faculty of Electronics, Telecommunications and Informatics,
Gdańsk University of Technology, Narutowicza 11/12, 80-233 Gdańsk, Poland
pborowie@eti.pg.gda.pl
[2] Fakultät für Mathematik,
TU Chemnitz, D-09107 Chemnitz, Germany
frank.goering@mathematik.tu-chemnitz.de

Abstract. A maximum independent set problem for a simple graph $G = (V, E)$ is to find the largest subset of pairwise nonadjacent vertices. The problem is known to be NP-hard and it is also hard to approximate. Within this article we introduce a non-negative integer valued function p defined on the vertex set $V(G)$ and called a potential function of a graph G, while $P(G) = \max_{v \in V(G)} p(v)$ is called a potential of G. For any graph $P(G) \leq \Delta(G)$, where $\Delta(G)$ is the maximum degree of G. Moreover, $\Delta(G) - P(G)$ may be arbitrarily large. A potential of a vertex lets us get a closer insight into the properties of its neighborhood which leads to the definition of the family of `GreedyMAX`-type algorithms having the classical `GreedyMAX` algorithm as their origin. We establish a lower bound $1/(P + 1)$ for the performance ratio of `GreedyMAX`-type algorithms which favorably compares with the bound $1/(\Delta + 1)$ known to hold for `GreedyMAX`. The cardinality of an independent set generated by any `GreedyMAX`-type algorithm is at least $\sum_{v \in V(G)} (p(v)+1)^{-1}$, which strengthens the bounds of Turán and Caro-Wei stated in terms of vertex degrees.

Keywords: independent set, stable set, graph algorithm, greedy algorithm, ordering, potential of a graph.

1 Introduction

The maximum independent set (MIS) problem is one of the fundamental problems of discrete optimization. It gained a significant interest as well in theoretical investigations as in the context of applications ranging from parallel computing, image processing to data mining and database design. Within this paper we consider MIS for finite, simple and undirected graphs $G = (V, E)$ with vertex set V, edge set E, order $n = |V(G)|$ and size $m = |E(G)|$. A set of vertices I, $I \subseteq V(G)$ is *independent* in G, if no two vertices of I are adjacent. The MIS problem is to find an independent set with the goal of maximizing its cardinality.

The maximum cardinality of an independent set in G is called the *independence number* $\alpha(G)$ of a graph G.

In context of applications a graph G is usually a conflict graph with vertices representing desirable objects and edges uv to express that objects u and v are in conflict, i.e. at most one of them can be selected. The goal is to select as many desirable objects as possible while never selecting two conflicting objects. Examples for applications of MIS problem are, e.g. disjoint paths for network routing [7], interval scheduling in manufacturing processes [26], map labeling [1] and frequency assignment [21] to mention just a few of them.

MIS problem is known to be NP-hard [9] and it is also hard to approximate within $n^{1-\varepsilon}$ [17]. Several exponential time exact algorithms for MIS were given in literature with the current best running time $O(2^{0.288n})$ [8]. A lot of effort was put as well into establishing the bounds on $\alpha(G)$ as into design and analysis of approximation algorithms (see e.g. [3,4,5,10,11,13,14,15,16,18,22,23,24,25,27,28] to mention just a few of them). The readers interested in more details are also referred to [12,19] for surveys.

In what follows we need the notion of the *neighborhood* of a vertex v in the graph G defined as a set $N_G(v) = \{u \in V(G) \mid vu \in E(G)\}$. A set $N_G[v] = N_G(v) \cup \{v\}$ is called a *closed neighborhood* of v in G. The *degree* of vertex v in G is the number $d_G(v) = |N_G(v)|$ of its neighbors, while $\Delta(G)$ stands for the maximum vertex degree in graph G. One of the best known general bounds on the independence number

$$\alpha(G) \geq CW(G) = \sum_{v \in V(G)} \frac{1}{d_G(v) + 1} \tag{1}$$

was independently given by Caro [5] and Wei [28] and it has been also proved using probabilistic methods, e.g. by Alon and Spencer [3] and Selkow [25]. Inequality (1) extends the classical result of Turán [27] $\alpha(G) \geq n/(\bar{d}(G) + 1)$ stated in terms of graph's order n and its average degree $\bar{d}(G)$. Griggs [10] and also Chvátal and McDiarmid [6] proved that the classical `GreedyMAX` algorithm, which selects a vertex of maximum degree, deletes it with all incident edges from the graph and iterates this process on the resulting graph until no edge remains, outputs an independent set of cardinality bounded by $CW(G)$.

In Section 2 we introduce a non-negative integer valued function p defined on the vertex set $V(G)$ called a potential function of a graph G and an invariant $P(G) = \max_{v \in V(G)} p(v)$ called a potential of a graph G. In contrast to Caro-Wei and Turán bounds, which strongly depend on counting the neighbors, the criteria relying on the potential function gives a closer insight into the properties of the neighborhoods. This leads to the definition of the family of `GreedyMAX`-type algorithms (see Sec. 3.1), which have their origin in the classical `GreedyMAX` algorithm, and lets us incorporate as well quantitative as qualitative aspects in algorithms analysis to obtain the improved bounds on their performance (see Sec. 3.2). In particular, since for any vertex $p_G(v) \leq d_G(v)$, a potential function is used to strengthen $CW(G)$ by proving that cardinality of an independent set produced by any `GreedyMAX`-type algorithm is at least

$CWP(G) = \sum_{v \in V(G)} (p(v)+1)^{-1}$. As a consequence, we obtain a lower bound $1/(P+1)$ for the performance ratio of `GreedyMAX`-type algorithms which favorably compares with the bound $1/(\Delta+1)$ known to hold for `GreedyMAX`.

2 Potential of a Graph

2.1 *k*-Sequences

Let $A \subset \mathbb{N}_0$ be a multiset. We say that the non-decreasing sequence $S = (s_1, \dots, s_k)$, $S \subseteq A$ is the *k-sequence in A*, if for each $i \in \{1, \dots, k\}$ it holds $s_i \geq i$ (if there is no ambiguity we also use k-sequences as if there were multisets). The k-sequence S is *maximal in A* if there does not exist $k_1 > k$ such that A contains a k_1-sequence. A maximal k-sequence with the largest sum of elements is called *maximum in A* and it is denoted by S^A_{max}, while S^A_{min} stands for k-sequence with the smallest sum which is called *minimum in A*. We say that the k-sequence S is *saturating* in A if $k = |A|$. Otherwise we say that S is *non-saturating.* It follows by maximality that for any k-sequence S maximal in A which is non-saturating in A there exists at least one element $s_i \in S$ such that $s_i = i$. Every element $s_i \in S$ for which $s_i = i$ is called a *blocking element* in S.

Lemma 1. *Let S and S' be any k-sequences maximal in A and let m, m' be the values of the largest blocking elements in S and S' respectively. Then $m = m'$.*

Proof. The lemma obviously holds when S and S' are saturating in A. Let $X = A \setminus S$, $M = \{s \in S \mid s \leq m\}$ and $L = S \setminus M$. The sets X', M' and L' for S' are defined analogously.

Assume that $m > m'$. If there existed $a \in L$ such that $a \notin L'$ then S' would not be maximal in A (take a $(k+1)$-sequence obtained from S' by inserting a just after m'). Hence $L \subseteq L'$. Let $C = L' \setminus L$. If C were not empty, then since m' is the largest blocking element in S', C would have to contain an element $c > m$. Obviously $C \subseteq X \cup M$ and if $X \cup M$ contained element $c > m$, S would not be maximal in A (take a $(k+1)$-sequence obtained from S by inserting c just after m). Therefore $m' \geq m$.

Since $m' \leq m$ follows by symmetry, we have $m' = m$. □

In what follows we use b^{max} to denote the value of the largest blocking element for k-sequences maximal in A. We also use e^{max} for the value of the largest element of $X = A \setminus S^A_{min}$, when $X \neq \emptyset$ and $e^{max} = 0$ if $X = \emptyset$.

Example 1. Let $A = \{1, 2, 3, 3, 3, 5, 5, 8\}$. Then $S^A_{max} = (3, 3, 3, 5, 5, 8)$, $b^{max} = 5$, while $S^A_{min} = (1, 2, 3, 5, 5, 8)$, $A \setminus S^A_{min} = \{3, 3\}$ and consequently $e^{max} = 3$.

Consider the two complementary lemmas on extending and shortening of k-sequences.

Lemma 2. *If $A' = A \cup \{t\}$, then every maximal $(k+1)$-sequence S' in A' contains t if and only if A contains a maximal k-sequence S such that exactly one of the following conditions is satisfied:*

(a) *S contains at least one blocking element and $t > b^{max}$,*
(b) *S does not contain blocking elements.*

Proof. ($\Rightarrow$) Let S' be a maximal $(k+1)$-sequence in A' and let $t \in S'$. Moreover, let $S' = (s'_1, \ldots, s'_i, t, s'_{i+2}, \ldots, s'_k, s'_{k+1})$ be such that t has the smallest index, say $i+1$. It follows that $i \leq s'_i < i+1 \leq t$. Hence elements of S' satisfy $s'_i = i$, $t \geq i+1$ and $s'_j \geq j$ for $j \in \{i+2, \ldots, k+1\}$. Now, consider the sequence $S = (s_1, s_2, \ldots, s_i, s_{i+1}, \ldots, s_{k-1}, s_k)$, where $s_j = s'_j$ for $j \in \{1, \ldots, i\}$ and $s_j = s'_{j+1}$ if $j \in \{i+1, \ldots, k\}$. Observe that all elements of S satisfy the following conditions: $s_j \geq j$ for $j \in \{1, \ldots, i\}$ with $s_i = i$ and $s_j = s'_{j+1} \geq j+1$ implying $s_j > j$ when $j \in \{i+1, \ldots, k\}$. It follows that s_i is the largest blocking element in S and $t > b^{max}$. If t has index one in S', then S does not contain blocking elements. Maximality of S follows by maximality of S'.

($\Leftarrow$) Assume that A contains a maximal k-sequence S containing no blocking elements. Then for every element $s_i \in S$, $i \in \{1, \ldots, k\}$ we have $s_i - i \geq 1$. Hence, every sequence S' obtained from S by inserting t, independently of index of t is a $(k+1)$-sequence in A'. Now, let $s_j \in S$ be the largest blocking element. Then similarly, for any $s_i \in S$, $i \in \{j+1, \ldots, k\}$ we have $s_i - i \geq 1$. Hence, any sequence $S' = (s_1, \ldots, s_j, t, s_{j+1}, \ldots, s_k)$ is a $(k+1)$-sequence in A'. □

Example 2. Let $A = \{1, 2, 3, 3, 3, 5, 5, 8\}$ and let $A' = A \cup \{t\}$. Consider $t > b^{max}$, e.g. $t = 6$. Then A' contains a maximal 7-sequence $S^{A'}_{max} = (3, 3, 3, 5, 5, 6, 8)$, while for $t \leq b^{max}$, e.g. $t = 5$ a 6-sequence $S^{A'}_{max} = (3, 3, 5, 5, 5, 8)$ is maximal.

Lemma 3. *Let A contain a maximal k-sequence S. If $A' = A \setminus \{t\}$, then A' contains a maximal $(k-1)$-sequence if and only if $t > e^{max}$.*

Proof. Let S be any k-sequence saturating in A. Then $e^{max} = 0$ and for any t, $e^{max} < t$. On the other hand, removal of any element t from S directly results in the $(k-1)$-sequence S' maximal in A'. Now, assume that S is any k-sequence non-saturating in A.

($\Rightarrow$) On the contrary assume that A' was obtained by removal of $t \leq e^{max}$ from $S = (s_1, \ldots, s_{i-1}, t, s_{i+1}, \ldots, s_k)$ and let $S' = S \setminus \{t\}$. If $t \notin S^A_{min}$, then $S^A_{min} \subseteq A'$, a contradiction. If $t \in S^A_{min}$, then there exists an element $t' \in A \setminus S^A_{min}$, $t' = e^{max}$ such that $S' = (s_1, \ldots, s_{i-1}, t', s_{i+1}, \ldots, s_k)$ is a k-sequence in A. A contradiction.

($\Leftarrow$) It is enough to see that all elements $t > e^{max}$ must belong to every maximal k-sequence S in A, or in other words no element from $A \setminus S$ could replace t without decreasing k. □

Example 3. Let $A = \{1, 2, 3, 3, 3, 5, 5, 8\}$ and let $A' = A \setminus \{t\}$. Recall $e^{max} = 3$ and consider $t > e^{max}$, e.g. $t = 5$. Then a 5-sequence $S^{A'}_{min} = (1, 2, 3, 5, 8)$ is maximal in A', while for $t \leq e^{max}$, e.g. $t = 2$ a 6-sequence still exists, e.g. $S^{A'}_{max} = (3, 3, 3, 5, 5, 8)$.

2.2 Potential Function

The *potential of a vertex v* is the length $p_G(v)$ of the maximal k-sequence in a multiset $A = \{d_G(u) \,|\, u \in N_G(v)\}$, while $P(G) = \max_{v \in V(G)} p(v)$ is called

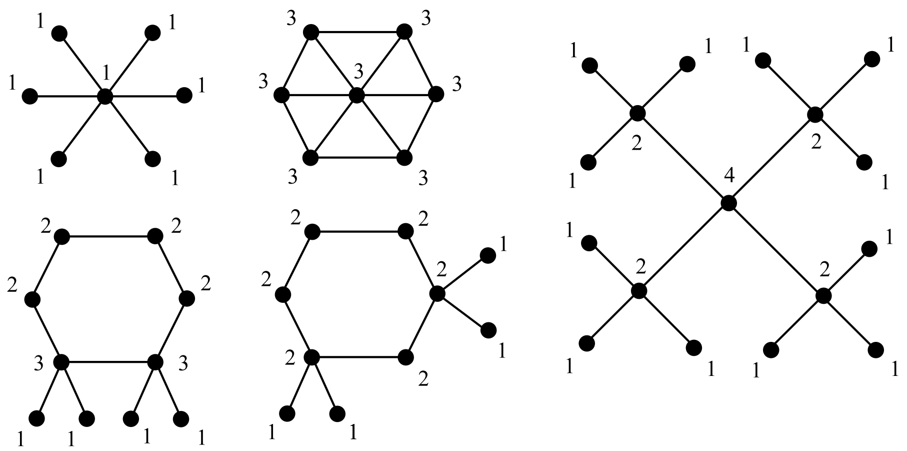


Fig. 1. Values of the potential function for various graphs

the *potential of a graph G* (see Fig. 1 for examples). From the definition we immediately have $p_G(v) \leq d_G(v)$ and any vertex v for which $d_G(v) = p_G(v)$ is called *saturated*. A vertex v is called *critical* if for every $u \in N_G(v)$ either

(a) u is saturated and $p_G(u) \leq d_G(v)$ or
(b) u is not saturated and $p_G(u) < d_G(v)$.

A critical vertex v may simultaneously have neighbors of both types (a) and (b).

In what follows we give a strengthening of lower bound (1) using potentials of vertices instead of their degrees. In order to prove the main result we need the following proposition.

Proposition 1. *Let $x \in V(G)$ be a critical vertex and let H be the subgraph of G induced by $V(G) \setminus \{x\}$. Then for every $u \in N_G(x)$ we have $p_H(u) \leq p_G(u) - 1$.*

Proof. Consider arbitrary vertex $u \in N_G(x)$ and let $e_G^{max}(u)$ denote e^{max} in a multiset $A = \{d_G(v) \,|\, v \in N_G(u)\}$. If u is saturated, then the assertion is obviously true. Assume u is not saturated. Since $e_G^{max}(u) \leq p_G(u)$, following the definition of a critical vertex we have $p_G(u) < d_G(x)$ and it follows that $e_G^{max}(u) < d_G(x)$. Hence, by Lemma 3, $p_H(u) \leq p_G(u) - 1$. □

To see that the potential of not saturated vertex u for which $p_G(u) = d_G(x)$ or $p_G(u) > d_G(x)$ does not have to decrease, consider $N_G(u)$ consisting of vertices having degrees $\{1, 2, 3, 3\}$ and remove vertex $x \in N_G(u)$ of degree 3 or x of degree 2, respectively. It is also worth pointing out that after deletion of vertex x, the potentials of some vertices $u \in N_G(x)$ may be even smaller, since they may be neighbors one of another, e.g. consider a graph $K_2 + 2K_1$, i.e. a join of K_2 with two disjoint copies of K_1, and delete vertex of degree 3. Note that for K_3-free graphs $p_H(u) = p_G(u) - 1$. Proposition 1 works fine for some natural choices of x, e.g. for x of locally largest degree, i.e. when for every $u \in N_G(x)$,

$d_G(u) \leq d_G(x)$. It holds also when x has globally largest degree $d_G(x) = \Delta(G)$. In the next section we discuss these and other choices in context of the selection rules of `GreedyMAX`-type algorithms.

3 Performance of `GreedyMAX`-type Algorithms

3.1 Family of `GreedyMAX`-type Algorithms

The `GreedyMAX`-type algorithm selects a critical vertex x, deletes it with all incident edges from the graph and iterates this process on the resulting graph until no edge remains. Repeating deletions naturally defines the sequence of vertices $(x_0, \ldots, x_{t-1})$ as well as the sequence of subgraphs $(G_0, \ldots, G_t)$, where $G_{i+1} = G_i[V(G_i) \setminus \{x_i\}]$ and t is the number of iterations. Let us consider the following template for `GreedyMAX`-type algorithms.

```
Algorithm GreedyMAX-type;
   input: $G$ - a simple graph;
  output: $I$ - a maximal independent set in $G$;
Begin
    $i \leftarrow 0$;  $G_i \leftarrow G$;  $I \leftarrow \emptyset$;
    While $E(G_i) \neq \emptyset$ do
        choose critical vertex $x_i$ according to the selection rule;
        $G_{i+1} \leftarrow G_i[V(G_i) \setminus \{x_i\}]$;
        $i \leftarrow i + 1$;
    $I \leftarrow V(G_i)$;
End.
```

In order to define a particular `GreedyMAX`-type algorithm one has to specify an appropriate *selection rule*. Following the definition of a critical vertex we give examples of criteria which can be used as a basis for the definition of selection rules:

(C1) choose x_i such that for every $u \in N_{G_i}(x_i)$, $d_{G_i}(u) \leq d_{G_i}(x_i)$,
(C2) select x_i for which $d_{G_i}(x_i) = \Delta(G_i)$,
(C3) pick not saturated x_i with $p_{G_i}(x_i) = P(G_i)$,
(C4) pick not saturated x_i with $p_{G_i}(x_i) = \mu(G_i)$, $\mu(G) = \min_{v \in V(G)} p_G(v)$.

Note that criteria (C1) - (C4) do not exclude each other and can be used simultaneously. For the correctness of the algorithm it is necessary to point out that in contrast to (C1) and (C2), that may be used as a standalone selection rules, G may not contain a vertex satisfying (C3), (C4). Though (C2) is a special case of (C1) it is given separately since the sole use of it defines the selection rule of the classical `GreedyMAX`.

It remains to argue that vertex x in (C1) - (C4) is critical. For (C1) recall $d_G(u) = p_G(u)$ for saturated u and $d_G(u) > p_G(u)$, when u is not saturated. Criticality in (C2) follows easily from (C1). Concerning (C3) it is enough to see that for every $u \in N_G(x)$, $p_G(u) \leq P(G) = p_G(x)$ and since x is not saturated,

$p_G(x) < d_G(x)$. For (C4) observe that since $p_G(x)$ is the smallest, for every $u \in N_G(x)$, $p_G(u) \geq p_G(x)$. Obviously $d_G(u) \geq p_G(u)$, thus $d_G(u) \geq p_G(x)$. If there existed even one $u' \in N_G(x)$ such that $d_G(u') > p_G(x)$, then since x is not saturated it still would have at least $k = d_G(x) - 1 \geq \mu(G)$ neighbors u having $d_G(u) = p_G(x)$. Hence, a sequence $(d_G(u_1), \ldots, d_G(u_k), d_G(u'))$, would be a $(\mu(G)+1)$-sequence in $N_G(x)$, a contradiction.

3.2 Analysis of `GreedyMAX`-type Algorithms

As the main result of this paper we prove the following theorem.

Theorem 1. *If I is an independent set generated by some* `GreedyMAX`*-type algorithm, then*

$$|I| \geq \sum_{v \in V(G)} \frac{1}{p_G(v)+1} \ . \tag{2}$$

Proof. Denote $CWP(G) = \sum_{v \in V(G)} 1/(p_G(v)+1)$. We are going to prove that for any $i \in \{0, \ldots, t-1\}$, $CWP(G_{i+1}) \geq CWP(G_i)$ and consequently $|I| = CWP(G_t) \geq CWP(G_0) = CWP(G)$. Recall that by Proposition 1, deletion of x_i decreases the potentials of vertices in $N_{G_i}(x_i)$. However, this also changes the degrees of all vertices from $N_{G_i}(x_i)$ and may in turn influence the potentials of some vertices in $U = V(G_i) \setminus N_{G_i}[x_i]$. Therefore, for the subgraph G_{i+1} we write

$$CWP(G_{i+1}) = \sum_{v \in U} \frac{1}{p_{G_{i+1}}(v)+1} + \sum_{u \in N_{G_i}(x_i)} \frac{1}{p_{G_{i+1}}(u)+1} \ ,$$

while for G_i we have

$$CWP(G_i) = \sum_{v \in U} \frac{1}{p_{G_i}(v)+1} + \sum_{u \in N_{G_i}(x_i)} \frac{1}{p_{G_i}(u)+1} + \frac{1}{p_{G_i}(x_i)+1} \ .$$

On the contrary assume that $CWP(G_i) > CWP(G_{i+1})$. Since for any $v \in U$ it holds $p_{G_i}(v) \geq p_{G_{i+1}}(v)$, we have

$$\sum_{u \in N_{G_i}(x_i)} \frac{1}{p_{G_i}(u)+1} + \frac{1}{p_{G_i}(x_i)+1} > \sum_{u \in N_{G_i}(x_i)} \frac{1}{p_{G_{i+1}}(u)+1} \ .$$

By Proposition 1 for any $u \in N_{G_i}(x_i)$, $p_{G_{i+1}}(u) \leq p_{G_i}(u) - 1$. Hence

$$\frac{1}{p_{G_i}(x_i)+1} > \sum_{u \in N_{G_i}(x_i)} \frac{1}{p_{G_i}(u)(p_{G_i}(u)+1)} \ . \tag{3}$$

Now, for $u \in N_{G_i}(x_i)$ consider values of $p_{G_i}(u)$ which minimize the right-hand side of (3). We argue that even for these values the inequality does not hold. Hence, in what follows, we assume that $p_{G_i}(u) = d_{G_i}(u)$ for all $u \in N_{G_i}(x_i)$, i.e. all neighbors of x_i are saturated.

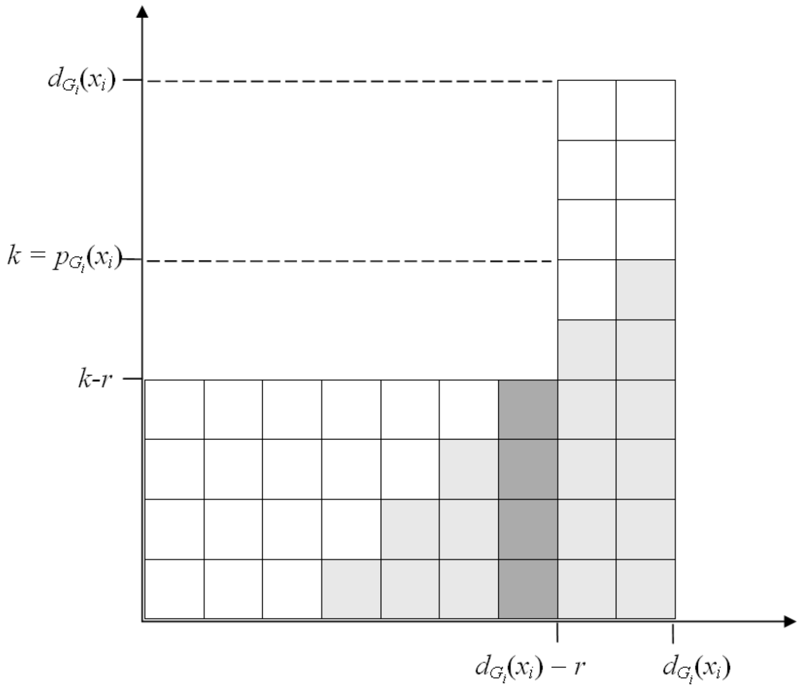


Fig. 2. A diagram of a two-level pattern for the sequence of degrees in $N_{G_i}(x_i)$

Case 1. Let vertex x_i be saturated, i.e. $p_{G_i}(x_i) = d_{G_i}(x_i)$. Since x_i is critical, $p_{G_i}(u) \leq d_{G_i}(x_i)$. Therefore

$$\begin{aligned}\frac{1}{p_{G_i}(x_i)+1} &> \sum_{u \in N_{G_i}(x_i)} \frac{1}{p_{G_i}(u)(p_{G_i}(u)+1)} \\ &\geq \sum_{u \in N_{G_i}(x_i)} \frac{1}{d_{G_i}(x_i)(d_{G_i}(x_i)+1)} = \frac{1}{d_{G_i}(x_i)+1} \ .\end{aligned}$$

Hence, $p_{G_i}(x_i) < d_{G_i}(x_i)$, a contradiction.

Case 2. Assume that x_i is not saturated, i.e. $p_{G_i}(x_i) < d_{G_i}(x_i)$. Let $k = p_{G_i}(x_i)$ and let $(u_1, \ldots, u_{d_{G_i}(x_i)})$ be the neighbors of vertex x_i ordered non-decreasingly with respect to their degrees. Observe that whenever for all neighbors $p_{G_i}(u_j) = k$, we have $1/(k+1) > d_{G_i}(x_i)/(k(k+1))$, a contradiction since $d_{G_i}(x_i)/k > 1$. Therefore, assume that x_i has r, $1 \leq r \leq k-1$, neighbors for which $p_{G_i}(u_j) > k$. Note that r has to be smaller than k and for the remaining $d_{G_i}(x_i) - r$ neighbors, $p_{G_i}(u_j) \leq k - r$ must hold (otherwise, since x_i is not saturated, $p_{G_i}(x_i)$ would be greater). Minimizing the right-hand side of (3) we take the largest possible values of neighbors' potentials, which results in the following two-level pattern: $p_{G_i}(u_j) = k - r, j \in \{1, \ldots, d_{G_i}(x_i) - r\}$ and $p_{G_i}(u_j) = d_{G_i}(x_i), j \in \{d_{G_i}(x_i) - r + 1, \ldots, d_{G_i}(x_i)\}$ (see Fig. 2 with gray bars

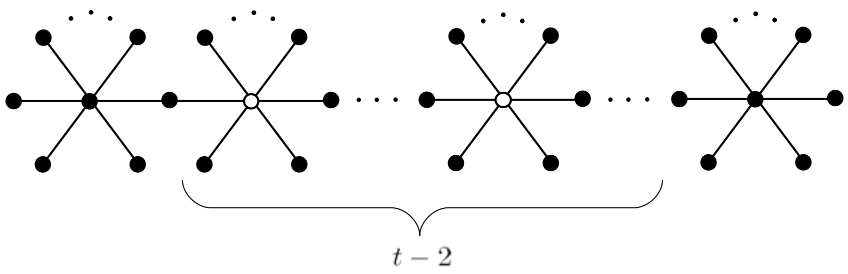


Fig. 3. $CWP(G) - CW(G)$ may be arbitrarily large

for elements of the k-sequence and dark gray for the blocking element). Let S_r stand for the right-hand side of (3) for $r \in \{1, \dots, k\}$. Consider $S_{r-1} - S_r$ for $r \in \{2, \dots, k-1\}$.

$$\begin{aligned} S_{r-1} - S_r &= \frac{d_{G_i}(x_i) - r + 1}{(k-r+1)(k-r+2)} + \frac{r-1}{d_{G_i}(x_i)(d_{G_i}(x_i)+1)} \\ &\quad - \frac{d_{G_i}(x_i) - r}{(k-r)(k-r+1)} - \frac{r}{d_{G_i}(x_i)(d_{G_i}(x_i)+1)} \\ &= \frac{k + r - 2d_{G_i}(x_i)}{(k-r)(k-r+1)(k-r+2)} - \frac{1}{d_{G_i}(x_i)(d_{G_i}(x_i)+1)} \ . \end{aligned}$$

Since $r \le k-1$ and $k < d_{G_i}(x_i)$, we have $k + r < 2d_{G_i}(x_i) - 1$. Hence, we have $S_{r-1} - S_r < 0$ and the considered minimum is attained for $r = 1$. However, it is not hard to see that also in this case (3) does not hold. □

Concerning the difference between $CW(G)$ and $CWP(G)$ we have the following

Theorem 2. *For any integer $\eta > 0$ there exists a connected graph G such that $CWP(G) - CW(G) > \eta$.*

Proof. Consider a graph G presented in Fig. 3 for which $t > 2$ is the number of vertices being neighbors of leaves, i.e. vertices of degree 1. Furthermore, let $d > 2$ be the degree of such vertices. Hence,

$$CW(G) = 2\left(\frac{d-1}{2} + \frac{1}{d+1}\right) + (t-2)\left(\frac{d-2}{2} + \frac{1}{3} + \frac{1}{d+1}\right) + \frac{1}{3} \ ,$$

while

$$CWP(G) = 2\left(\frac{d-1}{2} + \frac{1}{3}\right) + (t-2)\left(\frac{d-2}{2} + \frac{2}{3}\right) + \frac{1}{3} \ .$$

Therefore

$$CWP(G) - CW(G) = t\left(\frac{1}{3} - \frac{1}{d+1}\right)$$

may be arbitrarily large even for connected graphs. □

Let $\mathtt{A}(G)$ denote the cardinality of the independent set produced by the algorithm $\mathtt{A}$ for a graph G. A *performance ratio* $\rho_{\mathtt{A}}$ of $\mathtt{A}$ is defined as $\inf_G \mathtt{A}(G)/\alpha(G)$. Halldórsson and Radhakrishnan [11] provide a complete bipartite graph with removed perfect matching to argue that for any graph of maximum degree bounded by Δ we have $\rho_{\mathtt{GreedyMAX}} \leq 2/(\Delta+1)$, while the result of Griggs [10] implies $\rho_{\mathtt{GreedyMAX}} \geq 1/(\Delta+1)$. As a consequence of Theorem 1 we have the following, improved lower bound for the performance ratio of `GreedyMAX`-type algorithms for graphs having their maximum potential bounded by P.

Theorem 3. *If* $\mathtt{A}$ *is a* `GreedyMAX`*-type algorithm, then* $\rho_{\mathtt{A}} \geq 1/(P+1)$.

To see that for any $\eta > 0$ there exists a graph G such that $\Delta(G) - P(G) > \eta$, consider stars $K_{1,k}$ and wheels W_k. For stars $P(K_{1,k}) = 1$, $\Delta(K_{1,k}) = k$, while for wheels $P(W_k) = 3$ and $\Delta(W_k) = k$. In both cases we have a constant performance ratio, whereas the ratio depending on maximum degree gets worse when Δ grows. More complex examples are ct-graphs [2,20], where G is called a *ct-graph* if for each edge $uv \in E(G)$ the value of $|d_G(v) - d_G(u)|$ is constant.

References

1. Aardal, K., Verweij, B.: An optimization algorithm for maximum independent set with applications in map labeling. In: Nešetřil, J. (ed.) ESA 1999. LNCS, vol. 1643, pp. 426–437. Springer, Heidelberg (1999)
2. Acharya, B.D., Vartak, M.N.: On the construction of graphs with given constant valence-difference(s) on each of their lines. Wissenschaftliche Zeitschrifte TH Ilmenau 23(6), 33–60 (1977)
3. Alon, N., Spencer, J.H.: The probabilistic method. Wiley, New York (1992)
4. Boppana, R., Halldórsson, M.M.: Approximating maximum independent sets by excluding subgraphs. BIT 32, 180–196 (1992)
5. Caro, Y.: New results on the independence number. Technical Report, Tel-Aviv University (1979)
6. Chvátal, V., McDiarmid, C.: Small transversals in hypergraphs. Combinatorica 12, 19–26 (1992)
7. Erlebach, T., Jansen, K.: The maximum edge-disjoint paths problem in bidirected trees. SIAM J. Discrete Math. 14, 326–355 (2001)
8. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and Conquer: A simple $O(2^{0.288n})$ independent set algorithm. In: Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms, pp. 18–25 (2006)
9. Garey, M.R., Johnson, D.S.: Computers and Intractability. H. Freeman and Company, New York (1979)
10. Griggs, J.R.: Lower bounds on the independence number in terms of the degrees. J. Combin. Theory Ser. B 34, 22–39 (1983)
11. Halldórsson, M.M., Radhakrishnan, J.: Greed is good: Approximating independent sets in sparse and bounded-degree graphs. Algorithmica 18, 145–163 (1997)
12. Halldórsson, M.M.: Approximations of independent sets in graphs. In: Jansen, K., Rolim, J.D.P. (eds.) APPROX 1998. LNCS, vol. 1444, pp. 1–13. Springer, Heidelberg (1998)
13. Harant, J.: A lower bound on the independence number of a graph. Discrete Math. 188, 239–243 (1998)

14. Harant, J., Ryjaček, Z., Schiermeyer, I.: Forbidden subgraphs implying the MIN-algorithm gives a maximum independent set. Discrete Math. 256, 193–201 (2002)
15. Harant, J., Schiermeyer, I.: On the independence number of a graph in terms of order and size. Discrete Math. 232, 131–138 (2001)
16. Harant, J., Schiermeyer, I.: A lower bound on the independence number of a graph in terms of degrees. Discuss. Math. Graph Theory 26, 431–437 (2006)
17. Håstad, J.: Clique is hard to approximate within $n^{1-\varepsilon}$. Acta Math. 182(1), 105–142 (1999)
18. Hertz, A.: Polynomially solvable cases for the maximum stable set problem. Discrete Appl. Math. 60, 195–210 (1995)
19. Hochbaum, D.S.: Approximating covering and packing problems: set cover, vertex cover, independent set and related problems. In: Hochbaum, D.S. (ed.) Approximation Algorithms for NP-hard problems. PWS Publishing Company, Boston (1995)
20. Jendrol', S., Ryjaček, Z.: 3-polytopes of constant tolerance of edges. Commentationes Mathematicae Universitatis Carolinae 22(4), 843–850 (1981)
21. Malesińska, E.: Graph-theoretical models for frequency assignment problems. Ph.D. Thesis, Technische Universitat Berlin (1997)
22. Murphy, O.: Lower bounds on the stability number of a graphs computed in terms of degrees. Discrete Math. 90, 207–211 (1991)
23. Rautenbach, D.: On vertex orderings and the stability number in triangle-free graphs. Discrete Math. 231, 411–420 (2001)
24. Sakai, S., Togasaki, M., Yamazaki, K.: A note on greedy algorithms for the maximum weighted independent set problem. Discrete Applied Math. 126, 313–322 (2003)
25. Selkow, S.M.: A probabilistic lower bound on the independence number of graphs. Discrete Math. 132, 363–365 (1994)
26. Spieksma, F.: Approximating an interval scheduling problem. In: Jansen, K., Rolim, J.D.P. (eds.) APPROX 1998. LNCS, vol. 1444, pp. 169–180. Springer, Heidelberg (1998)
27. Turán, P.: On an extremal problem in graph theory. Mat. Fiz. Lapok 48, 436–452 (1941)
28. Wei, V.K.: A lower bound on the stability number of a simple graph. Technical Memorandum No. 81-11217-9, Bell Laboratories (1981)

Sequential Optimization of Matrix Chain Multiplication Relative to Different Cost Functions

Igor Chikalov, Shahid Hussain, and Mikhail Moshkov

Mathematical and Computer Sciences & Engineering Division
King Abdullah University of Science and Technology
Thuwal 23955-6900, Saudi Arabia
{igor.chikalov,shahid.hussain,mikhail.moshkov}@kaust.edu.sa

Abstract. In this paper, we present a methodology to optimize matrix chain multiplication sequentially relative to different cost functions such as total number of scalar multiplications, communication overhead in a multiprocessor environment, etc. For n matrices our optimization procedure requires $O(n^3)$ arithmetic operations per one cost function. This work is done in the framework of a dynamic programming extension that allows sequential optimization relative to different criteria.

Keywords. Dynamic programming, matrix chain multiplication, cost functions, sequential optimization.

1 Introduction

In *dynamic programming* as introduced by Bellman [1], we usually look for one optimal solution of a problem. In this paper, we consider an extension of dynamic programming which allows us to describe efficiently (in terms of *directed acyclic graph* with vertices corresponding to subproblems) the whole set of optimal solutions. We can continue the procedure of optimization on the resulting optimal set of solutions, but relative to another criterion etc. This approach is used in [2,3,4] for optimization of decision trees. In this paper, we illustrate this approach on matrix chain multiplication problem.

Matrix chain multiplication is a classic optimization problem in computer science. For a given sequence A_1, $A_2, \ldots, A_n$ of matrices, we need to find the product of these n matrices in the most efficient way. Often the most efficient way is the parenthesization that minimizes the total number of scalar multiplications on a single processor (sequential) machine. However, other optimization scenarios are also possible (some of these optimization criteria are discussed in this paper). Matrix chain multiplication problem can be found in any standard text on algorithms such as Cormen et al. [5] or Alsuwaiyel [6]. This classic problem was introduced by Godbole in [7]. Hu and Shing [8] present an $O(n \log n)$ solution of this problem in contrast to $O(n^3)$ solution to the problem posed by Godbole in [7].

We should find the product $A_1 \times A_2 \times \cdots \times A_n$ (it is well known that matrix multiplication is associative, i.e., $A \times (B \times C) = (A \times B) \times C$). The cost of multiplying a chain of n matrices depends on the order of multiplications. Each possible ordering of multiplication of n matrices corresponds to a different parenthesization (e.g., $A \times (B \times C)$ or $(A \times B) \times C$). It is well known that the total number of ways to fully parenthesize n matrices is equal to the nth Catalan number bounded below by $\Omega(4^n/n^{1.5})$. We will consider different cost functions (complexity measures) for parenthesizations. Our aim is to find a parenthesization with minimum cost. Furthermore, we present a generic method to apply different cost functions to optimize matrix chain multiplication one after another called sequential optimization.

This paper consists of seven sections including Introduction. In Sect. 2 we discuss in detail some cost functions for matrix chain multiplication. In Sect. 3 we provide a mechanism to construct a directed acyclic graph to carry out optimization. Sequential optimization relative to different cost functions is discussed in Sect. 4. We provide a detailed example in Sect. 5. In Sect. 6 we discuss computational complexity of optimization and we conclude the paper in Sect. 7.

2 Cost Functions

Let $A_1, A_2, \ldots, A_n$ be matrices with dimensions $m_0 \times m_1$, $m_1 \times m_2$, $\ldots$, $m_{n-1} \times m_n$, respectively. We consider the problem of matrix chain multiplication of these matrices and denote it as $S(1, n)$. Furthermore, we consider subproblems of the initial problem. For $1 \le i \le j \le n$, we denote by $S(i, j)$ the problem of multiplication of matrices $A_i \times A_{i+1} \times \cdots \times A_j$.

We describe inductively the set $P(i, j)$ of parenthesizations for $S(i, j)$. We have $P(i, i) = \{A_i\}$ for all $i = 1, \ldots, n$. For $i < j$ we define:

$$P(i,j) = \bigcup_{k=i}^{j-1} \{(p_1 \times p_2) : p_1 \in P(i,k), p_2 \in P(k+1,j)\}.$$

We realize a parenthesization using n processors $\pi_1, \pi_2, \ldots, \pi_n$. Initially, the processor π_i contains the matrix A_i, $i = 1, \ldots, n$. Suppose we compute $(p_1 \times p_2)$, where p_1 is computed by the processor π_{i_1} and p_2 by the processor π_{i_2}. At this point, there are two possibilities i.e., either the processor π_{i_1} sends the matrices corresponding to p_1 to the processor π_{i_2} or vice versa, where the receiving processor computes the product of matrices corresponding to p_1 and p_2.

Let we should compute $(p_1 \times p_2)$, the work with p_1 is finished by the processor π_{i_1} and the work with p_2 is finished by the processor π_{i_2}. After that we have two possibilities either π_{i_1} sends the matrix corresponding to p_1 to π_{i_2}, and π_{i_2} multiplies matrices corresponding to p_1 and p_2 or vice-versa.

We describe inductively the notion of cost function ψ which associates to each parenthesization p a nonnegative integer $\psi(p)$, which can be interpreted as the cost of the parenthesization. Let $F(x_1, x_2, y_1, y_2, y_3)$ be a function from ω^5 into ω, i.e., $F : \omega^5 \to \omega$, where $\omega = \{0, 1, 2, 3, \ldots\}$.

Now for $i = 1, \dots, n$ we have $\psi(A_i) = 0$. Let $1 \le i \le k < j \le n$, $p_1 \in P(i,k)$, $p_2 \in P(k+1,j)$ and $p = (p_1 \times p_2)$, then

$$\psi(p) = F(\psi(p_1), \psi(p_2), m_{i-1}, m_k, m_j).$$

Note that p_1 describes a matrix of dimension $m_{i-1} \times m_k$ and p_2 describes a matrix of dimension $m_k \times m_j$.

The function F (and corresponding cost function ψ) is called *monotonic* if for $x_1' \le x_1$ and $x_2' \le x_2$, the following inequality holds:

$$F(x_1', x_2', y_1, y_2, y_3) \le F(x_1, x_2, y_1, y_2, y_3).$$

The function F (and corresponding cost function ψ) is called *strictly monotonic* if it is monotonic and for $x_1' \le x_1$ and $x_2' \le x_2$ such that $x_1' < x_1$ or $x_2' < x_2$ we have

$$F(x_1', x_2', y_1, y_2, y_3) < F(x_1, x_2, y_1, y_2, y_3).$$

2.1 Some Cost Functions

Let us consider examples of functions F and corresponding cost functions ψ.

1. Functions F_1 and $\psi^{(1)}$.

$$\begin{aligned}\psi^{(1)}(p) &= F_1(\psi^{(1)}(p_1), \psi^{(1)}(p_2), m_{i-1}, m_k, m_j) \\ &= \psi^{(1)}(p_1) + \psi^{(1)}(p_2) + m_{i-1}m_k m_j.\end{aligned}$$

   Parenthesizations p_1 and p_2 represent matrices of dimensions $m_{i-1} \times m_k$ and $m_k \times m_j$. For multiplication of these matrices we need to make $m_{i-1}m_k m_j$ scalar multiplications. So $\psi^{(1)}(p)$ is the total number of scalar multiplications required to compute the product $A_i \times \cdots \times A_j$ according to parenthesization $p = (p_1 \times p_2)$. We should add that $\psi^{(1)}(p)$ can be considered as time complexity of computation of p (when we count only scalar multiplications) using one processor. It is clear that F_1 and $\psi^{(1)}$ are strictly monotonic.
2. Functions F_2 and $\psi^{(2)}$.

$$\begin{aligned}\psi^{(2)}(p) &= F_2(\psi^{(2)}(p_1), \psi^{(2)}(p_2), m_{i-1}, m_k, m_j) \\ &= \max\{\psi^{(2)}(p_1), \psi^{(2)}(p_2)\} + m_{i-1}m_k m_j.\end{aligned}$$

   This cost function describes time complexity of computation of p (when we count only scalar multiplications) using n processors. Here the functions F_2 and $\psi^{(2)}$ are monotonic functions.
3. Functions F_3 and $\psi^{(3)}$.

$$\begin{aligned}\psi^{(3)}(p) &= F_3(\psi^{(3)}(p_1), \psi^{(3)}(p_2), m_{i-1}, m_k, m_j) \\ &= \psi^{(3)}(p_1) + \psi^{(3)}(p_2) + \min\{m_{i-1}m_k, m_k m_j\}.\end{aligned}$$

This cost function describes the total cost of sending matrices between the processors when we compute p by n processors. We have the following situation: either processor π_{i_1} can send the $m_{i-1} \times m_k$ matrix to π_{i_2}, or π_{i_2} can send the $m_k \times m_j$ matrix to π_{i_1}. The number of elements in the first matrix is equal to $m_{i-1}m_k$ and the number of elements in the second matrix is equal to $m_k m_j$. To minimize the number of elements that should be sent we must choose minimum between $m_{i-1}m_k$ and $m_k m_j$. It is clear that F_3 and $\psi^{(3)}$ are strictly monotonic.

4. Functions F_4 and $\psi^{(4)}$.

$$\begin{aligned}\psi^{(4)}(p) &= F_4(\psi^{(4)}(p_1), \psi^{(4)}(p_2), m_{i-1}, m_k, m_j)\\ &= \max\{\psi^{(4)}(p_1), \psi^{(4)}(p_2)\} + \min\{m_{i-1}m_k, m_k m_j\}.\end{aligned}$$

This function describes the cost of sending matrices between processors in the worst-case where we use n processors. The functions F_4 and $\psi^{(4)}$ are monotonic functions.

3 Optimization

Now we describe a *directed acyclic graph* (DAG) G_0 which allows us to represent all parenthesizations $P(i,j)$ for each subproblem $S(i,j)$, $1 \le i \le j \le n$. The set of vertices of this graph coincides with the set

$$\{S(i,j) : 1 \le i \le j \le n\}.$$

If $i = j$ then $S(i,j)$ has no outgoing edges. For $i < j$, $S(i,j)$ has exactly $2(j-i)$ outgoing edges. For $k = i, \ldots, j-1$, exactly two edges start from $S(i,j)$ and finish in $S(i,k)$ and $S(k+1,j)$, respectively. These edges are labeled with the index k, we call these edges a *rigid pair* of edges with index k.

Let G be a subgraph of G_0 which is obtained from G_0 by removal of some rigid pairs of edges such that for each vertex $S(i,j)$ with $i < j$ at least one rigid pair outgoing from $S(i,j)$ remains intact.

Now for each vertex $S(i,j)$ we define, by induction, the set $P_G(i,j)$ of parenthesizations corresponding to $S(i,j)$ in G. For every i we have $P_G(i,i) = \{A_i\}$. For $i < j$, let $K(i,j)$ be the set of indexes of remaining rigid pairs outgoing from $S(i,j)$, then

$$P_G(i,j) = \bigcup_{k \in K(i,j)} \{(p_1 \times p_2) : p_1 \in P_G(i,k), p_2 \in P_G(k+1,j)\}.$$

Let ψ be a cost function, we consider the process of optimization of parenthesizations corresponding to vertices of G relative to ψ. We know that $\psi(A_j) = 0$ for $i = 1, \ldots, n$. For $1 \le i \le k < j \le n$, $p_1 \in P(i,k)$, $p_2 \in P(k+1,j)$ and $p = (p_1 \times p_2)$, we have

$$\psi(p) = F(\psi(p_1), \psi(p_2), m_{i-1}, m_k, m_j).$$

For each vertex $S(i,j)$ of the graph G we mark this vertex by the number $\psi_{i,j}$ which can be interpreted as minimum value of ψ on $P_G(i,j)$. For $i = 1,\ldots,n$, we have $\psi_{i,i} = 0$. If for $i < j$ and for each $k \in K(i,j)$ we know values $\psi_{i,k}$ and $\psi_{k+1,j}$, we can compute the value $\psi_{i,j}$:

$$\psi_{i,j} = \min_{k \in K(i,j)} F(\psi_{i,k}, \psi_{k+1,j}, m_{i-1}, m_k, m_j).$$

Let $\Psi(i,k,j)$ be defined as follows:

$$\Psi(i,k,j) = F(\psi_{i,k}, \psi_{k+1,j}, m_{i-1}, m_k, m_j),$$

now we should remove from G each rigid pair with index k such that

$$\Psi(i,k,j) > \psi_{i,j}.$$

We denote by G_ψ the resulting subgraph obtained from G. It is clear that for each vertex $S(i,j)$, $i < j$, at least one rigid pair outgoing from $S(i,j)$ was left intact.

The following theorem summarizes the procedure of optimization for monotonic cost function.

Theorem 1. *Let ψ be a monotonic cost function. Then for any i and j, $1 \le i \le j \le n$, $\psi_{i,j}$ is the minimum cost of a parenthesization from $P_G(i,j)$ and each parenthesization from $P_{G_\psi}(i,j)$ has the cost equal to $\psi_{i,j}$.*

Proof. The proof is by induction on $j - i$. If $i = j$ then $P_G(i,j) = \{A_i\}$, $P_{G_\psi}(i,j) = \{A_i\}$, $\psi(A_i) = 0$ and $\psi_{i,j} = 0$ by definition. So if $j - i = 0$ the considered statement holds. Let for some $t > 0$ and for each pair (i,j) such that $j - i \le t$ this statement hold.

Let us consider a pair (i,j) such that $j - i = t + 1$. Then

$$\psi_{i,j} = \min_{k \in K(i,j)} \Psi(i,k,j),$$

where $\Psi(i,k,j) = F(\psi_{i,k}, \psi_{k+1,j}, m_{i-1}, m_k, m_j)$. Since $i \le k < j$, $k - i \le t$ and $j - k - 1 \le t$. By induction hypothesis the considered statement holds for the pairs (i,k) and $(k+1,j)$ for each $k \in K(i,j)$. Furthermore, for $k \in K(i,j)$ let

$$P_G^k(i,j) = \{(p_1 \times p_2) : p_1 \in P_G(i,k), p_2 \in P_G(k+1,j)\}.$$

It is clear that

$$P_G(i,j) = \bigcup_{k \in K(i,j)} P_G^k(i,j). \tag{1}$$

Using the monotonicity of F we know that $\Psi(i,k,j)$ is the minimum cost of an element from $P_G^k(i,j)$. From this and (1), it follows that $\psi_{i,j}$ is the minimum cost of an element from $P_G(i,j)$. In G_ψ the only rigid pairs outgoing from $S(i,j)$ are those for which $\Psi(i,k,j) = \psi_{i,j}$ where k is the index of considered pair.

It is clear that the cost of each element from the set

$$P^k_{G_\psi}(i,j) = \{(p_1 \times p_2) : p_1 \in P_{G_\psi}(i,k), p_2 \in P_{G_\psi}(k+1,j)\}$$

is equal to

$$F(\psi_{i,k}, \psi_{k+1,j}, m_{i-1}, m_k, m_j) = \Psi(i,k,j)$$

since the cost of each element from $P_{G_\psi}(i,k)$ is equal to $\psi_{i,k}$ and the cost of each element from $P_{G_\psi}(k+1,j)$ is equal to $\psi_{k+1,j}$. Since

$$P_{G_\psi}(i,j) = \bigcup_{\substack{k \in K(i,j) \\ \Psi(i,k,j)=\psi_{i,j}}} P^k_{G_\psi}(i,j)$$

we have that each parenthesization from $P_{G_\psi}(i,j)$ has the cost equal to $\psi_{i,j}$. □

Furthermore, in case of a strictly monotonic cost function we have stronger result summarized in terms of following theorem.

Theorem 2. *Let ψ be a strictly monotonic cost function. Then for any i and j, $1 \le i \le j \le n$, $\psi_{i,j}$ is the minimum cost of a parenthesization from $P_G(i,j)$ and $P_{G_\psi}(i,j)$ coincides with the set of all elements $p \in P_G(i,j)$ for which $\psi(p) = \psi_{i,j}$.*

Proof. Since ψ is a strictly monotonic cost function then if follows from Theorem 1 that $\psi_{i,j}$ is the minimum cost of parenthesization in $P_G(i,j)$ and every parenthesization in $P_{G_\psi}(i,j)$ has the cost $\psi_{i,j}$. To prove this theorem it is enough to show that an arbitrary parenthesization p in $P_G(i,j)$ with optimal cost $\psi_{i,j}$ also belongs to $P_{G_\psi}(i,j)$.

We prove this statement by induction on $j-i$. If $j=i$ then $P_G(i,j) = \{A_i\}$ and $P_{G_\psi}(i,j) = \{A_i\}$. So for $j-i=0$ the considered statement holds. Let this statement hold for some $t > 0$ and for each pair (i,j) such that $j-i \le t$. Let us consider a pair (i,j) such that $j-i = t+1$.

Let $p \in P_G(i,j)$ and $\psi(p) = \psi_{i,j}$. Since $p \in P_G(i,j)$ then there is k, $i \le k < j$, such that $p = (p_1 \times p_2)$ where $p_1 \in P_G(i,k)$ and $p_2 \in P_G(k+1,j)$. Also, we know that $\psi(p) = F(\psi(p_1), \psi(p_2), m_{i-1}, m_k, m_j)$ and $\psi(p) = \psi_{i,j}$. Let us assume that $\psi(p_1) > \psi_{i,k}$ or $\psi(p_2) > \psi_{k+1,j}$. Since ψ and F are strictly monotonic, we have

$$\psi(p) > F(\psi_{i,k}, \psi_{k+1,j}, m_{i-1}, m_k, m_j) = \Psi(i,k,j) \ge \psi_{i,j},$$

however, this is impossible. So we have $\Psi(i,k,j) = \psi_{i,j}$, $\psi(p_1) = \psi_{i,k}$ and $\psi(p_2) = \psi_{k+1,j}$. Since $\Psi(i,k,j) = \psi_{i,j}$, we have that the graph G_ψ has a rigid pair with index k outgoing from the vertex $S(i,j)$.

From inductive hypothesis we have $p_1 \in P_{G_\psi}(i,k)$ and $p_2 \in P_{G_\psi}(k+1,j)$, therefore $p \in P_{G_\psi}(i,j)$. □

4 Sequential Optimization

Our optimization procedure (presented in previous section) is very generic in nature. We can apply several cost functions *sequentially* one after another to optimize matrix multiplication for several optimization criteria. In fact similar techniques have been used extensively for optimization of decision trees, see [2,3,4,9] for a wide range of complexity functions and their applications.

Initially we get a DAG $G = G_0$. This DAG G can be optimized according to any cost function ψ (monotonic or strictly monotonic). The generality of this technique allows us to use the resulting DAG G_ψ as input to further optimize for any other cost function say ψ' resulting in a DAG that is sequentially optimized according to two different cost functions ψ and ψ', i.e., $G_{\psi,\psi'}$. We can continue optimizing on the resulting graph for further cost functions.

It is important to note here that in the process of optimization we retain the vertices and only remove the edges. However, in this process there may be some vertices which become unreachable from the starting vertex (that represents the initial basic problem).

In the next section we discuss an example of matrix chain multiplication. We optimize it for several cost functions one after another, and do not consider vertices that are unreachable from the initial vertex $S(1, n)$.

5 An Example

We consider an example of matrix chain multiplication. Let A_1, A_2, A_3, A_4, and A_5 be five matrices with dimensions 2×2, 2×5, 5×5, 5×2, and 2×2, respectively, i.e., $m_0 = 2$, $m_1 = 2$, $m_2 = 5$, $m_3 = 5$, $m_4 = 2$, and $m_5 = 2$.

We use the cost function $\psi^{(1)}$ and obtain the following DAG (as shown in Fig. 1). This DAG is a subgraph of $G_{\psi^{(1)}}$ (note that $G = G_0$) as it does not

$$\begin{tikzcd}
 & & S(1,5) \arrow[dll, "1"'] \arrow[dl, "4"] \arrow[dr, "1"] \arrow[drr, "4"] & & \\
S(1,1) & S(1,4) \arrow[l, "1"'] \arrow[dr, "1"'] & & S(2,5) \arrow[r, "4"'] \arrow[dl, "4"] & S(5,5) \\
 & & S(2,4) \arrow[dl, "2"] \arrow[dr, "2"] & & \\
 & S(2,2) & & S(3,4) \arrow[dl, "3"] \arrow[dr, "3"] & \\
 & & S(3,3) & & S(4,4)
\end{tikzcd}$$

Fig. 1. Subgraph of $G_{\psi^{(1)}}$ after removing several rigid pairs from G with respect to the cost function $\psi^{(1)}$

include vertices $S(i,j)$ (representing subproblems) which are unreachable from the vertex $S(1,5)$ (representing the main problem).

The resulting directed acyclic graph $G_{\psi^{(1)}}$ describes the following two different parenthesizations, which are optimal from the point of view of $\psi^{(1)}$:

1. $(A_1 \times ((A_2 \times (A_3 \times A_4)) \times A_5))$,
2. $((A_1 \times (A_2 \times (A_3 \times A_4))) \times A_5)$.

We can apply another cost function say $\psi^{(3)}$ (which is also a strictly monotonic function) to our DAG $G_{\psi^{(1)}}$ and we obtain $G_{\psi^{(1)},\psi^{(3)}}$. Interestingly, we get exactly the same graph as there are no edges to remove from $G_{\psi^{(1)}}$.

We apply yet another cost function $\psi^{(4)}$ ($\psi^{(4)}$ is a monotonic function and not a strictly monotonic function). However, the example is highly symmetric and we again get the same DAG as depicted in Fig. 1.

6 Computational Complexity of Optimization

We assume that for each cost function ψ there exists a constant c_ψ such that to compute the value of ψ we need to make at most c_ψ arithmetic operations. The comparison operation is also considered as an arithmetic operation. In particular, we can choose $c_{\psi^{(1)}} = c_{\psi^{(2)}} = 4$ and $c_{\psi^{(3)}} = c_{\psi^{(4)}} = 5$.

Initially we get a DAG G_0. After optimization relative to a sequence of cost functions we get a DAG G, which is obtained from G_0 by removal of some rigid pairs. Let us now consider the process of optimization of G relative to a cost function ψ.

The number of subproblems $S(i,j)$ (vertices in G) is $O(n^2)$. It is enough to make $O(n)$ arithmetic operations to compute the value $\psi_{i,j}$ and to remove all rigid pairs for which

$$\Psi(i,k,j) > \psi_{i,j},$$

if the values $\psi_{i,k}$ and $\psi_{k+1,j}$ are already known for $k \in K(i,j)$. Therefore, the total number of arithmetic operations to obtain G_ψ from G is $O(n^3)$.

7 Conclusion

We have discussed in detail a procedure to sequentially optimize matrix chain multiplication relative to different cost functions. Furthermore, each step of optimization (relative to a cost function) can be carried out in polynomial-time with respect to the total number of matrices. We have also considered an example of matrix multiplication and successfully applied three different cost functions in succession.

References

1. Bellman, R.E.: Dynamic Programming. Princeton University Press, Princeton (1957)
2. Moshkov, M., Chikalov, I.: On algorithm for constructing of decision trees with minimal depth. Fundam. Inf. 41(3), 295–299 (2000)

3. Moshkov, M., Chikalov, I.: Consecutive optimization of decision trees concerning various complexity measures. Fundam. Inf. 61(2), 87–96 (2003)
4. Chikalov, I., Moshkov, M., Zelentsova, M.: On optimization of decision trees. In: Peters, J.F., Skowron, A. (eds.) Transactions on Rough Sets IV. LNCS, vol. 3700, pp. 18–36. Springer, Heidelberg (2005)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
6. Alsuwaiyel, M.H.: Algorithms: Design Techniques and Analysis. World Scientific, Singapore (1999)
7. Godbole, S.: On efficient computation of matrix chain products. IEEE Trans. Comput. 22(9), 864–866 (1973)
8. Hu, T.C., Shing, M.T.: Computation of matrix chain products. Part I. SIAM J. Comput. 11(2), 362–373 (1982)
9. Chikalov, I.V.: On algorithm for constructing of decision trees with minimal number of nodes. In: Ziarko, W.P., Yao, Y. (eds.) RSCTC 2000. LNCS (LNAI), vol. 2005, pp. 139–143. Springer, Heidelberg (2001)

One-Reversal Counter Machines and Multihead Automata: Revisited*

Ehsan Chiniforooshan[1], Mark Daley[1], Oscar H. Ibarra[2], Lila Kari[1], and Shinnosuke Seki[1]

[1] Department of Computer Science, University of Western Ontario, London Ontario N6A5B7, Canada
{ehsan,daley,lila,sseki}@csd.uwo.ca

[2] Department of Computer Science, University of California, Santa Barbara, CA 93106, USA
ibarra@cs.ucsb.edu

Abstract. Among the many models of language acceptors that have been studied in the literature are multihead finite automata (finite automata with multiple one-way input heads) and 1-reversal counter machines (finite automata with multiple counters, where each counter can only "reverse" once, i.e., once a counter decrements, it can no longer increment). The devices can be deterministic or nondeterministic and can be augmented with a pushdown stack. We investigate the relative computational power of these machines. Our results (where C_1 and C_2 are classes of machines) are of the following types:

1. Machines in C_1 and C_2 are incomparable.
2. Machines in C_1 are strictly weaker than machines in C_2.

In obtaining results of these types, we use counting and "cut-and-paste" arguments as well as an interesting technique that shows that if a language were accepted by a device in a given class, then all recursively enumerable languages would be decidable.

1 Introduction

A deterministic pushdown automaton (DPDA) is a deterministic finite automaton (DFA) augmented with a pushdown stack. The nondeterministic versions are called NPDA and NFA, respectively. It is well-known that a DPDA is weaker than an NPDA, and the latter machines accept exactly the context-free languages. A special case of a pushdown stack is one where the stack alphabet has only two symbols, one of which is used only to mark the bottom of the stack and cannot be modified. Such a stack is called a counter. Thus, we can think of a counter as a memory unit that holds a non-negative integer (initially zero), which can be incremented by 1, decremented by 1, left unchanged and tested for zero. A DFA (NFA) augmented with multiple (at least two) counters is equivalent to a Turing machine [12]. However, if we restrict the counters

* This research was supported by the National Science Foundation Grant CCF-0524136 of Oscar H. Ibarra, and by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant R2824A01, the Canada Research Chair Award in Biocomputing to Lila Kari.

to be *reversal-bounded*, the equivalence no longer holds. Here, reversal-bounded means that the number of alternations between non-decreasing mode and non-increasing mode each counter makes during the computation is at most r for some given positive integer r. An NFA augmented with such counters is called a *(reversal-bounded) counter machine* (NCM). The deterministic version is called a DCM. It was shown in [7] that an NCM can accept only a language whose Parikh map is semilinear. Closure and decidable properties for NCM and DCM were also investigated in [7]. For related results, see [2,3,4,5,8,10]. We note that a counter making r-reversals can be simulated by $\lceil \frac{r+1}{2} \rceil$ counters each of which makes 1 reversal. Hence, we may assume that the counters in a DCM and NCM are 1-reversal.

An NFA with multiple one-way input heads is called an NMHFA. The deterministic version is called DMHFA. These devices were first studied in [13], where it was reported that for any integer $k \geq 1$, a DMHFA (resp., NMHFA) with $k+1$ head is more powerful than one with only k heads. The proof in [13] was incomplete and was later completed in [15]. Stateless version of this hierarchy was later shown in [9].

A DPDA (resp. NPDA) can be generalized by augmenting it with multiple reversal-bounded counters – we will call it a DPCM (resp. NPCM), or by equipping it with multiple input heads – we will call it DMHPDA (resp. NMHPDA).

As for the classes of 1-reversal (pushdown) counter machines (DCM, NCM, DPCM, NPCM), our primary interest lies in the following questions:

1. whether a pushdown stack or non-determinism strictly strengthens 1-reversal counter machines, and
2. whether there is a trade-off, that is, whether an NCM can simulate a DPCM, and vice-versa.

We exhibit two languages which are in the symmetric difference of NCM and DPCM to answer the above questions negatively (Corollary 2).

It is of particular interest for applications to determine whether a given language is in DCM (or in DPCM) or not. This is because these deterministic classes possess desirable properties about decidability which are not shared by their non-deterministic counterparts such as the decidability of equivalence between a DCM and a DPCM (Corollary 5.4 in [7]) or between two DPDAs [14]. A typical tool to prove that a language is not in a given class is the pumping lemma, and actually pumping lemmas are available for DFA(= NFA), DPDA, and NPDA [6,16,17]. We propose another type of tool (Lemma 4), which reduces a language that is accepted by a DPCM with k-counters into its sub-language that is accepted by a DPCM with at most $k-1$ counters. Then, using the pumping lemma for DPDA [16], the reduction enables us to prove the existence of a language in NCM which cannot be accepted by any DPCM (Corollary 1).

In order to prove the incomparability between DPCM and NCM, we will also propose a language which is accepted by a DPCM but cannot be accepted by any NCM. A technique used toward this end deserves special mention: on the supposition that an NCM accepted this language, we could design an algorithm to decide the halting problem for Turing machines (Theorem 2).

2 Preliminaries

Let Σ be an alphabet and by Σ^* we denote the set of all words over Σ including the empty word ϵ. Let $\Sigma^+ = \Sigma^* \setminus \{\epsilon\}$. For a word $w \in \Sigma^*$, $|w|$ denotes its length and w^R denotes its reverse. By $|w|_a$, we denote the number of a letter $a \in \Sigma$ occurring in w. A subset L of Σ^* is called a *language*.

Let us recall the definition of reversal-bounded (pushdown) counter machines [7]. A *reversal-bounded counter machine* is a finite automaton augmented with reversal-bounded counters. We can further augment a reversal-bounded counter machine with a pushdown stack to obtain a *reversal-bounded pushdown counter machine*. Formally, a pushdown k-counter machine M is represented by a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where Q, Σ, Γ, F are the respective sets of states, input letters, stack symbols, and final states, q_0 is the initial state, and $Z_0 \in \Gamma$ is the particular stack symbol called the start symbol. We define the transition δ in a way that is different from but equivalent to the convention as a relation from $Q \times \Sigma \times \Gamma \times \{0,1\}^k$ into $Q \times \{\mathtt{S}, \mathtt{R}\} \times \Gamma^* \times \{-1, 0, +1\}^k$. S and R indicate the direction in which M moves its input head (S: stay, R: right). M is said to be *deterministic* if δ is a function. A configuration of M is given by a $(k+3)$-tuples $(q, w, x, c_1, \ldots, c_k)$ denoting the fact that M is in the state q, w is the "unexpended" input, the stack contains the word x, and $c_1, c_2, \ldots, c_k$ are the values contained in the k counters. Among configurations, we define a relation $\vdash_M$ as follows: $(q, aw, X\alpha, c_1, \ldots, c_k) \vdash_M (p, w', \beta\alpha, c_1 + e_1, \ldots, c_k + e_k)$ if $\delta(q, a, X, \lambda(c_1), \ldots, \lambda(c_k))$ contains $(p, d, \beta, e_1, \ldots, e_k)$, where $d \in \{\mathtt{S}, \mathtt{R}\}$, $e_1, \ldots, e_k \in \{-1, 0, +1\}$,

$$\lambda(c_i) = \begin{cases} 0 & \text{if } c_i = 0 \\ 1 & \text{otherwise,} \end{cases} \quad \text{and} \quad w' = \begin{cases} aw & \text{if } d = \mathtt{S} \\ w & \text{if } d = \mathtt{R}. \end{cases}$$

It is clear that the transition with the indicator S corresponds to the ϵ-transition in the conventional definition of pushdown counter machines, whereas that with R corresponds to the transition which consumes an input symbol. The reflexive and transitive closure of $\vdash_M$ is written as $\vdash_M^*$. The subscript is dropped from $\vdash_M$ and $\vdash_M^*$ whenever the particular M is understood. A word $w \in \Sigma^*$ is accepted by M if $(q_0, w, Z_0, 0, \ldots, 0) \vdash^* (q_f, \epsilon, \alpha, c_1, \ldots, c_k)$ for some $q_f \in F$. The set of all words accepted by M is denoted by $L(M)$. By ignoring the pushdown stack through the description so far, we can obtain the analogous definition and notions for counter machines (no pushdown stack).

A counter machine is said to be *reversal-bounded* if it has the property that for each of its counter the number of alternations between non-decreasing mode and non-increasing mode and vice-versa is bounded by a given constant in any computation[1]. For the reason mentioned in Introduction, we assume this

[1] The reversal-bounded property can be defined by taking into account only the accepting computation, but these two definitions are equivalent. A machine can remember, in its states, how many times each of its counters has made the reversal, and can abort a computation immediately after one of its counters makes the reversal more than the given constant times.

constant to be 1 in this paper. A finite state machine with a 1-reversal stack is said to be *1-turn*.

For an integer $k \geq 0$, let NCM(k) be the set of counter machines with k 1-reversal counters, and then let NCM $= \bigcup_{k=0}$ NCM(k). For notational conveniences, we also use NCM(k) to denote a machine in the class as well as the class of languages accepted by a machine in the class. Their deterministic subclasses are denoted by DCM(k) and DCM, respectively. Let us denote the class of pushdown machines with k 1-reversal counters (pushdown k-counter machines) by NPCM(k), and NPCM $= \bigcup_{k=0}$ NPCM(k). DPCM(k) and DPCM are their deterministic subclasses. Note that NCM(0) = DCM(0) is the class of regular languages, while NPCM(0) and DPCM(0) correspond to the classes of context-free languages and deterministic context-free languages, respectively.

3 NCM and DPCM Are Incomparable

We begin our investigation with proving that NCM and DPCM are incomparable with respect to the power to accept languages. To achieve this goal, we study the problem of how to determine whether a given language can be accepted by a machine in DCM(k), NCM(k), DPCM(k), or NPCM(k) for some $k \geq 0$ or not.

For specific values of k, pumping lemmata are available. Pumping lemmata for DCM(0) (the class of regular languages) as well as for NPCM(0) (that of context-free languages) are well-known the most [6,17]. We need the following one by Yu [16] for DPCM(0) (that of deterministic context-free languages).

Lemma 1 ([16]). *For $L \in$ DPCM(0), there exists a constant C for L such that for any pair of words $w, w' \in L$ if $w = xy$ and $w' = xz$ with $|x| > C$, and y and z begin with the same letter, then one of the following statements is true:*

- *there is a factorization $x = x_1x_2x_3x_4x_5$ with $x_2x_4 \neq \epsilon$ and $|x_2x_3x_4| \leq C$ such that for all $i \geq 0$, $x_1x_2^ix_3x_4^ix_5\{y, z\} \subseteq L$,*
- *there are factorizations $x = x_1x_2x_3$, $y = y_1y_2y_3$, and $z = z_1z_2z_3$, with $x_2 \neq \epsilon$ and $|x_2x_3| \leq C$ such that for all $i \geq 0$, $x_1x_2^ix_3\{y_1y_2^iy_3, z_1z_2^iz_3\} \subseteq L$.*

We use the above lemma along with a method which reduces a language in DPCM(k) into a language in DPCM(0) for an arbitrary k in order to prove that some language is NOT in DPCM.

Let M be a DPCM(k) for some $k \geq 0$. For an integer $1 \leq i \leq k$, we say that a word is *i-decreasing* if while M reading the word, the i-th counter is decreased.

Lemma 2. *Let $M \in$ DPCM(k). If there exists an integer $1 \leq i \leq k$ such that no word in $L(M)$ is i-decreasing, then $L(M) \in$ DPCM($k-1$).*

Proof. The basic idea is the following. It follows from the assumption that while processing an input, if M encounters the decrement of the i-th counter, the input should be rejected immediately. Also note that the transition function of M does not check the actual value of the counter, but it checks only whether the value is zero or non-zero. Thus, we can encode this zero-nonzero information about the i-th counter rather into its state, set its initial value to 0, and change it to 1 when M increments the i-th counter for the first time. □

Lemma 3. *Let $M \in \mathrm{DPCM}(k) \setminus \mathrm{DPCM}(k-1)$, and w be an i-decreasing word for some $1 \leq i \leq k$. Then $L^{(w)} := L(M) \cap w\Sigma^* \in \mathrm{DPCM}(k-1)$.*

Proof. First of all, Lemma 2 guarantees the existence of an i-decreasing word w in $L(M)$ for any $1 \leq i \leq k$. Let us build up a DPCM$(k-1)$ M' for $L^{(w)}$. Once activated, this machine firstly checks whether a given input begins with w and if not M' rejects the input immediately. During this check, M' simulates the computation of M on w except for the i-th counter. About the i-th counter, M remembers, using its state, the value of this counter which is obtained after M (deterministically) processes w. Since the i-th counter has been reversed already when M completes the processing of w, M' does not require infinite number of states in order to simulate the counter for the rest of its computation. □

By applying Lemmas 2 and 3 alternately as many times as necessary, we can find a subset of a given language in DPCM(k) which is accepted by a DPCM(0).

Lemma 4. *If $L \in \mathrm{DPCM}$, then there exists $w \in \Sigma^*$ such that $L^{(w)}$ is a nonempty* DPCM(0).

It is noteworthy that our argument so far does not rely on whether the pushdown stack is available for computation or not. Thus, analogous results of Lemmas 2, 3, and 4 hold for DCM.

Using Lemma 4, now we can prove the existence of a language which is in NCM, but cannot be accepted by any DPCM. As an example of such languages, we propose the *balanced language* L_b defined as follows:

$$L_b = \{a^{i_1}\#a^{i_2}\#\cdots\#a^{i_n} \mid n \geq 2, i_1, \ldots, i_n \geq 0 \\ \text{such that } i_1 + i_2 + \cdots + i_k = i_{k+1} + \cdots + i_n \text{ for some } 1 \leq k < n\}.$$

Informally speaking, an element of this language has the symbol # located at its fulcrum to the left and to the right of which there are the same number of a's. It is clear that L_b can be accepted by an NCM(1).

In order to verify that $L_b \notin \mathrm{DPCM}$, it suffices to prove that for any word $w \in \{a, \#\}^*$, $L_b^{(w)}$ cannot be accepted by any DPCM(0) due to Lemma 4.

Lemma 5. *For any $w \in \{a, \#\}^*$, $L_b^{(w)} = L_b \cap w\{a, \#\}^*$ is not a* DPCM(0).

Proof. Suppose $L_b^{(w)}$ were DPCM(0), then so is $L := L_b^{(w)} \cap w\Sigma^*\#\Sigma^*\#\Sigma^*$. Let $n = |w|_a$. Then $wa^C\#a^{C+n}\#, wa^C\#a^{C+n}\#a^{2(C+n)} \in L$, where C is the pumping constant given in Lemma 1. With $x = wa^C\#a^{C+n}$, $y = \#$, and $z = \#a^{2(C+n)}$, these words are written as xy and xz, which satisfy the requirements on x, y, z in Lemma 1. So one of the factorizations must hold. The pumped parts should not contain any # because any word of L contains exactly $|w|_\# + 2$ #'s. Let us consider the first factorization $x = x_1x_2x_3x_4x_5$. Since $x_2x_4 = a^m$ for some $1 \leq m \leq C$, omitting this pumpable part from $wa^C\#a^{C+n}\#a^{2(C+n)}$ would result in a word $wa^{C-i}\#a^{C+n-(m-i)}\#a^{2(C+n)}$ for some $i \geq 0$. This word is unbalanced so that other factorization $x = x_1x_2x_3$ and $y = y_1y_2y_3$ must hold.

Since $y = \#$, the pumped part y_2 is empty. If x_2 is in wa^C, then removing x_2 results in an unbalanced word. If x_2 is in a^{C+n}, then pumping x_2 more than once makes the resulting word unbalanced. □

Corollary 1. *L_b is not in* DPCM.

Proof. Suppose that $L_b \in$ DPCM. Lemma 4 implies that there exists $w \in \Sigma^*$ such that $L_b^{(w)} := L_b \cap w\Sigma^* \in$ DPCM(0), but this contradicts Lemma 5. □

Having seen that $L_b \in$ NCM \ DPCM, now we show that there are languages in DPCM that cannot be accepted by any NCM. This result shall lead us to the incomparability of NCM and DPCM. It would also follow that the machines in DCM are strictly less powerful than those in DPCM.

We give two proofs below. The first uses the following result in [1], whose proof is quite complicated (long and tedious).

Theorem 1 ([1]). *If M is an* NCM*, then we can construct another* NCM *M' such that $L(M') = L(M)$ and M' operates in linear time (i.e., every string of length n in L(M') has an accepting computation for which M' runs in linear time).*

Theorem 2. *There is a language accepted by a 1-turn* DPDA *that cannot be accepted by any* NCM.

Proof. Consider the language $L_{\text{pal}} = \{x\#x^R \mid x \in \{0,1\}^+\}$ (recall that R denotes reverse). It is obvious that L_{pal} can be accepted by a deterministic PDA (DPDA) and hence $L_{\text{pal}} \in$ DPCM(0). Suppose L_{pal} can be accepted by an NCM M with k counters. We may assume, by Theorem 1, that M operates in linear time. Consider an input $u\#u^R$, where $|u| = n$. Clearly, the number of possible configurations when the input head of M reaches $\#$ is $O(n^k)$. Now consider another input $v\#v^R$, where $|v| = n$ and $v \neq u$. It follows that since there are 2^n binary strings of length n, $u\#v^R$ will also be accepted by M for n large enough. This is a contradiction. □

We will give another proof of Theorem 2 since it employs an interesting technique, which constructs a contradictory algorithm to decide an undecidable recursively enumerable language on the assumption that a specific language can be accepted by an NCM, implying then that the language cannot be accepted by any NCM. Let us describe the details below.

1. Let $L \subseteq \{a\}^*$ be a unary recursively enumerable language that is not decidable (such L exists) and M be a Turing machine (TM) accepting L.
2. Let Q and Σ be the state set and worktape alphabet of M and let $q_0 \in Q$ be the initial state of M. Let $\Sigma' = Q \cup \Sigma \cup \{\#\}$. Note that a is in Σ. The halting computation of M on input a^d can be represented by the string $ID_1\#ID_3 \cdots \#ID_{2k-1}\#\#ID_{2k}^R \cdots \#ID_4^R\#ID_2^R$ for some $k \geq 2$ (without loss of generality, we can assume that the length of a computation is even), where $ID_1 = q_0a^d$ and ID_{2k} are the initial and halting configurations of M, and $(ID_1, ID_2, \cdots, ID_{2k})$ is a sequence of configurations of M on input a^d, i.e., configuration ID_{i+1} is a valid successor of ID_i.

Now consider the languages

$$\begin{aligned}L_1 = \{&ID_1\# \cdots \# ID_{2k-1}\#\# ID_{2k}^R \cdots \# ID_2^R \mid ID_{2k} \text{ is a halting configuration,}\\ &k \geq 2,\ ID_1 = q_0a^p (p \geq 1), \text{and } ID_{i+1} \text{ is a valid successor of } ID_i \text{ for odd } i\},\\ L_2 = \{&ID_1\# \cdots \# ID_{2k-1}\#\# ID_{2k}^R \cdots \# ID_2^R \mid ID_{2k} \text{ is a halting configuration,}\\ &k \geq 2,\ ID_1 = q_0a^p (p \geq 1), \text{and } ID_{i+1} \text{ is a valid successor of } ID_i \text{ for even } i\}.\end{aligned}$$

Clearly, L_1 and L_2 can be accepted by 1-turn DPDAs.

Theorem 3. *L_1 or L_2 cannot be accepted by any* NCM*.*

Proof. Suppose L_1 and L_2 are accepted by NCMs A_1 and A_2 with n_1 and n_2 1-reversal counters, respectively. We construct from A_1 and A_2 an NCM A accepting the language $L_1 \cap L_2 = \{ID_1\# \cdots \# ID_{2k-1}\#\# ID_{2k}^R \cdots \# ID_2^R \mid k \geq 2, ID_1 = q_0a^p, p \geq 1,\ ID_{2k}$ is a halting configuration, and ID_{i+1} is a valid successor of ID_i for $i \geq 1\}$. A simulates A_1 and A_2 in parallel (hence, A will have $n = n_1 + n_2$ 1-reversal counters).

Finally, we show (using NCM A) that there exists an algorithm to decide L, which would be a contradiction. The algorithm works as follows:

1. On an input a^d, construct a finite automaton B accepting $q_0a^d\#\Sigma'^*$.
2. From the finite automaton B and the NCM A (accepting $L_1 \cap L_2$), construct an NCM A' which accepts $L(A) \cap L(B)$.
3. Test if the language accepted by A' is empty. This is possible since the emptiness problem for NCMs (or even for NPCMs) is decidable [7].

Note that $a^d \notin L$ if and only if the language accepted by A' is empty. □

From Corollary 1 and Theorem 2, we have:

Corollary 2. *1.* NCM *are strictly more powerful than* DCM*.*
2. NCM *and* DPCM *are incomparable.*
3. DPCM *are strictly more powerful than* DCM*.*

Remark 1. Note that the fact that NCM is closed under intersection and has decidable emptiness problem while this problem is undecidable for DPCM does not necessary imply that these two classes are incomparable. This kind of argument would lead to a fallacy. For example, the class DFA of finite automaton languages is closed under intersection and has decidable emptiness problem. Now consider the class Null-TM of languages defined by TMs whose inputs can only be the null string ϵ. Then the emptiness problem for Null-TM is undecidable (because the halting problem for TMs on an initially blank tape is undecidable). However, Null-TM is contained in DFA (since Null-TM consists only of the two languages $\emptyset$ and $\{\epsilon\}$), hence, not incomparable.

4 Counter Machines and Multihead Automata

A (non-deterministic) multihead finite automaton (NMHFA) M with k heads (written as NMHFA(k)) is a machine with k one-way input heads operating on an input string with a right end marker \$. At the start of the computation, the heads are on the leftmost symbol of the input string and M is in its initial state. A move of the machine is a transition of the form $\delta(q, a_1, \ldots, a_k) = \{(p, d_1, \ldots, d_k)\}$, where q is the current state, $a_1, \ldots, a_k$ are the symbols scanned by heads, p is the next state, and $d_1, \ldots, d_k$ are the movements of the heads, which can be R or S (one position to the right, or do not move). Note that the heads are *non-sensing* (i.e., the heads cannot sense the presence of the other heads on the same position). An input is accepted if M eventually enters an accepting state and all heads are on the right end marker. The machine can be augmented with a pushdown stack (NMHPDA) in the obvious way. It can be deterministic (DMHFA, DMHPDA).

The main aim of this section is to investigate the comparability and incomparability among the classes of finite state machines with those of finite state machines with multiple heads. First, we prove that a DCM can be simulated by a DMHFA. In order to prove this, we propose some properties which can be assumed for DCMs without loss of generality. The first property says that given a DCM, we can construct a DCM with the same number of counters which reads any input till its end.

Lemma 6. *Any* DCM M *can be converted to a* DCM M' *such that* $L(M') = L(M)$ *and for every input,* M' *does not go into an infinite loop on a symbol on the input tape.*

Proof. Let M have s states. M' is constructed as follows:

1. M' simulates M.
2. If M has made more than s moves while remaining on the symbol without at least one of the following happening:
   (a) a positive counter decrementing,
   (b) a zero counter becoming positive (note that when a positive counter becomes zero from being positive, it can no longer become positive again),

then M is looping on the symbol. M' then has two cases to handle:

Case 1: During the looping, M does not enter any accepting state. In this case, M' enters a special (new) reject state r and scans the remaining input in state r until the head falls off the input.

Case 2: During the looping, M enters an accepting state. In this case, M' enters a special (new) accepting state f (thus accepting the input if there is no more symbol to the right of the head). Then M', in state f, on any input enters a special rejecting state r and scans the remaining input in state r until the head falls off the input.

The next lemma provides us with one desirable property that DCMs satisfying Lemma 6 have.

Lemma 7. *Any* DCM M *can be converted into a* DCM M' *such that* $L(M') = L(M)$ *and there exist constants* c_1, c_2*, and for any input* w*, the values of counters are at most* $c_1|w| + c_2$ *during the computation by* M' *on* w.

Proof. Let M be a DCM(k) with s states for some $k \geq 1$ (since we concern the value of counters, we ignore the case $k = 0$). Let M' be the DCM(k) converted from M according to Lemma 6. Let $c = s \times 2^k$. Since $c \geq s$, according to the design in Lemma 6, if M' makes more than c moves while its head remaining on the same symbol, then either the event (a) or event (b) must occur.

We claim that during the computation by M' on an input of length n, the value of each counter can be at most $(c+2)^{k-1}(c+1)(n+k)$, i.e., $O(n)$, by induction on the number of counters. Note that when two counters contain the respective values n_1 and n_2, M' can increment the first counter further by cn_2 while decreasing the second one up to 0. In order to make the value of a counter as large as possible, we employ the following strategies:

1. at each transition, M' will increase the values of all counters which are in non-decreasing mode;
2. M' never decrease two counters at the same transition.

For the case $k = 1$, more than c transitions should not occur uninterruptedly without moving its head. According to the first strategy, M should increment the counter from the very beginning transition, and once being decremented, M cannot increment the counter any more. Thus the value of this counter can be at most $(c+1)n + c \leq (c+1)(n+1)$.

Now suppose that the claim holds for some $k-1$ and consider the case when M' has k counters. Let us assume that M decrements $(k-1)$-th counter for the first time while M' moving its head from the $(m-1)$-th input symbol to the m-th one. At the point, the value of $(k-1)$-th or k-th counter can reach at most $(c+2)^{k-2}(c+1)(m+k-1)$ according to the induction hypothesis, even if M' makes the other counters to be 0. While decrementing the $(k-1)$-th counter up to 0, the k-th counter can increase at most by $(c+1)\cdot(c+2)^{k-2}(c+1)(m+k-1)+c$. By expending the rest of the input ($n-m$ symbols), we can increase at most $(c+1)(n-m)+c$. Thus, the k-th counter can become largest when $m = n$ and the value is $(c+2)^{k-2}(c+1)(n+k-1)+(c+1)\cdot(c+2)^{k-2}(c+1)(n+k-1)+c$, which is at most $(c+2)^{k-1}(c+1)(n+k)$. Thus, the induction step is verified and the claim holds. By letting $c_1 = (c+2)^{k-1}(c+1)$ and $c_2 = k(c+2)^{k-1}(c+1)$, this lemma holds. □

Proposition 1. *For any* $k \geq 0$, DCM(k) $\subseteq$ DMHFA($2k+1$).

Proof. Let $M \in$ DCM(k). Each 1-reversal counter C of M can be simulated by two input heads H_1 and H_2, which are initially positioned on the leftmost symbol of the input. When C increments by 1, H_1 is moved one position to the right. When C reverses, both heads are moved simultaneously to the right until H_1 reaches the end marker before continuing with the simulation of C. After that, decrementing C would correspond to moving H_2 one position to the right.

When H_2 reaches the end marker, this indicates that C has the value zero. This simulation works if the counter values are bounded by n, where n is the length of the input. If the counter values are bounded by $c_1 n + c_2$ for some constant c_1, c_2 (i.e., the counter values are linearly bounded), then H_1 (resp. H_2) operates modulo c_1, i.e., H_1 (resp. H_2) is moved one position to the right for every c_1 increments (resp. decrements) of C. The existence of such c_1, c_2 is guaranteed by Lemma 7. □

Next, we prove the incomparability between NPCM and NMHFA. Following the notation in the proof of Theorem 2, let us firstly consider the language:

$$L = \{ID_1 \# ID_2 \# ID_3 \# ID_4 \# \cdots \# ID_k \mid ID_1 = q_0 a^p \text{ for some } p \geq 1 \\ \text{and for } i \geq 1,\ ID_{i+1} \text{ is a valid successor of } ID_i\}.$$

It is clear that $L \in$ DMHFA(2). In contrast, the technique used to prove Theorem 2 is also applicable to show that $L \notin$ NPCM.

Proposition 2. *There is a language accepted by a* DMHFA(2) *that cannot be accepted by any* NPCM*.*

Due to Proposition 2, now it suffices to propose a language in NPCM\NMHFA to show the incomparability between NPCM and NMHFA. The marked palindromic language $L_{\text{pal}} = \{x \# x^R \mid x \in \{0,1\}^+\}$ serves this purpose, which is actually in DPDA. The following results may have already been known, but we have not been able to find an appropriate reference. We give a proof below for completeness. The proof uses a Kolmogorov-complexity-based idea in [11]. The Kolmogorov complexity of a word $w \in \Sigma^*$, denoted by $K(w)$, is defined to be the length of the shortest program that outputs only w. Let $K(w|y)$ be the conditional Kolmogorov complexity of w with respect to a given extra information y. A word w is said to be *random* if $K(w) \geq |w|$. It is well-known that there exist random strings. We state one more well-known fact without proof.

Fact 1. *If string uvw is random, then $K(v|uw) \gg |v| - O(\log|uvw|)$.*

Proposition 3. $L_{\text{pal}} \notin$ DMHFA(2)*.*

Proof. Suppose that there were a DMHFA(2) M such that $L(M) = L_{\text{pal}}$. Let h_r, h_l be the rightmost and leftmost heads of M, respectively.

Let us consider a random word $w = w_1 w_2$ satisfying $|w_1| = |w_2| \gg \log|w| + |M|$, where $|M|$ denotes the (program) size of M. Note that $w_1 \neq w_2$ because of the randomness of w. Then we put the following input into M:

$$I_1 = *w_1 * w_2 * \# * w_2^R * w_1^R * .$$

For this input, we say that w_i $(i = 1, 2)$ is *checked* if there is a time t when h_r is on w_i^R while h_l is on w_i.

We claim that both w_1 and w_2 have to be checked. Indeed, suppose that w_1 were not checked. Consider the computation of M on I_1. Let $ID_{\text{in}}(h_r)$

($ID_{\text{out}}(h_r)$) be the configuration of M when h_r first reaches the first (resp. second) * sign in $*w_1^R*$ of I_1. The analogous notation is defined for h_l. Note that these IDs can be described of length $O(\log|w|)$. Given these IDs and w_2, we can reconstruct w_1 by a short program as follows: For a word x with $|x| = |w_1|$, let $P(x) = *0^{|w_1|} * w_2 * \# * w_2^R * x *$. We simulate M on this program $P(x)$ starting with $ID_{\text{in}}(h_r)$ and $ID_{\text{in}}(h_l)$. If $ID_{\text{in}}(h_r) \vdash^*_{P(x)} ID_{\text{out}}(h_r)$ and $ID_{\text{in}}(h_l) \vdash^*_{P(x)} ID_{\text{out}}(h_l)$ hold, then M accepts $I_1(x) = *w_1 * w_2 * \# * w_2^R * x *$. This can happen if and only if $x = w_1^R$ (otherwise, an input not in L_{pal} would be accepted by M). Therefore, $K(w_1|w_2) = O(\log|w|)$, but this contradicts Fact 1.

Now, the claim has been verified so that both w_1 and w_2 have to be checked, but clearly it cannot be done by one-way multihead FA (if w_2 is checked, then h_l is on w_2 so that it cannot return back to w_1 in order to check w_1.) □

We can easily modify the proof of Proposition 3 to prove that L_{pal} cannot be accepted by any DMHFA not depending on how many heads are available. It suffices to split the random string w into k substrings of the same length as $w = w_1 w_2 \cdots w_k$ in order to prove that any DMHFA(k) cannot accept this language. With this modification, we can verify the next proposition.

Proposition 4. $L_{\text{pal}} \notin$ DMHFA.

We further strengthen the above result by showing that even non-determinism does not help for multihead finite automata to accept L_{pal}.

Lemma 8. *Let Σ be a non-unary alphabet, and @ be a symbol that is not in Σ. For a language $L \in$ NMHFA(k), there exists a language $L' \subseteq (\Sigma \cup \{@\})^*$ in* DMHFA($k+1$) *with a property that $w \in L$ if and only if $w@\Sigma^* \cap L' \neq \emptyset$.*

Proof. The transition function of an NMHFA(k) maps a tuple $(q, a_1, a_2, \ldots, a_k)$ into a set $\{(p_1, d_{11}, d_{12}, \ldots, d_{1k}), \ldots (p_m, d_{m1}, d_{m2}, \ldots, d_{mk})\}$, where $m \geq 0$ is an integer, p_is are future states, and d_{ij}s are either S or R. Hence, if an NMHFA(k) is in configuration $(q, a_1, a_2, \ldots, a_k)$, it will have m transition choices.

If L is accepted by an NMHFA(k) M and w is a word in L, then there is a sequence $c_1, c_2, \ldots, c_{|w|}$ such that if we feed w into M and choose the c_ith transition at the ith step, we reach an accepting state. On the other hand, if $w \notin L$, there will be no such sequence. Therefore, one can build L' from L by attaching $@\langle c_1, c_2, \ldots, c_{|w|}\rangle$ at the end of every $w \in L$, where $\langle c_1, c_2, \ldots, c_{|w|}\rangle$ denotes any encoding of the sequence using the alphabet Σ. □

The above-mentioned lemma can be used to show that L_{pal} is not in NMHFA. For the sake of contradiction, suppose $L_{\text{pal}} \in$ NMHFA(k) for some k. Let L'_{pal} be the language that can be constructed from L_{pal} as in Lemma 8. Then, $L'_{\text{pal}} \in$ DMHFA($k+1$). However, an argument very similar to the one in the proof of Proposition 3 can be used to show that $L'_{\text{pal}} \notin$ DMHFA($k+1$).

Proposition 5. $L_{pal} \notin$ NMHFA.

Corollary 3. *Neither* DMHFA *nor* NMHFA *is comparable with either* DPCM *or* NPCM.

References

1. Baker, B.S., Book, R.V.: Reversal-bounded multipushdown machines. Journal of Computer and System Sciences 8, 315–332 (1974)
2. Daley, M., Ibarra, O., Kari, L.: Closure and decidability properties of some language classes with respect to ciliate bio-operations. Theoretical Computer Science 306, 19–38 (2003)
3. Duris, P., Galil, Z.: On reversal-bounded counter machines and on pushdown automata with a bound on the size of the pushdown store. Information and Control 54, 217–227 (1982)
4. Eğecioğlu, Ö., Ibarra, O.H.: On stateless multicounter machines. In: Ambos-Spies, K., Löwe, B., Merkle, W. (eds.) CiE 2009. LNCS, vol. 5635, pp. 178–187. Springer, Heidelberg (2009)
5. Harju, T., Ibarra, O.H., Karhumäki, J., Salomaa, A.: Some decision problems concerning semilinearity and commutation. Journal of Computer and System Sciences 65, 278–294 (2002)
6. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
7. Ibarra, O.H.: Reversal-bounded multicounter machines and their decision problems. Journal of the ACM 25, 116–133 (1978)
8. Ibarra, O.H., Eğecioğlu, Ö.: Hierarchies and characterizations of stateless multicounter machines. In: Ngo, H.Q. (ed.) COCOON 2009. LNCS, vol. 5609, pp. 408–417. Springer, Heidelberg (2009)
9. Ibarra, O.H., Karhumäki, J., Okhotin, A.: On stateless multihead automata: Hierarchies and the emptiness problem. Theoretical Computer Science 411, 581–593 (2010)
10. Ibarra, O.H., Woodworth, S., Yu, F., Păun, A.: On spiking neural P systems and partially blind counter machines. Natural Computing 7, 3–19 (2008)
11. Li, M., Yesha, Y.: String-matching cannot be done by a two-head one-way deterministic finite automaton. Information Processing Letters 22, 231–235 (1986)
12. Minsky, M.L.: Recursive unsolvability of Post's problem of "tag" and other topics in theory of turing machines. Annals of Mathematics 74, 437–455 (1961)
13. Rosenberg, A.L.: On multi-head finite automata. IBM Journal of Research and Development 10, 388–394 (1966)
14. Sénizergues, G.: $l(a) = l(b)$? a simplified decidability proof. Theoretical Computer Science 281, 555–608 (2002)
15. Yao, A.C., Rivest, R.L.: $k + 1$ heads are better than k. Journal of the ACM 25(2), 337–340 (1978)
16. Yu, S.: A pumping lemma for deterministic context-free languages. Information Processing Letters 31, 47–51 (1989)
17. Yu, S.: Regular languages. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 1, pp. 41–110. Springer, Heidelberg (1997)

Collisionless Gathering of Robots with an Extent*

Andreas Cord-Landwehr, Bastian Degener, Matthias Fischer, Martina Hüllmann, Barbara Kempkes, Alexander Klaas, Peter Kling, Sven Kurras, Marcus Märtens, Friedhelm Meyer auf der Heide, Christoph Raupach, Kamil Swierkot, Daniel Warner, Christoph Weddemann, and Daniel Wonisch

Heinz Nixdorf Institute, Computer Science Department, University of Paderborn

Abstract. Gathering n mobile robots in one single point in the Euclidean plane is a widely studied problem from the area of robot formation problems. Classically, the robots are assumed to have no physical extent, and they are able to share a position with other robots. We drop these assumptions and investigate a similar problem for robots with (a spherical) extent: the goal is to gather the robots as close together as possible. More exactly, we want the robots to form a sphere with minimum radius around a predefined point. We propose an algorithm for this problem which synchronously moves the robots towards the center of the sphere unless they block each other. In this case, if possible, the robots spin around the center of the sphere. We analyze this algorithm experimentally in the plane. If R is the distance of the farthest robot to the center of the sphere, the simulations indicate a runtime which is linear in n and R. Additionally, we prove a theoretic upper bound for the runtime of $O(nR)$ for a discrete version of the problem. Simulations also suggest a runtime of $O(n + R)$ for the discrete version.

1 Introduction

Given n robots which are distributed in the plane, the goal of the traditional gathering problem is to let all robots gather in one point. This problem is one of the simplest problems from the area of robot formation problems. Therefore, a lot of effort has been put into pinpointing the needed capabilities and possible limitations of robots that are able to perform gathering in finite time and in different time models. Mostly, the assumptions are quite theoretical: Robots occupy only one single point in the plane and may share a position, viewing ranges are exact spheres etc. Recently, the main focus is on more realistic assumptions regarding the robot abilities. Especially robustness against faulty compasses or other types of imprecise measurements is considered. However, little effort has been put into the fact that in a realistic setting, gathering on a single point for a reasonable number of robots is not possible due to their physical extent. Even

* Partially supported by the EU within FP7-ICT-2007-1 under contract no. 215270 (FRONTS) and DFG-project "Smart Teams" within the SPP 1183 "Organic Computing" and International Graduate School "Dynamic Intelligent Systems".

worse, colliding robots could prevent each other from moving toward some target at all. The focus of this paper is to overcome these inherent obstacles by modeling robots with an extent rather than as simple points. Assuming the robots to have a circular shape, gathering in this context means letting the robots gather in an area (of circular shape) which is as small as possible. During the movement robots are not allowed to collide. We take a local approach to this problem: When deciding where to move, robots only take their direct neighborhood into account. Moreover, our goal is to not only show that gathering the robots in this setting is possible in finite time, but we also analyze the time it takes until the robots have reached a close-to-optimal or optimal gathering.

Our Contribution. We present a simple, local algorithm in two variants: one for gathering in a continuous environment and in continuous time, and one for a setting with a discrete environment and discrete time. We provide a runtime bound of $O(nR)$ for the latter setting and give experimental insight that the runtime in general could be of order $O(n + R)$, where n is the number of robots and R the diameter of the system. To enforce this conjecture, a statistical analysis using the mean value of the measured running time and the difference to its 95%-confidence interval is presented.

Related Work. While the gathering problem is well studied, most studies assume that several robots can share the same position. The main focus in the literature is to limit the needed capabilities of robots in a given time model such that gathering is still possible. For instance, authors often assume robots not to know their ID (*anonymity*), not to remember their past (*oblivious*), and not to have a common sense of direction (*disoriented*). In [10] it was shown that robots which are anonymous, disorientated, oblivious, and can not communicate are able to gather in a semi-synchronous time model if and only if n is odd. Another example is [19], where two robots gather with inaccurate compasses or [14], where the compasses even vary over time. There is also work for gathering in graphs instead of Euclidean spaces [9,15], and in models with uncertainty [17]. Impossibility results are also known. For instance, in [18] it is shown under which circumstances robots are unable to gather. In [13] an exponential lower bound for randomized algorithms is shown assuming very restricted robots. They also propose a linear time gathering algorithm on the base of multiplicity detection. If at least one robot behaves maliciously, gathering is only possible if there are at least three robots in total [1]. Another important aspect is robustness [6]. Since robots in practice cannot look arbitrarily far, recently robots with local view were considered [7,8,3,2]. A similar problem considered under local view is to transform a long, winding chain of robots into a short one [16,11]. There are also runtime bounds given, which are, in contrast to our work, rarely stated for gathering. A notable exception is [4], where an upper bound of $O(n^2)$ for the easier convergence problem in several time models is shown. Closest to our work is [5], where the authors solve a different gathering problem for robots with an extent. Similar to our model, robot collisions are prohibited. In spite of this, the main problem tackled by the authors is that the line of sight of one robot may be blocked by the extent of another. While the goal in [5] is to gather the robots in such a way that the union of all (circular) robot shapes is connected, our goal is to gather the robots (at a fixed position) as dense as possible. The authors use an asynchronous time model for which no runtime bounds are shown.

2 Basic Model and Problem Description

Consider a set $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ of n robots in the 2–dimensional Euclidean space $\mathbb{R}^2$. We imagine the robots to have a spherical shape of diameter 1. By $p_i \in \mathbb{R}^2$ we indicate the center of the robot r_i, called its *position*. We do not allow any two robots to penetrate each other, thus, the Euclidean distance between robot positions may not drop below 1. Our goal is to let the robots gather in an area which is as small as possible. To describe the problem formally, we introduce the notion of the robots' *C-miniball* $\mathcal{B}_C$, which is the smallest sphere around all robots centered in point C. The problem $\texttt{T-GATHER}_d$ is now as follows: given a set $\mathcal{R}$ of robots in $\mathbb{R}^2$ and a *gathering point* $T \in \mathbb{R}^2$, how must the robots move such that the diameter of their T–miniball becomes minimal?

Our algorithms can be executed by very simple robots. The robots have a limited viewing range in which they observe the positions of neighbors. This viewing range must only be large enough to avoid collisions. Thus, a viewing range of 2 is sufficient. The only global information the robots have is the gathering point: each robot has its own local coordinate system, it knows the position of the gathering point and its own position with respect to its own coordinate system. Memory is only needed for the gathering point, that is, robots do not remember past situations: a robot's decision at a time t is only based on the positions of the robots within the viewing range at time t and the gathering point. Thus, the robots are oblivious. Moreover, the robots do not communicate[1] and they do not need to distinguish robots from each other. The described model is used in both the continuous (Section 3) and the discrete (Section 4) setting. However, we left some aspects of the model—especially the time model—undefined. They depend on the actual setting and are introduced in the corresponding sections.

3 The Continuous Setting

In this section we describe and study the algorithm PULLSPIN in a continuous environment. Continuous models are rather uncommon in this research area: while the gathering problem has been considered in several different time models (synchronous, semi-synchronous, asynchronous, etc.; see for example [12] for an introduction), nearly all results use a discrete time model. That is, the robots observe their environment, perform computations on these observations and eventually move to a new position. However, in reality these observations are often made (nearly) continuously and the entities react (nearly) immediately on their input. The theoretic handling of such continuous models seems rather

[1] In the discrete setting, we do symmetry breaking in a global fashion. A local implementation would need communication between nearby robots. Still, the robots being oblivious, communication cannot be used to compute a global solution.

complex and very different from their discrete counterparts. A recent example for a problem that has been considered in a discrete *and* continuous environment is the communication chain problem [8]. Due to the apparent complexity of a theoretic analysis, we will only provide an experimental evaluation of the algorithm in several different situations. However, in Section 4 we consider the problem in a discrete environment. This simplification will allow us to prove several theoretical results about the discrete version of the PullSpin algorithm. Note that we consider an idealized, continuous time model. That is, the robots move on continuous trajectories and continuously perceive other robots within their viewing range. In our experiments, we approximated this model by very short discrete simulation steps.

Algorithm Description. Our PullSpin algorithm is named after and build around two central operations: the `PULL` and `SPIN` operations. The `PULL` operation causes the robots to approach the gathering point. If pulling further would cause them to penetrate each other, we say that they are *pull-blocked*. In this case, the robots use the `SPIN` operation to rotate around the gathering point. By choosing the rotation direction uniformly at random, the robots may move away from each other and thereby re-enable `PULL` operations. The distance covered by a `SPIN` operation is called *spin parameter*. Its choice has an impact on the running time and success of the algorithm.

It is easy to see that this algorithm does not guarantee the robots to gather optimally around the gathering point. Indeed Figure 1c depicts a situation where robots have not yet reached a very dense gathering around the gathering point: the corresponding miniball's radius is linear in the number of robots. No robot can perform a `PULL` operation without penetrating its neighbors. Moreover, all possible spin directions are blocked by other robots. That is, these robots are pull- and *spin-blocked*. Such a situation is called a *deadlock*.

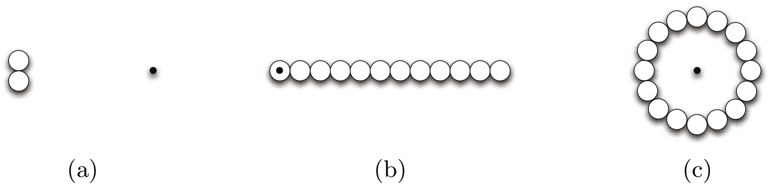


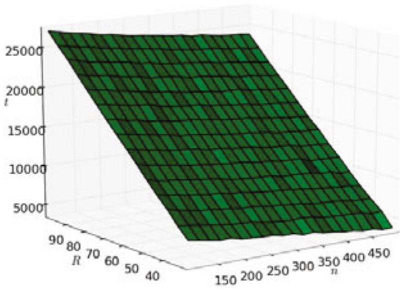
Fig. 1. Three different kinds of pull- and/or spin-blocks. While the T-miniballs corresponding to (b) and (c) have a diameter that is linear in the number of robots, the corresponding diameter for (a) may be arbitrarily large. The situations (a) and (b) can be resolved by spin operations.

However, while our algorithm is—in the continuous model—not even guaranteed to lead to a nearly optimal gathering (i.e., optimal up to a constant), experiments suggest that such deadlock situations are unstable and very unlikely to occur, at least if the robots do not already start in such a highly symmetric situation. In our experiments, typical results were dense packings with a few robots still performing spin operations, but unable to find a space for a pull-operation. Note however, that the algorithm for the discrete problem variant described in Section 4 always achieves optimal configurations.

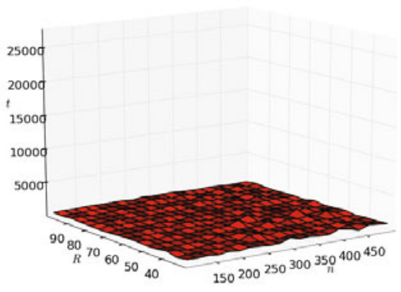
Experiments. We designed several experiments to evaluate the quality of the PullSpin-algorithm. The idealized continuous time model was approximated by using a discrete model and restricting the distance robots can move during one *simulation step* to the relatively small value of 0.01. For the experiments, we created several initial arrangements differing in the number of robots n and the maximal distance R of a robot to the gathering point. Our main interest was to capture interesting and hard configurations. There are two extreme cases yielding two different problems: widespread configurations (robots have to travel a large distance) and configurations where some robots are close together (and block each other). To capture the former (homogeneous arrangement), every robot was given an integer starting position taken uniformly at random from the sphere of radius R around the gathering point. For the latter, four cluster centers were chosen uniformly at random within radius R of the gathering point. The robots were evenly distributed on the clusters in compact spheres. Instances with overlapping clusters were discarded. For every arrangement, it was validated that there is at least one robot of distance R to the gathering point and that every pair of robots obeys the minimum distance of 1.0 to each other. The `SPIN` operation was implemented as follows: whenever a robot is pull-blocked in the last step, it spins clock- or counterclockwise (uniformly at random) around the gathering point for a distance of at most 2.0 or until it becomes blocked by another robot[2]. Afterward, the robot tries to `PULL` again. Due to the random nature of the initial arrangements, we created 25 different starting settings for each parameter tuple (n, R). We measured the number of simulation steps it took until all robots reached a "close to optimal" configuration. More precisely, we stopped a run when all robots were contained in a sphere of radius $2 \cdot r$ around the gathering point. Here, $r = \frac{1}{2}\sqrt{n}$ is a lower bound of the optimal radius (the radius of a sphere with the same volume as all robots together).

As expected, the PullSpin algorithm has no problems to deal with the homogeneous arrangements. At the beginning of these experiments, almost every robot is able to perform several `PULL` operations until it is forced to `SPIN` for the first time. The number of spin operations increases as the region around the gathering point becomes more and more dense. Additional problems occur in the inhomogeneous arrangements. In these arrangements almost all robots block each other initially, preventing them from approaching the origin. This way they force each other to perform `SPIN` operations. However, after a small amount of time all the clusters become loose enough such that most robots are able to continue with `PULL` operations. Even though many robots may reach the gathering point from the same direction in these arrangements (by a corresponding choice of the clusters), it seems that—mainly due to the `SPIN` operation—deadlock situations are very rare: only in 29 out of 5525 runs the termination criterion of our

[2] This proved to be a sensible choice in some preliminary experiments.

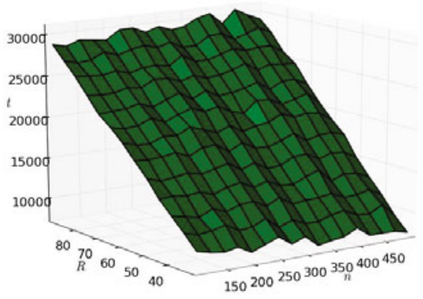


(a) Mean of running time

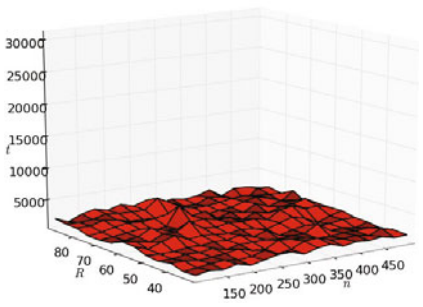


(b) Difference of the boundary of the 95%-conf. interval to the mean values

Fig. 2. Homogeneous PullSpin experiments for different (n, R)-tuples. The running time t is measured in simulation steps.



(a) Mean of running time



(b) Difference of the boundary of the 95%-conf. interval to the mean values

Fig. 3. Inhomogeneous PullSpin experiments for different (n, R)-tuples. The running time t is measured in simulation steps.

experimental analysis was not reached within a feasible time (100,000 simulation steps)[3]. Furthermore, note that at most two of them occurred for the same pair of n and R values.

Figures 2 and 3 show the mean value and the difference between the boundary of the 95% confidence interval and the mean value. For example, Figure 2a shows the mean values we measured over all runs in the homogeneous PullSpin arrangements. Figure 2b indicates how much this mean value differs from the upper bound of the corresponding 95% confidence interval. As can be seen in both the homogeneous and inhomogeneous PullSpin arrangements, the difference is rather small. Together, the diagrams indicate that the average running time of PullSpin is linear in the parameters n and R for homogeneous and inhomogeneous arrangements. The initial radius R seems to be more dominating than the number of robots n. Surprisingly, for fixed R the mean running time even

[3] These were considered as outliers and not included in the statistical analysis.

decreases for large values of n, due to the fact that we stopped the simulations on arrival of the last robot in some nearly optimal sphere as described above. Since the radius of this sphere grows for large n, the simulation terminates earlier, compensating the effect of additional collisions by the higher amount of robots.

4 The Discrete Setting

In the discrete setting, we no longer allow the robots to move arbitrarily in $\mathbb{R}^2$, but restrict them to positions on the 2-dimensional integral grid $\mathbb{Z}^2$. This discretization yields several advantages for the analysis, while the central difficulties still hold. Most important, robot movements are easy to describe and the property that robots may not collide due to their extent is properly captured. As in the continuous setting, we want the robots to gather around the origin. However, we use a more natural distance notion on the grid: the metric induced by the L^1*-norm* (also known as the taxicab metric or Manhattan distance). We use $\|x\|_1 := |x_1| + |x_2|$ to refer to the L^1-norm of $x \in \mathbb{Z}^2$. Given the restriction to discrete robot positions and the finite number of robots, we can, w.l.o.g., assume a discrete time model starting at time $t = 0$. A robot that wants to move to a (free) neighbored grid point $p \in \mathbb{Z}^2$ will immediately reach p in the next time step. To avoid collisions, we do not allow two distinct robots to be placed on the same position. Our goal is to find a gathering that minimizes the radius of the minimal L^1*-sphere* containing all robots. We define $B(k) := \{x \in \mathbb{Z}^d \mid \|x\|_1 \leq k\}$ as the (filled) L^1-sphere and $B^\circ(k) := \{k \in \mathbb{Z}^d \mid \|x\|_1 = k\}$ as the hollow L^1-sphere of radius $k \in \mathbb{N}$. It is obvious that, given exactly $|B(k)|$ robots, there is a unique optimal gathering: the gathering where all robots lie on a grid point within $B(k)$. To simplify the analysis, in the remainder we will assume, w.l.o.g., that the number n of robots is of the form $|B(k)|$ for some $k \in \mathbb{N}$.

Algorithm Description. In the following, we present a discretization of the PullSpin algorithm named GridPullSpin. In contrast to the original PullSpin algorithm, GridPullSpin is deterministic. This eases the analysis. Once more, the discrete algorithm is based on two operations: `PULL` and `SPIN` (discrete variants of the corresponding continuous operations). They are implicitly defined as follows: `PULL` decreases the distance to the origin by exactly one. `SPIN` increases/decreases the distance to the origin *in every dimension* by at most one, such that the distance to the origin *does not change*. As illustrated in Figure 4, robots may have the choice between several `PULL` and/or `SPIN` operations. In such situations, our GridPullSpin algorithm chooses from the available operations in the following order: `PULL` in x-direction, `PULL` in y-direction, counterclockwise `SPIN` operation.

The notions introduced in Section 3 (e.g., pull-blocked, spin-blocked, ...) apply accordingly to the discrete setting. As in the continuous setting, there are situations where *all* robots are pull-blocked, even if they have not reached an optimal gathering (see Figure 5). Again, `SPIN` operations are used to resolve pull-blocks. Note that there are no deadlocks (other than the optimal situation) in the discrete setting. In fact, the GridPullSpin algorithm is guaranteed to

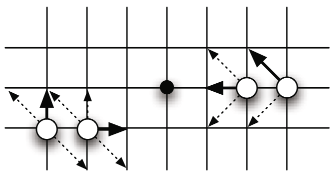


Fig. 4. Possible PULL/SPIN operations (dashed) and choices of GridPullSpin (solid)

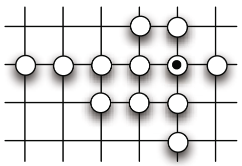


Fig. 5. Non-optimal and pull-blocked gathering

lead to an optimal gathering around the origin, as we will prove in Theorem 1. If several robots simultaneously move to the same position, we arbitrarily select one of them, favoring robots performing a PULL operation[4]. This robot performs its move as usual, while the remaining robots idle for one step.

GridPullSpin Analysis. We begin by showing that the GridPullSpin algorithm leads to an optimal gathering in a finite amount of time. Afterward, we will carefully analyze *how long* the algorithm needs to achieve such an optimal gathering and prove an upper time bound of $\mathcal{O}(nR)$.

Theorem 1. *Given an arbitrary initial arrangement of n robots on the grid $\mathbb{Z}^2$,* GridPullSpin *leads to an optimal gathering of the robots around the origin.*

Proof. First note that the total number of PULL operations is finite. Assuming the robots have not yet reached an optimal gathering at time $t \in \mathbb{N}$, consider the innermost level $l_{\min}$ around the origin that has less robots than in the optimal gathering. Furthermore, let $l_{next} > l_{\min}$ denote the next level that contains at least one robot at time t. If $l_{next} > l_{\min} + 1$, then at least one robot at level l_{next} performs a PULL operation at the next time step. Otherwise, the robots at level l_{next} will perform SPIN operations until one of them perceives a free position at level $l_{\min}$ and performs a PULL operation. Note that there must be a free position somewhere at level $l_{\min}$ and, because of the robots spinning deterministically counterclockwise around the origin, one of them will perceive this free position after at most $4(l_{\min} + 1)$ steps. Since only finitely many PULL operations are performed, an optimal gathering must be reached within finite time. □

Theorem 2. *Given n robots on the 2-dimensional grid $\mathbb{Z}^2$,* GridPullSpin *leads to an optimal gathering around the origin after at most $3nR$ steps.*

Before we prove this Theorem, remember that we assume (w.l.o.g.) the number of robots to be of the form $n = |B(k)|$ for some $k \in \mathbb{N}$. Thus, the robots have reached an optimal gathering around the origin if and only if each point in $B(k)$ is occupied. That is, $B(k)$ forms the set of *target positions*. The target positions

[4] This operation can be implemented locally if robots can communicate and have either IDs or use randomization. In the latter case, each of the (at most four) robots tosses a coin. Only if exactly one of them tosses head, this robot moves to the position. In expectation, it takes a constant number of tosses until one of the robots proceeds.

$B(k)$ can be partitioned into $k+1$ levels B_i ($i = 0, \dots, k$) with $B_i := B^\circ(i)$. We will refer to B_i as the i-th level. A target position $x \in B_i$ can be in one of two states: For $i = 0$, x is called *occupied* if a robot is at position x. If $i > 0$, x is called *occupied* if all target positions on levels $j < i$ are occupied and there is a robot at position x. If x is not occupied, it is called *free*. We call level i *occupied* if all target positions in B_i are occupied. An occupied target position can become free again: the occupying robot may simply move away. However, once a level is occupied, it can not become free anymore. Let us gather this and some other basic observations:

Observation 1
(a) Once a level is occupied, it cannot become free anymore.
(b) The number of occupied target positions in B_i is monotonically increasing.
(c) Level k is occupied iff the robots have reached the optimal gathering.

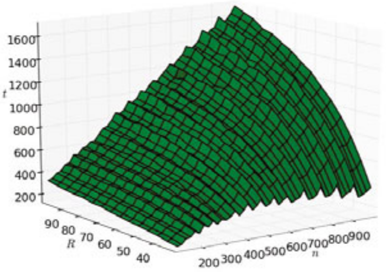
To determine how long it takes to occupy level k, we define a total order on the robots. Theorem 1 and Observation 1.c guarantee that eventually all target positions become occupied. Let us consider the robot that increases the number of occupied target positions on level $i \in \{1, \dots, k\}$ from $j-1$ to $j \in \{1, 2, \dots, 4i\}$ and refer to it by $r_{i,j}$, breaking ties arbitrarily. The robot eventually occupying B_0 is referred to by $r_{0,1}$. This defines a bijection between the set of robots and the corresponding indices (i, j). We use these indices to order the robots lexicographically and write $r_{i,j} < r_{i',j'}$ if $(i, j) < (i', j')$. We call robot $r_{i,j}$ *settled* when it caused the j-th target position of level i to become occupied.

Proposition 1
(a) If $r_{i,j} < r_{i',j'}$, $r_{i,j}$ will be settled before or at the same time as $r_{i',j'}$.
(b) Robot $r_{0,1}$ becomes settled after at most R steps.
(c) If all robots $r < r_{i,j}$ are settled, $r_{i,j}$ becomes settled after at most $3R$ steps.

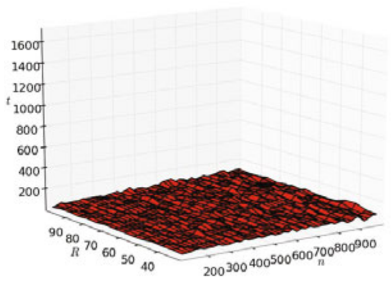
Proof
(a) This follows immediately from the definition of the total order on the robots.
(b) Let $\hat{t}$ denote the time when $r_{0,1}$ becomes settled and consider an arbitrary time $t < \hat{t}$. Let $\mathcal{R}_t$ denote the set of robots having minimal distance $\delta_t > 0$ to the origin. All robots in $\mathcal{R}_t$ will perform a `PULL` operation, some of them not succeeding in case of competition for the same position. This implies $\emptyset \neq \mathcal{R}_{t+1} \subseteq \mathcal{R}_t$. At time $\hat{t} - 1$, only one robot will succeed with its `PULL` operation: robot $r_{0,1}$. Now we have $r_{0,1} \in \mathcal{R}_{\hat{t}} \subseteq \mathcal{R}_{\hat{t}-1} \subseteq \ldots \subseteq \mathcal{R}_0$. That is, $r_{0,1}$ always performed a (successful) `PULL` operation. This implies $\hat{t} = \delta_0 \leq R$.
(c) Consider robot $r_{i,j}$ at time t_0 when all robots $r < r_{i,j}$ have been settled. All of them lie on a level $\leq i$. For a time $t \geq t_0$, let $\mathcal{R}_t$ denote the set of robots having minimal distance $\delta_t > i$ to the origin. Similar to the proof of Proposition 1.b, we can identify a robot in $\mathcal{R}_{t_0}$ that performs only `PULL` operations until it reaches level $i+1$. Thus, there is a robot $r \in \mathcal{R}_{t_0}$ that reaches level $i+1$ after $\delta - i - 1 < R$ steps. Having reached level $i+1$, it may now be pull-blocked by an already settled robot at level i. Let us denote this time with $t_1 < t_0 + R$ and consider two cases: If at least j target positions in level i are occupied, robot $r_{i,j}$ has been settled

at a time $\leq t_1 < t_0 + R < t_0 + 3R$ (by definition). Otherwise, if at most $j - 1$ target positions in level i are occupied, then—since all robots r with $r < r_{i,j}$ have been settled—there are *exactly* $j - 1$ occupied target positions in level i. Furthermore, there is a free target position somewhere on level i. Robot r will `SPIN` counterclockwise, while the target position "spins" clockwise. If r is not spin-blocked during these steps, it will reach the target position after at most $4i/2 = 2i \leq 2R$ steps. If r is spin-blocked by a robot r', we can iterate the argument on r' until we find a robot r'' that is not spin-blocked before it reaches the free target position. Therefore, after at most $2R$ steps, another target position on level i has been occupied. Thus, the j-th target position on level i must have been occupied at a time $\leq t_1 + 2R < t_0 + 3R$. In both cases, robot $r_{i,j}$ has been settled after at most $3R$ steps. □

Proof (of Theorem 2). Proposition 1 immediately implies that it takes at most $n \cdot 3R$ steps until all robots are settled. Especially, since all level k marks have been occupied, the robots have reached an optimal gathering (Observation 1). □



(a) Mean of running time.

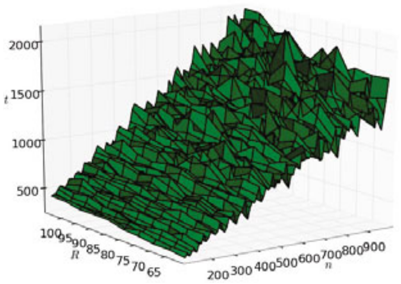


(b) Difference of the boundary of the 95%-conf. interval to the mean values.

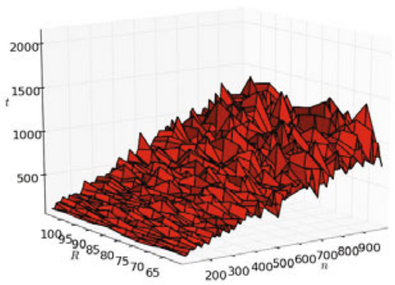
Fig. 6. Homogeneous GridPullSpin experiments for different (n, R)-tuples

Experiments. Similar to the experiments in Section 3, we used homogeneous and inhomogeneous arrangements to test the capabilities of GridPullSpin. The experimental setting differs from the one in Section 3 in the several aspects. On the one hand, for all distances the L^1-norm is used instead of the Euclidean norm. On the other hand, the maximum movement distance is 2 (in L_1-norm), since GridPullSpin moves robots by at most one in each dimension. Furthermore, note that the criterion for termination is the achievement of an optimal gathering. Apparently, GridPullSpin needs more time to terminate in the inhomogeneous arrangements (Figure 7) than in the homogeneous arrangements (Figure 6). One has to be careful when comparing these results to the corresponding results in the continuous setting. By the choice of our parameters, the robots in the continuous setting cover much smaller distances in one time unit. However, in both cases the experiments clearly indicate a linear dependence in n

and R of the average running time. The stair-like artifacts in Figure 6 stem from the fact that the radius of the optimal gathering is not directly proportional to the number of robots: an almost empty last level is easier to reach than one that is almost full. The decreasing running time for growing n and small, fixed R can be explained by the low difference of these arrangements from an optimal gathering. For inhomogeneous arrangements the deviation of the confidence interval boundaries seems (in contrast to all other cases) not to be constant, but linear dependent with respect to n. However, this does not affect the observed averaged linear running time.



(a) Mean of running time.



(b) Difference of the boundary of the 95%-conf. interval to the mean values.

Fig. 7. Inhomogeneous GridPullSpin experiments for different (n, R)-tuples

5 Conclusion and Future Work

We conjecture that our runtime bound can be improved to $O(n + R)$. After all, our experiments suggest this bound and our proof ignores the parallel movement of the robots. Moreover, we would like to extend our algorithms to arbitrary dimensions. To do this in the discrete setting it seems crucial to guarantee that robots spin around an almost compact sphere in such a way that they find the possibly sole gap they can pull into. This is related to the exploration of sphere surfaces, only that the gap may move as well. The experiments seem to suggest that similar runtime bounds apply to the continuous setting. Here proofs are still needed. A first step is to discretize only the time, such that the robots' movement distance is bounded to a small value at every time step.

References

1. Agmon, N., Peleg, D.: Fault-tolerant gathering algorithms for autonomous mobile robots. In: Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1070–1078. SIAM (2004)
2. Ando, H., Oasa, Y., Suzuki, I., Yamashita, M.: Distributed memoryless point convergence algorithm for mobile robots with limited visibility. IEEE Transactions on Robotics and Automation 15(5), 818–828 (1999)

3. Ando, H., Suzuki, Y., Yamashita, M.: Formation agreement problems for synchronous mobile robotswith limited visibility. In: Proc. IEEE Syp. of Intelligent Control, pp. 453–460 (1995)
4. Cohen, R., Peleg, D.: Convergence properties of the gravitational algorithm in asynchronous robot systems. SIAM Journal on Computing 34(6), 1516–1528 (2005)
5. Czyzowicz, J., Gasieniec, L., Pelc, A.: Gathering few fat mobile robots in the plane. Theoretical Computer Science 410(6-7), 481–499 (2009)
6. Défago, X., Gradinariu, M., Messika, S., Raipin-Parvédy, P.: Fault-tolerant and self-stabilizing mobile robots gathering. In: Dolev, S. (ed.) DISC 2006. LNCS, vol. 4167, pp. 46–60. Springer, Heidelberg (2006)
7. Degener, B., Kempkes, B., auf der Heide, F.M.: A local $O(n^2)$ gathering algorithm. In: Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures, pp. 217–223. ACM, New York (2010)
8. Degener, B., Kempkes, B., Kling, P., Meyer auf der Heide, F.: A continuous, local strategy for constructing a short chain of mobile robots. In: Patt-Shamir, B., Ekim, T. (eds.) SIROCCO 2010. LNCS, vol. 6058, pp. 168–182. Springer, Heidelberg (2010)
9. Dessmark, A., Fraigniaud, P., Kowalski, D.R., Pelc, A.: Deterministic rendezvous in graphs. Algorithmica 46(1), 69–96 (2006)
10. Dieudonné, Y., Petit, F.: Self-stabilizing deterministic gathering. In: Dolev, S. (ed.) ALGOSENSORS 2009. LNCS, vol. 5804, pp. 230–241. Springer, Heidelberg (2009)
11. Dynia, M., Kutylowski, J., Lorek, P., auf der Heide, F.M.: Maintaining communication between an explorer and a base station. In: World Computer Congress: International Conference on Biologically Inspired Computing. pp. 137–146 (2006)
12. Efrima, A., Peleg, D.: Distributed Models and Algorithms for Mobile Robot Systems. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 70–87. Springer, Heidelberg (2007)
13. Izumi, T., Izumi, T., Kamei, S., Ooshita, F.: Randomized gathering of mobile robots with local-multiplicity detection. In: Guerraoui, R., Petit, F. (eds.) SSS 2009. LNCS, vol. 5873, pp. 384–398. Springer, Heidelberg (2009)
14. Izumi, T., Katayama, Y., Inuzuka, N., Wada, K.: Gathering autonomous mobile robots with dynamic compasses: An optimal result. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 298–312. Springer, Heidelberg (2007)
15. Klasing, R., Markou, E., Pelc, A.: Gathering asynchronous oblivious mobile robots in a ring. Theoretical Computer Science 390(1), 27–39 (2008)
16. Kutylowski, J., Meyer auf der Heide, F.: Optimal strategies for maintaining a chain of relays between an explorer and a base camp. Theoretical Computer Science 410(36), 3391–3405 (2009)
17. Martínez, S.: Practical multiagent rendezvous through modified circumcenter algorithms. Automatica 45(9), 2010–2017 (2009)
18. Prencipe, G.: Impossibility of gathering by a set of autonomous mobile robots. Theoretical Computer Science 384(2-3), 222–231 (2007)
19. Souissi, S., Défago, X., Yamashita, M.: Gathering asynchronous mobile robots with inaccurate compasses. In: Shvartsman, M.M.A.A. (ed.) OPODIS 2006. LNCS, vol. 4305, pp. 333–349. Springer, Heidelberg (2006)

Min-Max Coverage in Multi-interface Networks

Gianlorenzo D'Angelo[1], Gabriele Di Stefano[1], and Alfredo Navarra[2]

[1] Dipartimento di Ingegneria Elettrica e dell'Informazione, Università degli Studi dell'Aquila, Italy
gianlorenzo.dangelo@univaq.it, gabriele.distefano@univaq.it

[2] Dipartimento di Matematica e Informatica, Università degli Studi di Perugia, Italy
navarra@dmi.unipg.it

Abstract. We consider devices equipped with multiple wired or wireless interfaces. By switching among interfaces or by combining the available interfaces, each device might establish several connections. A connection is established when the devices at its endpoints share at least one active interface. Each interface is assumed to require an activation cost. In this paper, we consider the problem of establishing the connections defined by a network $G = (V, E)$ while keeping as low as possible the maximum cost set of active interfaces at the single nodes. Nodes V represent the devices, edges E represent the connections that must be established. We study the problem of minimizing the maximum cost set of active interfaces among the nodes of the network in order to cover all the edges. We prove that the problem is NP-hard for any fixed $\Delta \geq 5$ and $k \geq 16$, with Δ being the maximum degree, and k being the number of different interfaces among the network. We also show that the problem cannot be approximated within $\Omega(\ln \Delta)$. We then provide a general approximation algorithm which guarantees a factor of $O((1 + b)\ln(\Delta))$, with b being a parameter depending on the topology of the input graph. Interestingly, b can be bounded by a constant for many graph classes. Other approximation and exact algorithms for special cases are presented.

1 Introduction

The heterogeneity of modern devices poses new challenges to the scientific community. The interest is also increased by the wide range of real-world applications inferred. For instance, the equipment of recent devices provides users with the opportunity to access to different networks by means of the selection of suitable interfaces. Classical problems related to wired and wireless networks can be reconsidered with respect to the new environment. Different computational power, energy consumption, radio interfaces, supported communication protocols, and other peculiarities can characterize the involved devices. In this paper, we are interested in multiple interfaces equipments where a connection between two or more devices might be accomplished by means of different communication networks according to provided requirements. The selection of the most suitable interface for a specific connection might depend on various factors. Such factors include: its availability in specific devices, the cost (in terms of energy consumption) of maintaining an active interface, the available neighbors, and so forth.

While managing such connections, a lot of effort must be devoted to energy consumption issues. Devices are, in fact, usually battery powered and the network survivability might depend on their persistence in the network.

We study communication problems in wireless networks supporting multiple interfaces. In the considered model, the input network is described by a graph $G = (V, E)$, where V represents the set of wireless devices and E is the set of required connections according to proximity of devices and the available interfaces that they may share. Each $v \in V$ is associated with a set of available interfaces $W(v)$. The set of all the possible interfaces available in the network is then determined by $\bigcup_{v \in V} W(v)$; we denote the cardinality of this set by k. We say that a connection is covered when the endpoints of the corresponding edge share at least one active interface. If an interface x is activated at some node u, then u consumes some energy $c(x)$ for maintaining x as active. In this setting, we study the problem of covering all the edges of G by minimizing the maximum cost required at the single nodes. This implies that the cost imposed by all the interfaces activated in the whole network to accomplish the coverage requirement might not be the global minimum. Indeed, the chosen requirement is in favor of a uniform energy consumption among the devices, as it tries to maintain as low as possible the maximum cost spent by the single devices. This plays a central role in the context of wireless networks where the whole network survivability might depend on few devices.

1.1 Related Work

Multi-interface wireless networks have been recently studied in a variety of contexts, usually focusing on the benefits of multiple radio devices of each node [6,9,10]. Many basic problems of standard wireless network optimization can be reconsidered in such a setting [2]. However, previous works have been mainly focused on the minimization of the costs among the whole network. In [5,14], for instance, the same problem of *Coverage* has been investigated, but with the goal of activating the minimum cost set of interfaces among all the nodes in the network in such a way that all the edges of G are covered. *Connectivity* issues have been addressed in [1,8,14,15]. The goal becomes to activate the minimum cost set of interfaces in G in order to guarantee a path of communication between every pair of nodes. In particular, [8] considers the connectivity task under the same objective function of this paper, i.e., the minimization of the maximum cost spent by each single node. In [3,15], the attention has been devoted to the so called *Cheapest path* problem. This corresponds to the well-known shortest path problem, but in the context of multi-interface networks.

1.2 Our Results

In this paper, we study the problem of establishing all the connections defined by G which minimize the maximum cost required at the single nodes. We call this problem the Minimum Maximum Cost Coverage problem in Multi-Interface Networks (*MMCC* for short). The chosen requirement is intended as a first step toward distributed environments where the objective function refers to local properties rather than global costs.

We consider two variants of the above problem: the parameter k is either considered as part of the input (this is called the *unbounded case*), or k is a fixed constant (the *bounded case*). The case where the cost function is constant for each interface is called the *unit cost* case.

First, we prove that the problem is *NP*-hard, even for the unit cost case and even when the number of interfaces k and the maximum node degree Δ are fixed. In particular, we prove that the problem remains *NP*-hard for any fixed $\Delta \geq 5$ and $k \geq 16$. Then, we present efficient algorithms that optimally solve the problem in some relevant special cases. In detail, we focus on instances where the input graph is a tree, by giving a polynomial time algorithm for fixed k or fixed Δ. By using this algorithm we can derive efficient algorithms for $\Delta \leq 2$. Furthermore, we give a polynomial time algorithm for $k \leq 3$. For fixed k, $4 \leq k \leq 15$, and fixed Δ, $3 \leq \Delta \leq 4$, the complexity of *MMCC* remains open.

Concerning approximation results for *MMCC*, we show that the problem is not approximable within an $\eta \ln(\Delta)$ factor for a certain constant η, unless $P = NP$. This result holds even in the unit cost case and when the input graph is a tree but only when k or Δ are unbounded. We then provide an approximation algorithm that guarantees a factor of $\ln(\Delta) + 1 + b \cdot \min\{c_{max}, (\ln(\Delta) + 1)\}$, with $c_{max} = \max_{i \in \{1,\ldots k\}} c(i)$ and b being a parameter depending on structural properties of the input graph. Such parameter can be bounded by a constant in many graph classes (see Section 4). Note that, the obtained approximation guarantees a $1+b$ factor from the best possible algorithm. Another approximation factor which is directly implied by [8] is $\frac{k}{2}$. This clearly might be useful for small values of k.

1.3 Structure of the Paper

In the next section, we formally define the problem of covering all the edges of the input graph by minimizing the maximum cost required at the single nodes and give some preliminary results. In Section 3, we study the complexity of *MMCC* by analyzing the cases where the problem is *NP*-hard, and when the problem can be optimally solved. In Section 4, we provide inapproximability results and we present a polynomial time approximation algorithms for both the general case and particular cases. In Section 5, we outline some conclusion and possible future research.

2 Preliminaries and Notation

For a graph G, we denote by V its node set, by E its edge set. We denote the sizes of V and E by n and m, respectively. For any $v \in V$, let $N(v)$ be the set of its neighbors, and $deg(v) = |N(v)|$ be its degree in G. The maximum degree of G is denoted by $\Delta = \max_{v \in V} deg(v)$. Unless otherwise stated, the graph $G = (V, E)$ representing the network is always assumed to be simple (i.e., without multiple edges and loops), undirected and connected.

A global assignment of the interfaces to the nodes in V is given in terms of an appropriate interface assignment function W, according to the following definition.

Definition 1. *A function $W\colon V \to 2^{\{1,2,\ldots,k\}}$ is said to* cover *graph G if for each $\{u,v\} \in E$ we have $W(u) \cap W(v) \neq \emptyset$.*

The cost of activating an interface i is given by the cost function $c\colon \{1,2,\ldots,k\} \to \mathbb{R}_+$ and it is denoted as $c(i)$. It follows that each node holding an interface i pays the same cost $c(i)$ by activating i. The considered *MMCC* optimization problem is formulated as follows.

MMCC: Minimum Maximum Cost Coverage in Multi-Interface Networks	
***Input*:**	A graph $G = (V,E)$, an allocation of available interfaces $W\colon V \to 2^{\{1,2,\ldots,k\}}$ covering graph G, an interface cost function $c\colon \{1,2,\ldots,k\} \to \mathbb{R}_+$.
***Solution*:**	An allocation of active interfaces $W_A\colon V \to 2^{\{1,2,\ldots,k\}}$ covering G such that $W_A(v) \subseteq W(v)$ for all $v \in V$.
***Goal*:**	Minimize the maximum cost of the active interfaces among all the nodes, i.e. $\min_{W_A} \max_{v\in V} \sum_{i\in W_A(v)} c(i)$.

We recall that two variants of the above problem are considered: when the parameter k is part of the input (i.e., the unbounded case), and when k is a fixed constant (i.e., the bounded case). In both cases we assume $k \geq 2$, since the case $k = 1$ admits the obvious solution provided by activating the unique interface at all the nodes.

It is worth to mention that for paths and trees the *MMCC* problem coincides with the Connectivity problem studied in [8] where the aim is to allow a communication path between any pair of nodes. In fact, the following statement holds.

Proposition 1. *When the input graph is a tree, any solution for MMCC is also a solution for Connectivity at the same cost.*

3 Complexity

In this section, we study the complexity of *MMCC*. First, we prove that the problem is *NP*-hard and then we identify special cases where it is polynomially solvable.

Theorem 1. *MMCC is NP-hard even when restricted to the bounded unit cost case, for any fixed $\Delta \geq 5$ and $k \geq 16$.*

Proof. We prove that the underlying decisional problem, denoted by $MMCC_D$, is in general *NP*-complete. We need to add one bound $B \in \mathbb{R}_+$ such that the problem will be to ask whether there exists an activation function which induces a maximum cost of the active interfaces per node of at most B. In detail, *$MMCC_D$* is defined as follows.

$MMCC_D$	
***Input*:**	A graph $G = (V, E)$, an allocation of available interfaces $W \colon V \to 2^{\{1,2,\ldots,k\}}$ covering graph G, an interface cost function $c \colon \{1, 2, \ldots, k\} \to \mathbb{R}_+$, and a bound $B \in \mathbb{R}_+$.
***Question*:**	Is there an allocation of active interfaces $W_A \colon V \to 2^{\{1,2,\ldots,k\}}$ covering G such that $W_A(v) \subseteq W(v)$ for all $v \in V$ and $\max_{v \in V} \sum_{i \in W_A(v)} c(i) \le B$?

The problem is in *NP* as, given an allocation function of active interfaces for an instance of *$MMCC_D$*, to check whether it covers the input graph G with a maximum cost of active interfaces per node of at most B is linear in the size of the instance.

The proof then proceeds by a polynomial reduction from the well-known *Satisfiability* problem. The problem is known to be *NP*-complete [12] and it can be stated as follows.

SAT: Satisfiability	
***Input*:**	Set U of variables and collection C of clauses over U.
***Question*:**	Is there a satisfying truth assignment for C?

SAT remains *NP*-complete even if there are at most three literals for each clause and a variable appears, negated or not, in at most three clauses [12]. Moreover, the problem remains *NP*-complete even if we assume that there are no clauses with a single literal. Then, in the following reduction, we assume that each clause has two or three literals and each variable belongs to at most three clauses.

Given an instance of *SAT*, we can build an instance of *$MMCC_D$* in polynomial time as follows. Let $B = 3$. The graph $G = (V, E)$ of *$MMCC_D$* has, for each variable $u \in U$, three nodes a_u, b_u, c_u in V and two edges $\{a_u, b_u\}$ and $\{a_u, c_u\}$. For each clause $q \in C$, G has two nodes d_q, e_q in V and an edge $\{d_q, e_q\}$. Let $D = \{d_q \in V \mid q \in C\}$. If clause q has two literals, we add a new node f_q and the edge $\{d_q, f_q\}$. Finally, for each variable u and each clause q containing u, the graph G has an edge $\{a_u, d_q\}$.

Note that nodes in G have degree at most five, then $\Delta = 5$.

There are three interfaces I_b, I_c, I_d and, for each variable $u \in U$, two further interfaces: T_u and F_u.

Node a_u has four interfaces: T_u, F_u, I_b, I_c, node b_u has interface I_b and node c_u has interface I_c, for each $u \in U$. For each clause $q \in C$, node d_q has interfaces T_w, F_w for each variable $w \in U$ that appears in q; d_q has an additional interface I_d if q has only two literals; e_q has either interface F_w or T_w, according to whether w is negated in q or not, for each variable $w \in U$ that appears in q. Nodes f_q have only interface I_d for each $q \in C$ having only two literals.

Let us assume that *SAT* admits a satisfying truth assignment for its variables. For each variable $u \in U$, we activate interfaces I_b and I_c in a_u, b_u, c_u, and if u has true (false, resp.) assignment, we activate interface T_u (F_u, resp.) in a_u.

For each clause $q \in C$, and for each variable w in q, we activate on nodes d_q, e_q interfaces T_w if the corresponding literal has a true value, F_w otherwise. Moreover, if q has only two literals, we activate interfaces I_d on nodes d_q and f_q.

Now, the number of active interfaces for each node is at most $B = 3$ and each edge is covered. In fact edges $\{a_u, b_u\}$ and $\{a_u, c_u\}$ are covered by interfaces I_b and I_c, respectively, for each $u \in U$. Each edge $\{d_q, f_q\}$ is covered by interface I_d for each clause $q \in C$ with two literals. As there exists at least one true literal for each clause q, then edge $\{d_q, e_q\}$ is covered by the corresponding interface T_w or F_w according to the literal is w or $\overline{w}$. Finally, for each clause $q \in C$ and for each variable u in q, each edge $\{a_u, d_q\}$ is covered by the interface T_u or F_u according to whether u is true or false, respectively.

On the contrary, let us assume that $MMCC_D$ has a positive answer. Then both interfaces I_b and I_c are active on each node a_u, $u \in U$, to cover edges $\{a_u, b_u\}$ and $\{a_u, c_u\}$. Being $B = 3$, each a_u can activate either T_u or F_u to cover edges connecting it to nodes in D. For each $u \in U$, if a_u activates T_u we assign true to u, otherwise, if a_u activates F_u we assign false to u.

Now, each node $d_q \in D$, where q is a clause in C, activates either interface T_u or interface F_u for each variable u in q (and interface I_d, if it has only two variables, to cover edge $\{d_q, f_q\}$). Being $B = 3$, one of these interface is also used to cover edge $\{d_q, e_q\}$, corresponding to a true value for one literal in q. Then q is satisfied.

This shows that $MMCC_D$ is *NP*-complete. To show that the problem remains *NP*-complete even if k is bounded, note that it is not necessary to use all the interfaces T_u and F_u for each variable $u \in U$. In fact it is sufficient that each node $d \in D$ has a set of distinct interfaces, two for each variable in the corresponding clause. Then, provided that two variables x and y never appear at the same time into a single clause, the pair of interfaces T_x and F_x associated to x can be reused for y.

To assign interfaces to variables, and in particular to nodes in a_u, $u \in U$, we proceed as follows. We build the conflict graph $H = (U, E_H)$, where there is an edge $\{u, v\}$ in E_H between two variables u, v in U if there exists a node $d \in D$ and two edges $\{a_u, d\}$, $\{a_v, y\}$ in G. We find a minimum coloring of H and, if $\chi(u)$ is the color assigned to a variable $u \in U$ we replace the pair of interfaces T_u and F_u with the pair $T_{\chi(u)}$ and $F_{\chi(u)}$ in each node of G. As H has maximum degree 6, it is possible to color it with at most 7 colors. In conclusion, 14 interfaces are sufficient. Concerning I_d, at each pair of connected nodes $d_q \in D$ and f_q, interface I_q can be substituted by any interface among the 14 used by the previous coloring which has not been already assigned to d_q. Other two interfaces I_b and I_c completes the set. Hence, we require a total of 16 interfaces. □

Theorem 2. *In the unit cost case with $k \leq 3$, MMCC is optimally solvable in $O(m)$ time.*

Proof. The proof is based on the analysis of Algorithm 1. The case $k = 1$ is trivial and it is solved by code lines 2–3 of the algorithm. When $k = 2$, either

Algorithm 1.

```
1.  $V' := \emptyset$
2.  **if** $\exists\ i \in \bigcap_{v \in V} W(v)$ **then**
3.    $W_A(v) := \{i\}$ for each $v \in V$
4.  **else**
5.    **for each** $v \in V$ s.t. $|W(v)| \leq 2$
6.      $V' := V' \bigcup \{v\}$
7.      $W_A(v) := W(v)$
8.    **for each** $v \in V$ s.t. $|W(v)| = 3$
9.      **if** $\exists$ a set $S \subset W(v)$, $|S| \leq 2$, s.t.
          $\forall\ u \in N(v) \cap V'$, $S \cap W_A(u) \neq \emptyset$, **then**
10.       $W_A(v) := S$
11.       **if** $|S| = 1$ **then** activate one further arbitrary interface at $v$
12.     **else**
13.       **if** $\forall\ u \in N(v)$, $|W(u)| = 3$ **then**
14.         activate two arbitrary interfaces at $v$
15.       **else**
16.         $W_A(v) := W(v)$
```

there exists one common interface for all the nodes (again code lines 2–3 of Algorithm 1), or the optimal solution costs 2 which equals to activate all the available interfaces at all the nodes (code lines 5–7). Note that in this case code lines 8–16 are not executed as no node holds more than 2 interfaces. When $k = 3$, if there exists a solution of cost 1 (code lines 2–3), again it is easily verifiable by checking whether all the nodes hold one same interface. If not, in order to check whether there exists a solution of cost 2, it is possible to activate all the interfaces at the nodes holding less than 3 interfaces. This can be realized as at code lines 8–16. For each node v holding 3 interfaces, it is possible to check whether at most 2 interfaces among the available 3 are enough to connect v to all its neighbors holding less than 3 interfaces. If not, then the optimal solution costs 3 and all the nodes can activate all their interfaces to accomplish the coverage task (code lines 15–16). If yes, then v activates the 2 interfaces induces by its neighborhood (code lines 9–10); if only 1 or 0 interfaces are induced by the neighborhood then v activates one further (code line 11) or two interfaces (code lines 13–14), respectively, chosen arbitrarily. In this way, all the edges connecting nodes holding at most 2 interfaces and all the edges connecting nodes holding 3 interfaces with nodes holding at most 2 interfaces are covered. In order to conclude the proof, we need to show that all the edges between nodes holding 3 interfaces are covered by the designed activation function. Indeed, since each node holding 3 interfaces activates 2 interfaces, every two of such neighbors must share at least one common interface, and the claim holds. The above algorithm requires $O(m)$ time, as the execution of code lines 9–11 might refer to all the edges of the input graph. □

In [8], it has been shown that when the input graph is a tree and $k = O(1)$ or $\Delta = O(1)$, then Connectivity is polynomially solvable in $O(n)$ or $O(k^{2\Delta}n)$, respectively. Hence, by Proposition 1, we can state the following theorem.

Theorem 3. *If the input graph is a tree and $k = O(1)$ or $\Delta = O(1)$, MMCC can be optimally solved in $O(n)$ or $O(k^{2\Delta}n)$ time, respectively.*

Theorem 3 implies that if the input graph is a path, *MMCC* can be optimally solved in $O(k^4 n)$ time. The case of cycles requires some more insights.

Theorem 4. *If the input graph G is a cycle, MMCC can be optimally solved in $O(k^6 n)$ time.*

Proof. Let OPT be the optimal solution over G, and x be a generic node. In OPT, x makes use of interface i to establish the connection with one of its neighbors, say y, and interface j for the other neighbor z. Possibly, $i \equiv j$. If we consider the path obtained by removing $\{x, y\}$ from the G and by adding a new neighbor x' to y, with x holding only interface j and x' holding only interface i, then the solution OPT is also an optimal solution with respect to the obtained path. In fact, if there exists a better solution OPT' for the obtained path, it must activate the only available interfaces i and j at nodes x' and x to communicate with y and z, respectively, and then it saves something with respect to OPT on the other connections. This would imply that by activating at node x interfaces i and j in the original cycle and by following the solution provided by OPT' for the other nodes, we should obtain a better solution for G with respect to OPT, despite its optimality.

The aforementioned property suggests a way to compute an optimal solution for cycles by means of an algorithm for paths. In order to find the optimal solution for G, we may consider all the path instances obtainable as previously described by associating to x and x' only one interface, possibly the same one, among the original set of interfaces associated with x in G. Such paths are at most $\binom{k}{2} + k = O(k^2)$, and we choose the solution which minimizes the cost in the original cycle G. Hence, by applying the algorithm from Theorem 3 for the case of $\Delta = 2$ for all the obtained path instances, we can find the optimal solution in $O(k^2 \cdot k^4 n)$. □

4 Approximation Results

In this section, we study the approximability properties of *MMCC*. We first show that the problem is not approximable within $\Omega(\ln(\Delta))$, and then we devise a polynomial time algorithm which guarantees an approximation factor of $O((1 + b)\ln(\Delta))$ with b being a parameter depending on structural properties of the input graph. We remind the reader that such a parameter can be bounded by a constant in many graph classes.

In [8], it has been shown that the Connectivity problem is not approximable within $\eta \ln(\Delta)$ for a certain constant η, by an approximation factor preserving reduction from Set Cover (SC). As such a reduction is based on a star topology, from Proposition 1 the following theorem holds.

Theorem 5. *Unless $P = NP$, MMCC in the unit cost unbounded case cannot be approximated within an $\eta \ln(\Delta)$ factor for a certain constant η, even when the input graph is a tree.*

In order to devise an approximation algorithm, we provide a characterization of the graph according to the existence of a *b-bounded ownership function* [7,11]. Given a graph $G = (V, E)$, an ownership function $Own : E \rightarrow V$ is a function that assigns each edge $\{u, v\}$ to an *owner* node between u or v. The set of nodes connected to node u by the edges owned by u is denoted as Own', i.e., $Own'(u) = \{v|Own(\{u, v\}) = u\}$. Function Own is said to be b-bounded if the maximum number of edges owned by a node is less than or equal to b, that is $|Own'(u)| \leq b$ for each $u \in V$.

Parameter b can be computed in polynomial time by using structural properties of the graph. For example b is easily bounded by the *maximum degree*, the *treewidth*, and the *arboricity* of G. In [7], the authors provide a linear time algorithm to find a 3-bounded ownership function for *planar graph*. In [11] it has been observed that for a graph with *pagenumber* p, $b \leq p$ and that, as for graphs with *genus* g, $p = O(\sqrt{g})$ [16], then $b = O(1 + \sqrt{g})$. Moreover, for any graph $g \leq m$, and then for general graphs $b = O(\sqrt{m})$. Finally, in [4] it has been observed that for general graphs $b = O\left(\frac{m}{n}\right)$. All these b-bounded functions can be computed in polynomial time. The resulting bounds are summarized in Table 1.

Table 1. Known bounds on ownership functions for some graph classes

General graphs	$b = O(\sqrt{m})$, $b = O\left(\frac{m}{n}\right)$
Planar graphs	$b \leq 3$
Graph with genus g	$b = O(1 + \sqrt{g})$
Graphs with arboricity a	$b \leq a$
Graphs with maximum degree Δ	$b \leq \Delta$
Graphs with pagenumber p	$b \leq p$
Graphs with treewidth t	$b \leq t$

The approximation algorithm is given in Figure 2. It is based on suitable instances of Set Cover. Here we remind the definition of such a problem:

SC: Set Cover	
Input:	A set U with n elements and a collection $S = \{S_1, S_2, \ldots, S_q\}$ of subsets of U.
Solution:	A cover for U, i.e. a subset $S' \subseteq S$ such that every element of U belongs to at least one member of S'.
Goal:	Minimize $|S'|$.

Algorithm 2 activates a coverage of the graph. In fact, for each node u, it covers all the edges $\{u, v\}$, $v \in N(u) \setminus Own'(u)$ at code lines 7–8 by activating the interfaces corresponding to a solution of $I_{SC}(u)$. While edges $\{u, v\}$, $v \in Own'(u)$ are covered during the iteration related to node v as, by definition of accounting function, $u \in N(v) \setminus Own'(v)$.

It is easy to see that Algorithm 2 is polynomial and its computational time is given by algorithms used to compute function Own at code line 1 and to solve

Algorithm 2.

```
Algorithm 2
1.  Compute a $b$-bounded accounting function $Own$ for $G$
2.  for each node $u \in V$
3.    Define an instance $I_{SC}(u)$ of $SC$ as follows
4.      $U := N(u) \setminus Own'(u)$
5.      for each $i \in W(u)$
6.        $S_i := \{v \in N(u) \setminus Own'(u) \mid i \in W(v)\}$
7.    Solve $I_{SC}(u)$ by using the best known approximation algorithm for $SC$ and
      activate at $u$ the corresponding set of interfaces
8.    Activate at each $v \in N(u) \setminus Own'(u)$ the interface of minimal cost
      in $\{i \mid S_i \in S'\} \cap W(v)$
```

$I_{SC}(u)$ at code line 7. The following theorem gives us the approximation bound for Algorithm 2.

Theorem 6. *Let I be an instance of MMCC where the input graph admits a b-bounded ownership function, the solution provided by Algorithm 2 guarantees a $(\ln(\Delta) + 1 + b \cdot \min\{\ln(\Delta) + 1, c_{max}\})$-approximation factor, with $c_{max} = \max_{i \in \{1,\ldots k\}} c(i)$.*

Proof. Let OPT denote the cost of an optimal solution for I, we show that the solution provided by Algorithm 2 has a cost C such that $C \leq (\ln(\Delta) + 1 + b \cdot \min\{\ln(\Delta) + 1, c_{max}\}) \cdot \text{OPT}$. Given a node $u \in V$, let us denote as $\text{OPT}_{SC}(u)$ and $C_{SC}(u)$ the cost of an optimal solution for instance $I_{SC}(u)$ of $SC(u)$ defined at code lines 3–6 and the cost of the solution for $I_{SC}(u)$ computed at code line 7, respectively. Moreover, let $\text{OPT}_{SC} = \max_{u \in V}\{\text{OPT}_{SC}(u)\}$ and $C_{SC} = \max_{u \in V}\{C_{SC}(u)\}$.

Node u will activate a set of interfaces corresponding to the solution computed at code line 7 at the cost of $C_{SC}(u) \leq C_{SC}$ plus $|Own'(u)|$ interfaces for the connection to nodes in $Own'(u)$ activated at code line 8, in the iteration related to such nodes. Note that, the cost of each interface induced by nodes $v \in Own'(u)$ cannot be bigger than both c_{max} and $C_{SC}(v) \leq C_{SC}$. Moreover, as $|Own'(u)| \leq b$ we obtain,

$$C \leq C_{SC} + |Own'(u)| \cdot \min\{C_{SC}, c_{max}\} \leq C_{SC} + b \cdot \min\{C_{SC}, c_{max}\}.$$

Let us denote as $\text{OPT}(u)$ the cost at u induced by an optimal solution. By definition, for any optimal solution $\text{OPT}(u) \leq \text{OPT}$. Moreover, as an optimal solution has to cover all the edges incident to u,

$$\text{OPT}_{SC}(u) \leq \text{OPT}(u).$$

From [13], there exists a $(\ln|U| + 1)$-approximation algorithm for weighted SC that can be applied at code line 7. Therefore, since $|U| \leq \Delta$,

$$C_{SC}(u) \leq (\ln|U| + 1) \cdot \text{OPT}_{SC}(u) \leq (\ln(\Delta) + 1) \cdot \text{OPT}_{SC}(u).$$

As the above inequalities hold for any $u \in V$, it follows that

$$C_{SC} \leq (\ln(\Delta) + 1) \cdot \textsc{opt},$$

and hence,

$$C \leq (\ln(\Delta)+1)\cdot \textsc{opt} + b \cdot \min\{(\ln(\Delta)+1)\cdot \textsc{opt}, c_{max}\}. \qquad \square$$

Note that, the previous theorem is a generalization of the result stated in [8] for trees. In fact, tree topologies induce $b = 1$, hence obtaining a $2(\ln(\Delta) + 1)$-approximation factor.

Finally, it is worth to mention a further approximation factor that can be achieved for the unit cost case.

Theorem 7. *In the unit cost case, MMCC is $\frac{k}{2}$-approximable in $O(n)$ time.*

The theorem is based on the fact that an optimal solution either activates the same interface among all the nodes (if possible) or it must activates at least two interfaces at some node. Hence, a simple algorithm can check whether all the nodes share a common interface or activates all the available interfaces at all the nodes. Such interfaces are at most k by definition.

5 Conclusion

We have considered the Coverage problem in Multi-Interface Networks. The new objective function with respect to previous works in this area considers the minimization of the maximum cost required by the single nodes of the network. We focused on problem hardness and approximation factors in general and more specific settings. In summary, *MMCC* is *NP*-hard for any fixed $\Delta \geq 5$, while it is polynomially solvable for $\Delta \leq 2$. Moreover, it is *NP*-hard for any fixed $k \geq 16$ while it is polynomially solvable for $k \leq 3$. For fixed k, $4 \leq k \leq 15$ and for fixed Δ, $3 \leq \Delta \leq 4$, the complexity of *MMCC* remains open.

Concerning approximation results for *MMCC*, we show that the problem is not approximable within a factor of $\eta \ln(\Delta)$ for a certain constant η, unless $P = NP$. This result holds even in the unit cost case and when the input graph is a tree, but only when k or Δ are unbounded. We then provide an approximation algorithm that guarantees a factor of $\ln(\Delta) + 1 + b \cdot \min\{c_{max}, (\ln(\Delta) + 1)\}$, with $c_{max} = \max_{i \in \{1,\ldots k\}} c(i)$ and b being a parameter depending on structural properties of the input graph. Another approximation algorithm guarantees a $\frac{k}{2}$ factor of approximation.

This paper represents a first step towards distributed approaches as the objective function refers to local parameters rather than global ones. Further investigations on experimental results and modifications to the proposed model are of main interest.

References

1. Athanassopoulos, S., Caragiannis, I., Kaklamanis, C., Papaioannou, E.: Energy-efficient communication in multi-interface wireless networks. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 102–111. Springer, Heidelberg (2009)
2. Bahl, P., Adya, A., Padhye, J., Walman, A.: Reconsidering wireless systems with multiple radios. SIGCOMM Comput. Commun. Rev. 34(5), 39–46 (2004)
3. Barsi, F., Navarra, A., Pinotti, C.M.: Cheapest paths in multi-interface networks. In: Garg, V., Wattenhofer, R., Kothapalli, K. (eds.) ICDCN 2009. LNCS, vol. 5408, pp. 37–42. Springer, Heidelberg (2008)
4. Bruera, F., Cicerone, S., D'Angelo, G., Di Stefano, G., Frigioni, D.: Dynamic multi-level overlay graphs for shortest paths. Mathematics in Computer Science 1(4), 709–736 (2008)
5. Caporuscio, M., Charlet, D., Issarny, V., Navarra, A.: Energetic Performance of Service-oriented Multi-radio Networks: Issues and Perspectives.. In: 6th Int. Workshop on Software and Performance (WOSP), pp. 42–45. ACM Press, New York (2007)
6. Cavalcanti, D., Gossain, H., Agrawal, D.: Connectivity in multi-radio, multi-channel heterogeneous ad hoc networks. In: IEEE 16th Int. Symp. on Personal, Indoor and Mobile Radio Communications (PIMRC), pp. 1322–1326. IEEE, Los Alamitos (2005)
7. Chrobak, M., Eppstein, D.: Planar orientations with low out-degree and compaction of adjacency matrices. Theoretical Computer Science 86(2), 243–266 (1991)
8. D'Angelo, G., Di Stefano, G., Navarra, A.: Minimizing the Maximum Duty for Connectivity in Multi-Interface Networks. In: Zhong, F. (ed.) COCOA 2010, Part II. LNCS, vol. 6509, pp. 254–267. Springer, Heidelberg (2010)
9. Draves, R., Padhye, J., Zill, B.: Routing in multi-radio, multi-hop wireless mesh networks. In: 10th Annual International Conference on Mobile Computing and Networking (MobiCom), pp. 114–128. ACM, New York (2004)
10. Faragó, A., Basagni, S.: The effect of multi-radio nodes on network connectivity—a graph theoretic analysis. In: IEEE Int. Workshop on Wireless Distributed Networks (WDM). IEEE, Los Alamitos (2008)
11. Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Semidynamic algorithms for maintaining single-source shortest path trees. Algorithmica 22(3), 250–274 (1998)
12. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
13. Johnson, D.S.: Approximation algorithms for combinatorial problems. Journal of Computer and Sysntem Sciences 9, 256–278 (1974)
14. Klasing, R., Kosowski, A., Navarra, A.: Cost Minimization in Wireless Networks with a Bounded and Unbounded Number of Interfaces. Networks 53(3), 266–275 (2009)
15. Kosowski, A., Navarra, A., Pinotti, M.C.: Exploiting Multi-Interface Networks: Connectivity and Cheapest Paths. Wireless Networks 16(4), 1063–1073 (2010)
16. Malitz, S.M.: Genus g graphs have pagenumber $O(\sqrt{g})$. Journal of Algorithms 17(1), 85–109 (1994)

Bandwidth Constrained Multi-interface Networks

Gianlorenzo D'Angelo[1], Gabriele Di Stefano[1], and Alfredo Navarra[2]

[1] Dipartimento di Ingegneria Elettrica e dell'Informazione,
Università degli Studi dell'Aquila, Italy
gianlorenzo.dangelo@univaq.it, gabriele.distefano@univaq.it
[2] Dipartimento di Matematica e Informatica, Università degli Studi di Perugia, Italy
navarra@dmi.unipg.it

Abstract. In heterogeneous networks, devices can communicate by means of multiple wired or wireless interfaces. By switching among interfaces or by combining the available interfaces, each device might establish several connections. A connection is established when the devices at its endpoints share at least one active interface. Each interface is assumed to require an activation cost, and provides a communication bandwidth. In this paper, we consider the problem of activating the cheapest set of interfaces among a network $G = (V, E)$ in order to guarantee a minimum bandwidth B of communication between two specified nodes. Nodes V represent the devices, edges E represent the connections that can be established. In practical cases, a bounded number k of different interfaces among all the devices can be considered. Despite this assumption, the problem turns out to be *NP*-hard even for small values of k and Δ, where Δ is the maximum degree of the network. In particular, the problem is *NP*-hard for any fixed $k \geq 2$ and $\Delta \geq 3$, while it is polynomially solvable when $k = 1$, or $\Delta \leq 2$ and $k = O(1)$. Moreover, we show that the problem is not approximable within $\eta \log B$ or $\Omega(\log \log |V|)$ for any fixed $k \geq 3$, $\Delta \geq 3$, and for a certain constant η, unless $P = NP$. We then provide an approximation algorithm with ratio guarantee of b_{max}, where b_{max} is the maximum communication bandwidth allowed among all the available interfaces. Finally, we focus on particular cases by providing complexity results and polynomial algorithms for $\Delta \leq 2$.

1 Introduction

The interest in wireless networks has rapidly grown during the last decades. Their success is certainly due to the wide range of applications for which such networks are designed. A very important issue is constituted by the heterogeneity of the devices which might interact in order to exchange data. Wireless networks are, in fact, composed of devices with different characteristics like computational power, energy consumption, radio interfaces, supported communication protocols, and so forth. In this paper, we are mainly interested in devices equipped with multiple interfaces (like Bluetooth, WiFi, GPRS, etc.). A connection between two or more devices might be accomplished by means of different communication networks according to connectivity and quality of service requirements.

The selection of the most suitable interface for a specific connection might depend on various factors. Such factors include: its availability in specific devices, the required communication bandwidth, the cost (in terms of energy consumption) of maintaining an active interface, the available neighbors, and so forth. While managing such connections, a lot of effort must be devoted to energy consumption issues. Devices are, in fact, usually battery powered and the network survivability might depend on their persistence in the network.

We study communication problems in wireless networks supporting multiple interfaces. In the considered model, the input network is described by a graph $G = (V, E)$, where V represents the set of wireless devices and E is the set of possible connections according to proximity of devices and the available interfaces that they may share. Each $v \in V$ is associated with a set of available interfaces $W(v)$. The set of all the possible interfaces available in the network is then determined by $\bigcup_{v \in V} W(v)$; we denote the cardinality of this set by k. We say that a connection is satisfied (or covered) when the endpoints of the corresponding edge share at least one active interface. If an interface x is activated at some node u, then u consumes some energy $c(x)$ for maintaining x as active, and it provides a maximum communication bandwidth $b(x)$ with all its neighbors which share interface x. In this setting, we study the problem of establishing a communication path between two selected nodes $s, t \in V$ of minimum cost in terms of energy consumption, while guaranteeing a minimum communication bandwidth B. In other words, we look for the minimum cost set of active interfaces among the input graph G, in such a way that s is guaranteed to exchange data with t at least with some bandwidth B. This implies that between s and t not necessarily a path of covered edges must be established but a more complex graph might be required according to the topology and to the available interfaces.

Related work. Multi-interface wireless networks have recently been studied in a variety of contexts, usually focusing on the benefits of multiple radio devices of each node. Many basic problems of standard wireless network optimization can be reconsidered in such a setting [3], in particular, focusing on issues related to routing [7] and network connectivity [5,8]. The study of combinatorial problems on multi-interface wireless networks has originated from [4]. That paper, as well as [13] investigate the so called *Coverage* problem, where the goal is the activation of the minimum cost set of interfaces in such a way that all the edges of G are covered. *Connectivity* issues have been addressed in [2,6,14]. The goal becomes to activate the minimum cost set of interfaces in G in order to guarantee a path of communication between every pair of nodes. In [14], the attention has been devoted to the so called *Cheapest path* problem. This corresponds to the well-known shortest path problem but in the context of multi-interface networks. A natural continuation on investigating such kind of networks is certainly to consider also quality of service constraints in the problem. To the best of our knowledge, bandwidth issues have never been treated before in this context.

Our results. In this paper, we are interested in establishing the cheapest way of communication between two given nodes while guaranteeing a minimum

Table 1. Complexity results achieved in this paper for *BCMI*

Δ	**k**	**Complexity**
$\Delta = 1$	Bounded	Optimally solvable in $O(1)$ time
	Unbounded	NP-hard (equiv. MinKnapsack), $(1+\epsilon)$-apx in $O(\frac{k^2}{\epsilon})$
$\Delta = 2$	Bounded	Optimally solvable in $O(\|V\|)$
	Unbounded	NP-hard; $(2+\epsilon)$-apx in $O(\|V\|\frac{k^2}{\epsilon})$ for paths
Fixed $\Delta \geq 3$	Fixed $k \geq 2$	NP-hard (from *X3C*)
	Fixed $k \geq 3$	Not apx within $\eta \log B$, or within $\Omega(\log\log \|V\|)$
Any	$k = 1$	Opt. solvable in $O(\|V\| + \|E\|)$ (equiv. Shortest Path)
	Any	$b_{\max}$-apx (optimal for constant bandwidth)

bandwidth of communication. The resulting problem, called *Bandwidth Constraints in Multi-Interface Networks* (*BCMI*) is similar to the better known *Minimum Edge Cost Flow* [9]. The main difference resides in the fact that we do not consider costs and capacities for the edges of the network but we have to cope with interfaces at the nodes that require some costs and can manage some maximum bandwidths. In the special case where each connection can be established by means of a different interface, the two problems coincide. Hence, it is not surprising that *BCMI* turns out to be *NP*-hard when the number k of interfaces is unbounded. However, in practical cases it is more realistic to consider a bounded number of interfaces. Despite the expectations, we show that the problem is *NP*-hard even when k is a fixed small number. In detail, we prove that the problem is *NP*-hard for any fixed $k \geq 2$ and $\Delta \geq 3$, where Δ is the maximum degree of the network, while it is polynomially solvable when $k = 1$, or $\Delta \leq 2$ and $k = O(1)$. Moreover, we show that the problem is not approximable within $\eta \log B$ or $\Omega(\log\log |V|)$ for any fixed $k \geq 3$, $\Delta \geq 3$, and for a certain constant η, unless $P = NP$. We then provide an approximation algorithm with ratio guarantee of b_{max}, where b_{max} is the maximum communication bandwidth allowed among all the available interfaces. This algorithm optimally solves the problem in the case that the bandwidth is constant for all the interfaces. Finally, we focus on particular cases by providing complexity results and polynomial algorithms for $\Delta \leq 2$. Surprisingly, when k is unbounded and the network reduces to a single edge the problem remains *NP*-hard. Table 1 summarizes the results.

2 Definitions and Notation

For a graph G, we denote by V its node set, by E its edge set, and by Δ its maximum node degree. Unless otherwise stated, the graph $G = (V, E)$ representing the network is assumed to be undirected, connected, and without multiple edges and loops. A global assignment of the interfaces to the nodes in V is given in terms of an appropriate interface assignment function W, as follows.

Definition 1. *A function* $W\colon V \to 2^{\{1,2,\ldots,k\}}$ *is said to* cover *graph* G *if for each* $\{u,v\} \in E$ *we have* $W(u) \cap W(v) \neq \emptyset$.

The cost of activating an interface i is given by the cost function $c\colon \{1,2,\ldots,k\} \to \mathbb{Z}_0^+$ and it is denoted as $c(i)$. The bandwidth allowed by a given interface i is defined by the bandwidth function $b\colon \{1,2,\ldots,k\} \to \mathbb{Z}_0^+$ and it is denoted as $b(i)$. It follows that each node holding an interface i pays the same cost $c(i)$ and provides the same bandwidth $b(i)$ by activating i. The considered *BCMI* optimization problem is formulated as follows.

BCMI: Bandwidth Constraints in Multi-Interface Networks

***Input*:** A graph $G = (V,E)$, a source node $s \in V$, a target node $t \in V$, a set of interfaces $I = \{1,2,\ldots,k\}$, an allocation of available interfaces $W\colon V \to 2^I$ covering graph G, an interface cost function $c\colon I \to \mathbb{Z}_0^+$, an interface bandwidth function $b\colon I \to \mathbb{Z}_0^+$ and a bound $B \in \mathbb{Z}_0^+$.

***Solution*:** An allocation of active interfaces $W_A\colon V \to 2^I$, $W_A(v) \subseteq W(v)$ for all $v \in V$ and a flow function $f : V \times V \times I \to \mathbb{Z}_0^+$ such that:

- $f(u,v,i) = 0$ if $\{u,v\} \notin E$ or $W_A(u) \cap W_A(v) = \emptyset$ for all $u,v \in V$ and $i \in I$
- $\sum_{v\in V: f(u,v,i)>0} f(u,v,i) \leq b(i)$ for all $u \in V$ and $i \in I$
- $f(u,v,i) = -f(v,u,i)$ for all $u,v \in V$ and $i \in I$
- $\sum_{v\in V, i\in I} f(u,v,i) = 0$ for all $u \in V \setminus \{s,t\}$
- $\sum_{v\in V, i\in I} f(s,v,i) = \sum_{v\in V, i\in I} f(v,t,i) \geq B$

***Goal*:** Minimize the total cost of the active interfaces, $c(W_A) = \sum_{v\in V} \sum_{i\in W_A(v)} c(i)$.

Note that we can consider two variants of the above problem: the parameter k can be considered as part of the input (this is called the *unbounded case*), or k may be a fixed constant (the *bounded case*). In both cases we assume $k \geq 2$, since the case $k = 1$ admits an obvious unique solution given by the shortest path connecting s to t of maximum bandwidth $b(1)$. The case where the cost function is constant for each interface is called the *unit cost* case.

3 Hardness and Approximation

In this section we first prove that *BCMI* is *NP*-hard even in the restricted case of unit cost, fixed $k \geq 2$, and fixed $\Delta \geq 3$. We then prove that, unless $P = NP$, the problem is inapproximable within a factor of $\eta \log B$, for a certain constant η, or within a factor of $\Omega(\log\log |V|)$. Finally, we provide a polynomial time $b_{\max}$-approximation algorithm, where $b_{\max} = \max_{i\in I} b(i)$.

Theorem 1. *BCMI is NP-hard even when restricted to the unit cost interface case for any fixed* $\Delta \geq 3$ *and* $k \geq 2$.

Proof. We prove that the underlying decisional problem, denoted by $BCMI_D$, is in general *NP*-complete. We need to add one further bound $B' \in \mathbb{Z}_0^+$ such that the problem will be to ask whether there exists an activation function which induces a total cost of the active interfaces of at most B'.

Given an allocation function of active interfaces for an instance of $BCMI_D$, to check whether the induced subgraph allows a flow bandwidth greater than or equal to B of total cost smaller than or equal to B' is linear in the number of edges of the input graph G. The proof then proceeds by a polynomial reduction from the well-known *Exact Cover by* 3-*Sets* problem. The problem is known to be *NP*-complete [9] and it can be stated as follows:

X3C: Exact Cover by 3-Sets	
Input:	Set X with $\|X\| = 3q$ and a collection C of 3-element subsets of X.
Question:	Is there an exact set cover for X, i.e. a subset $C' \subseteq C$ such that $\|C'\| = q$ and every element of X belongs to exactly one member of C'?

Given an instance of *X3C*, we construct an instance of $BCMI_D$ where the graph G consists of copies of subgraphs $N(\ell)$ and $T(\ell)$, $\ell \geq 1$ (see Fig. 1). Subgraph $N(\ell)$ consists of 3ℓ nodes $\{x_1, x_2, \ldots, x_\ell\} \cup \{y_1, y_2, \ldots, y_\ell\} \cup \{w_1, w_2, \ldots, w_\ell\}$ and edges $\{x_i, x_{i+1}\}$, $\{w_i, w_{i+1}\}$, for $i = 1, 2, \ldots, \ell - 1$ and $\{x_i, y_i\}$, $\{y_i, w_i\}$, for $i = 1, 2, \ldots, \ell$. Subgraph $T(\ell)$ is a binary tree consisting of a complete binary tree BT with $2^{\lceil \log_2 \ell \rceil} - 1$ nodes, and ℓ nodes adjacent to the leaves of BT. These nodes are the only leaves of $T(\ell)$, i.e. every leaf of BT is connected to at least one leaf of $T(\ell)$. We call r the root of $T(\ell)$. Note that, each path from r to a leaf of $T(\ell)$ is constituted of $\lceil \log_2 \ell \rceil + 1$ nodes. Moreover, when $\ell = 1$, BT is empty and $T(\ell)$ consists of a single node.

We now define the graph G, see Fig. 1 right. Let s and t be two nodes of G. For each element C_i of C, $i = 1, 2, \ldots, |C|$, G contains a node c^i, a copy of $N(3)$, denoted as $N^i(3)$ and a copy of $T(3)$, denoted as $T^i(3)$, with root r^i and leaves l_1^i, l_2^i, l_3^i. Vertices x_1^i and w_3^i of $N^i(3)$ are adjacent to c^i and r^i, respectively. All nodes c^i form a path P in G, that is $\{c^i, c^{i+1}\}$ is an edge of G, for $i = 1, 2, \ldots, |C| - 1$. Node s of G is adjacent to c^1, while node $c^{|C|}$ is adjacent to node x_1^0 belonging to a copy $N^0(1)$ of $N(1)$ with nodes x_1^0, y_1^0 and w_1^0.

Let e_j, $j = 1, 2, \ldots, 3q$, be the elements of X and let $\mu(e_j)$ be the number of sets $C_i \in C$ containing e_j, for each j. Let $\mu = \max_j\{\mu(e_j)\}$. For each element e_j, G contains a copy of $T(\mu)$, called $T^j(\mu)$, with root r^j, and a copy $N^j(1)$ of $N(1)$, with nodes x_1^j, y_1^j and w_1^j. Root r^j is adjacent to $x_1^j \in N^j(1)$, for each $j = 1, 2, \ldots, 3q$. If e_j is in C_i, for some i and j, then there is an edge from a leaf of $T^i(3)$ to a leaf of $T^j(\mu)$. These edges are pairwise disjoint. Note that, even if each leaf of $T^i(3)$, $i = 1, 2, \ldots, |C|$ is adjacent to a leaf in $T^j(\mu)$, for some $j \in \{1, 2, \ldots, 3q\}$, the contrary is not true: there could be a leaf of $T^j(\mu)$, for some j, not adjacent to any leaf of $T^i(3)$, $i = 1, 2, \ldots, |C|$.

G also contains a copy of $T(3q + 1)$, having the root adjacent to node t, and leaves adjacent to nodes w_1^j, $j = 0, 1, \ldots, 3q$. The set of interfaces I is $\{1, 2\}$, with $c(1) = c(2) = 1$ and $b(1) = 1$, $b(2) = 3q + 1$. All the nodes in G have interface 2

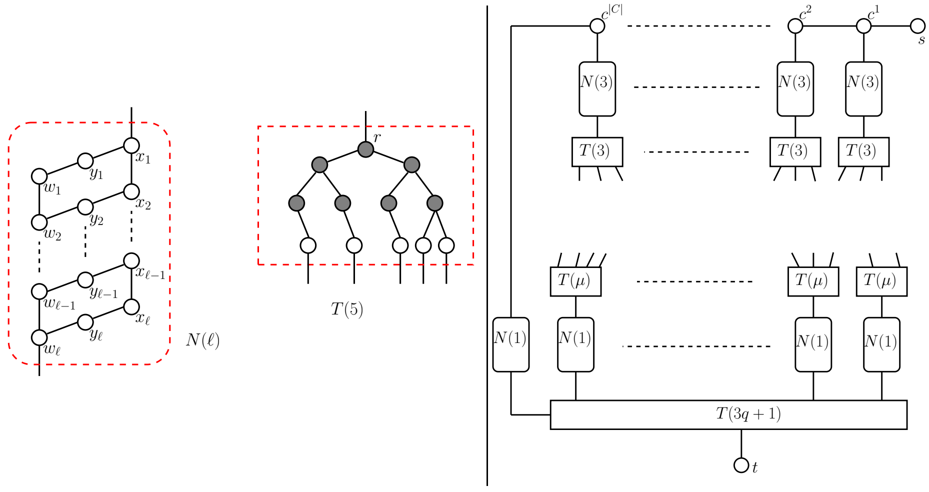


Fig. 1. Left: The subgraphs used in the proofs of Theorems 1 and 2. Right: The graph G in the transformation from *X3C* to $BCMI_D$.

apart from nodes labeled y in the copies of $N(1)$ and $N(3)$. All the nodes in the copies of $N(1)$ and $N(3)$ have interface 1: no further node in G has interface 1. When all the interfaces of the nodes in copies of $N(\ell)$ ($T(\ell)$, resp.), for a certain $\ell \geq 0$, are active the total cost is 5ℓ ($2^{\lceil \log_2 \ell \rceil} - 1 + \ell$, resp.). In $T(\ell)$, when only the interfaces of the nodes in a single path from r to a leaf are active, the total cost is $\lceil \log_2 \ell \rceil + 1$. Let $B = 3q+1$ and $B' = |C| + q(42 + 3\lceil \log_2 \mu \rceil) + 2^{\lceil \log_2 (3q+1) \rceil} + 7$.

Assume that *X3C* has a positive answer, i.e., there exists an exact set cover $C' = \{C_{i_1}, C_{i_2}, \ldots, C_{i_q}\} \subseteq C$ for X. We show that also $BCMI_D$ has a positive answer, i.e., there exists an activation function W_A of the available interfaces such that the bandwidth allowed from s to t is bigger than or equal to B and the total cost is smaller than or equal to B'. Function W_A is defined as follows. Along with interfaces of nodes s, t, all the interfaces of nodes in $T(3q+1)$, $N^j(1)$, $j = 0, 1, \ldots, 3q$, and c^i, $i = 1, 2, \ldots, |C|$, are active. All the interfaces of nodes in $N^{i_j}(3)$ and $T^{i_j}(3)$, for each $C_{i_j} \in C'$, $j = 1, 2, \ldots, q$, are active. Moreover, if $e_j \in X$ is covered by $C_i \in C'$, then all the interfaces of nodes in $T^j(\mu)$ belonging to the path from r^j to a leaf in $T^i(3)$ are active. No further interface is active. The flow function is defined as 1 in nodes y of active copies of $N(1)$ and $N(3)$ and in the remainder of G it is defined to satisfy the flow conservation constraints.

The total cost of active interfaces is given by 2, for nodes s and t; $|C|$, for nodes $c^i \in P$, $i = 1, 2, \ldots, |C|$; $15q + 6q$ for nodes in $N^{i_j}(3)$ and $T^{i_j}(3)$, $j = 1, 2, \ldots, q$; $3q(\lceil \log_2 \mu \rceil + 1)$ for nodes in $T^j(\mu)$, $j = 1, 2, \ldots 3q$; $5(3q+1)$ for nodes in $N^j(1)$, $j = 0, 1, \ldots 3q$; and $2^{\lceil \log_2 (3q+1) \rceil} + 3q$ for nodes in $T(3q+1)$. Summing up all the values we obtain a cost equal to B'.

Regarding the total bandwidth, note that a copy of $N(\ell)$ has a maximum bandwidth of ℓ. As *X3C* has a positive answer, each element of X is covered, then the flow through each subgraph $N^j(1)$, $j = 1, 2, \ldots, 3q$, is exactly 1. As all the interfaces in P are active, we also have another unit of flow from $N^0(1)$ that reaches t through the $T(3q+1)$ subgraph, hence obtaining a total flow of $3q+1$, i.e., $BCMI_D$ has a positive answer.

Now, let us assume we have a positive answer to $BCMI_D$. As the total flow received by t is greater than or equal to $B = 3q+1$, there is a flow of value 1 in each subgraph $N^j(1)$, $j = 0, 1, \ldots, 3q$, meaning that each element of X is covered. Let us suppose, by contradiction, that the flow reaching the $N^j(1)$, $j = 1, 2, \ldots, 3q$ subgraphs, implies the activation of the interfaces in $q' > q$ subgraphs among the $N^i(3)$, $i = 1, 2, \ldots, |C|$ copies of $N(3)$. In this case there will be q'_1 subgraphs having one unit of flow, q'_2 subgraphs having 2 units of flow, and q'_3 subgraphs having 3 units of flow such that $q'_1 + 2q'_2 + 3q'_3 = 3q$.

The total cost for the interfaces activation is: 2, for nodes s and t; $|C|$, for nodes in P (all the interfaces in P are active as $N^0(1)$ receives one unit of flow); $7q'_1 + 11q'_2 + 15q'_3$ for nodes in $N^i(3)$; $6q$ for nodes in $T^i(3)$, $i = 1, 2, \ldots, q$; $3q(\lceil \log_2 \mu \rceil + 1)$ for nodes in $T^j(\mu)$, $j = 1, 2, \ldots 3q$; $5(3q+1)$ for nodes in $N^j(1)$, $j = 0, 1, \ldots 3q$, and $2^{\lceil \log_2(3q+1) \rceil} + 3q$ for nodes in $T(3q+1)$.

Then the total cost is $|C| + q(27q + 3\lceil \log_2 \mu \rceil) + 2^{\lceil \log_2(3q+1) \rceil} + 7 + 7q'_1 + 11q'_2 + 15q'_3$. As $7q'_1 + 11q'_2 + 15q'_3 > 5(q'_1 + 2q'_2 + 3q'_3) = 15q$, the total cost is greater than B', a contradiction. Hence there are exactly q subgraphs $N^{i_j}(3)$, $j = 1, 2, \ldots, q$ with 3 units of flow each and the corresponding sets C_{i_j}, $j = 1, 2, \ldots, q$, represent a solution for *X3C*. □

Theorem 2. *BCMI cannot be approximated within a factor of $\eta \log B$, for a certain constant η, or within a factor of $\Omega(\log \log |V|)$, for any fixed $\Delta \geq 3$ and $k \geq 3$, unless P = NP.*

Theorem 2 also holds when the number of interfaces is unbounded. We now provide a $b_{\max}$-approximation algorithm for any instance of *BCMI*, where $b_{\max}$ is the maximum bandwidth value among the interfaces in I. The algorithm consists in relaxing *BCMI* to the well-known *Integral Minimum Cost Flow* (*IMCF*) problem [1]. In the proof of the next theorem, we transform an instance of *BCMI* into an instance of *IMCF*, and we show that such a transformation guarantees an approximation factor of $b_{\max}$. Let $\mathcal{A}$ be an algorithm which optimally solves *IMCF* in a graph $H = (V', E')$ in polynomial time $P_{\mathcal{A}}(|V'| + |E'|)$.

Theorem 3. *There exists a polynomial time $b_{\max}$-approximation algorithm for BCMI which requires $O(|V|k^2 + |E| + P_{\mathcal{A}}(|V|k^2 + |E|))$ time.*

Proof. First, we transform an instance I_1 on a graph $G = (V, E)$ of *BCMI* in an instance of an equivalent problem defined on a directed graph $G' = (V', A)$ without using multiple interfaces but associating costs and bandwidth only to arcs in A. The particular instance I_2 of such problem is defined as follows. Informally, for each interface of each node, there is an arc which has the same cost and bandwidth of the considered interface. The head of each of such arcs is connected to the tail of another arc of the same kind if they share an interface or they represent different interfaces of the same node. Formally, there are two nodes in V' for each node in V and for each interface of each node:

$V' = \{(\overline{v}, i), (\underline{v}, i) \mid v \in V, i \in W(v)\} \cup \{\tilde{s}, \tilde{t}\}$,

$A = \{((\overline{v},i),(\underline{v},i)) \mid v \in V, i \in W(v)\} \cup \{((\underline{v},i),(\overline{v},j)) \mid v \in V\,,\ i,j \in W(v)$ s.t. $i \neq j\} \cup \{((\underline{v},i),(\overline{u},i)) \mid \{u,v\} \in E, i \in W(v) \cap W(u)\} \cup \{(\tilde{s},(\overline{s},i)),((\underline{t},j),\tilde{t})$ $\mid i \in W(s), j \in W(t)\}$.

The capacity of each arc $a = ((\overline{v},i),(\underline{v},i))$ is set to $b'(a) = b(i)$ whereas the capacity of each other arc is unlimited. The cost $c'(a)$ of each arc $((\overline{v},i),(\underline{v},i))$ is set to $c(i)$ and it is 0 for the remaining arcs. The objective is to find a flow function which minimizes the overall cost of arcs with positive flow and guarantees a flow of B between $\tilde{s}$ and $\tilde{t}$.

Given a solution for I_2, which defines a flow function f_2, we can define a solution for I_1 by assigning a flow function $f_1(v,u,i) = f_2((\underline{v},i),(\overline{u},i)) - f_2((\underline{u},i),(\overline{v},i))$, for each $v,u \in V$ and $i \in W(v) \cap W(u)$. Vice versa, given a solution for I_1, which defines a flow function f_1', we can define a solution for I_2 by assigning a flow function f_2' such that $f_2'((\underline{v},i),(\overline{u},i)) = f_1'(v,u,i)$, if $f_1'(v,u,i) > 0$ and $f_2'((\underline{v},i),(\overline{u},i)) = 0$ otherwise, for each $v,u \in V$ and $i \in W(v) \cap W(u)$. The flows in the remainder of A are set in order to satisfy flow conservation constraints. It is not difficult to note that the feasibility of f_2 (f_1', resp.) implies the feasibility of f_1 (f_2'). Moreover, the cost of f_2 (f_1', resp.) is equal to the cost of f_1 (f_2') as the cost of arcs $((\overline{v},i),(\underline{v},i))$ in A is $c(i)$ and it is 0 for any other arc. By the above discussion it follows that we can solve I_1 by solving I_2.

We find an approximate solution for I_2 by using an *IMCF* instance. The *IMCF* problem consists of finding an integral flow greater than or equal to a given quantity between two nodes in a directed graph H where each arc a has a capacity $\beta(a)$ and cost $\chi(a)$. The objective is to minimize the function $\sum_{a \in A^+} \chi(a) \cdot f(a)$, where $f(a)$ is the flow on arc a and A^+ is the set of arcs with positive flow. This problem admits a polynomial time algorithm (see, e.g., [15]).

We obtain an *IMCF* instance I_3 from I_2 by setting $H = G'$, $\beta(a) = b'(a)$, and $\chi(a) = c'(a)/b'(a)$, for each $a \in A$.

Let us denote as f^* and f^{IMCF} two optimal flow functions for I_2 and I_3, respectively and as A^* and A^{IMCF} the corresponding sets of arcs with positive flow. By definition, $\text{OPT} = \sum_{a \in A^*} c'(a)$. As $f^*(a) \leq b'(a)$, it follows that

$$\sum_{a \in A^*} c'(a) \geq \sum_{a \in A^*} c'(a) \cdot \frac{f^*(a)}{b'(a)} = \sum_{a \in A^*} \chi(a) \cdot f^*(a).$$

By the optimality of A^{IMCF} it follows that

$$\sum_{a \in A^*} \chi(a) \cdot f^*(a) \geq \sum_{a \in A^{IMCF}} \chi(a) \cdot f^{IMCF}(a) = \sum_{a \in A^{IMCF}} \frac{c'(a)}{b'(a)} \cdot f^{IMCF}(a).$$

As $f^{IMCF}(a) \in \mathbb{Z}_0^+$, for each $a \in A$, then $f^{IMCF}(a) \geq 1$, for each $a \in A^{IMCF}$. Moreover, $b_{\max} \geq b'(a)$, for each $a \in A^{IMCF}$.

Therefore, $\sum_{a \in A^{IMCF}} \frac{c'(a)}{b'(a)} \cdot f^{IMCF}(a) \geq \frac{1}{b_{\max}} \sum_{a \in A^{IMCF}} c'(a)$. □

Corollary 1. *Let $b \in \mathbb{Z}_0^+$. If $b(i) = b$ for each $i \in I$, BCMI is solvable within $O(|V|k^2 + |E| + P_{\mathcal{A}}(|V|k^2 + |E|))$.*

Proof. If $b = 1$, then the $b_{\max}$-approximation algorithm given in Theorem 3 optimally solves *BCMI*. Otherwise, it is enough to solve the problem with required bandwidth of $\bar{B} = \lceil \frac{B}{b} \rceil$ and bandwidth $\bar{b}(i) = 1$, for each interface i. □

4 Particular Cases, $\Delta \leq 2$

In this section, we consider graphs of bounded degree $\Delta \leq 2$. As announced in Table 1, we now prove that when the number of interfaces k is a given constant, the problem can be optimally solved in polynomial time. On the other hand, if k is unbounded, we show that the problem remains *NP*-hard.

For $\Delta \leq 1$, the input graph can be composed of either one single node or two nodes connected by one edge. In the first case, there are no interfaces to be activated, as the source and the destination coincide. In the second case, the problem already starts to be interesting.

Lemma 1. *BCMI is polynomially solvable within $O(1)$ time in the bounded case with $\Delta = 1$.*

Proof. *BCMI* can be solved by an exhaustive search among all the possible combinations of interfaces shared by s and t. The number of such combinations is $O(2^k)$. Among them, a resolution algorithm has to choose the cheapest one that guarantees at least B bandwidth. □

For the unbounded case, i.e., when k is not a given constant, the same arguments of Lemma 1 do not apply to *BCMI* as the provided algorithm would show an exponential behavior. Surprisingly, in this setting the problem turns out to be already *NP*-hard by means of a simple polynomial transformation from the well known Knapsack problem. Indeed, we need to consider the so called Minimization Knapsack problem [11,12].

MinKP: Minimization Knapsack	
Input:	An integer $d \in \mathbb{Z}_0^+$ and a set of n items, each one having weight $w_i \in \mathbb{Z}_0^+$ and profit $p_i \in \mathbb{Z}_0^+$, $i = 1, 2, \ldots, n$.
Solution:	An allocation of variables $y_i \in \{0, 1\}$, for $i = 1, 2, \ldots, n$, such that $\sum_{i=1}^n w_i y_i \geq d$
Goal:	Minimize $\sum_{i=1}^n p_i y_i$.

MinKP problem is the corresponding minimization version of the Knapsack problem. In other words, the goal is to minimize the profits of the items that remain out of the knapsack. If x_i, $i = 1, 2, \ldots, n$, are the variables selecting the items for the classical knapsack problem and $c \in \mathbb{Z}_0^+$ its capacity, then the problem can be solved by means of *MinKP*, by setting $d = \sum_{i=1}^n w_i - c$ and $y_i = 1 - x_i$, $i = 1, 2, \ldots, n$.

When $\Delta = 1$, that is when the input graph G consists of a single edge from s to t, the required solution must select a subset of interfaces among the ones shared by s and t in such a way that a bandwidth of B is guaranteed, and the cost for activating such interfaces is minimized. Intuitively, this particular case of *BCMI* is equivalent to the *MinKP* problem.

Theorem 4. *BCMI is polynomially equivalent to MinKP in the unbounded case with $\Delta = 1$.*

Proof. We have to show that there exist two polynomial time algorithms $\mathcal{A}$ and $\mathcal{B}$ such that, for each instance I_1 of *MinKP*, $\mathcal{A}(I_1)$ returns an instance I_2 of *BCMI*, for any solution σ' of I_2, $\mathcal{B}(\sigma') = \sigma$ is a solution for I_1, and the values of solutions σ and σ' are equal. Moreover, we have to show that there exist two polynomial time algorithms $\mathcal{A}^{-1}$ and $\mathcal{B}^{-1}$ such that, for each instance I_2 of *BCMI*, $\mathcal{A}^{-1}(I_2)$ returns an instance I_1 of *MinKP*, for any solution σ of I_1, $\mathcal{B}^{-1}(\sigma) = \sigma'$ is a solution for I_2, and the values of solutions σ and σ' are equal.

We now show the first part of the above statement by defining the polynomial algorithms $\mathcal{A}$ and $\mathcal{B}$. Given an instance I_1 of *MinKP*, we consider an instance I_2 of *BCMI* made of nodes s and t, edge $\{s,t\}$ and, for each item i of I_1, an interface shared between s and t with cost $c(i) = \frac{1}{2}p_i$ and bandwidth $b(i) = w_i$. Moreover, let $k = n$ and $B = d$. Note that, if, for some i, p_i is an odd number, we can scale all the profits p_i of a factor 2 in order to have $c(i) \in \mathbb{Z}_0^+$ for each $i = 1, 2, \ldots, n$. This does not affect the generality of the proof as it is enough to divide by 2 the objective function value of the solution for I_1 which will be defined in the following. A feasible solution for I_2 selects a set of interfaces W, by means of an activation function, in such a way that $B \leq \sum_{i \in W} b(i)$. As $d = B \leq \sum_{i \in W} b(i) = \sum_{i \in W} w_i$ and the cost of activating interfaces W in both s and t is $2\sum_{i \in W} c(i) = \sum_{i \in W} p_i$ we can define algorithm $\mathcal{B}$ as the algorithm which selects items W in order to output a solution for I_1. Finally, both $\mathcal{A}$ and $\mathcal{B}$ are polynomial time algorithms. This proves the first part of the theorem. For the second part of the theorem, it is enough to note that algorithms $\mathcal{A}$ and $\mathcal{B}$ can be naturally inverted. □

Corollary 2. *BCMI is NP-hard in the unbounded case with $\Delta = 1$.*

Corollary 3. *In the unbounded case with $\Delta = 1$, BCMI admits a $(1+\epsilon)$-approximation algorithm which requires $O(\frac{k^2}{\epsilon})$ time, for any $\epsilon > 0$.*

Proof. It follows by applying the linear time algorithm $\mathcal{A}$ of Theorem 4 which requires $O(k)$ time, and the algorithm from [10] which provides a $(1+\epsilon)$-approximation for *MinKP* in $O(\frac{k^2}{\epsilon})$ time. □

For $\Delta = 2$, the input graph of *BCMI* is either a path or a cycle. Clearly, from Corollary 2, *BCMI* remains *NP*-hard in the unbounded case. The following theorems give polynomial time algorithms for the bounded case, and a refined approximation algorithm for paths in the unbounded case.

In the remainder, for a set of interfaces W, we denote as $c(W)$ the cost of activating the interfaces in W, formally: $c(W) = \sum_{i \in W} c(i)$.

Theorem 5. *BCMI is solvable within $O(|V|)$ time in the bounded case when the input graph is a path.*

Theorem 6. *In the unbounded case, if the input graph is a path, BCMI admits a $(2+\epsilon)$-approximation algorithm which requires $O(|V|\frac{k^2}{\epsilon})$ time, for any $\epsilon > 0$.*

Proof. Let us denote the input path as a sequence of n nodes: $s \equiv x_0, x_1, \ldots,$ $x_{n-1} \equiv t$. We define an algorithm $\mathcal{C}$ as follows. It defines $n-1$ *MinKP* problems, each one arising from one different edge $e_i = \{x_{i-1}, x_i\}$ of the path, $1 \leq i \leq n-1$, by using the linear time algorithm $\mathcal{A}$ of Theorem 4. From Corollary 3, this implies that for each e_i and for any $\epsilon > 0$, a $(1+\epsilon)$-approximation for *MinKP* can be guaranteed. Algorithm $\mathcal{C}$ chooses, for each $1 \leq i \leq n-1$, interfaces W_i arising from the approximate solution of the related knapsack problem on edge e_i, that is interfaces W_i are activated on nodes x_{i-1} and x_i.

For each $1 \leq i \leq n-1$, let us denote as W_i^*, the sets of active interfaces in nodes x_{i-1} and x_i covering edge e_i for an optimal solution of *BCMI* for the input path; and let W_i^{MK} the sets of active interfaces in nodes x_{i-1} and x_i covering edge e_i for an optimal solution of the *MinKP* problem obtained by $\mathcal{C}$ for the input path.

Note that, for some i, the set $W_i \cap W_{i+1}$ is not necessarily empty, which means that node x_i uses a set of interfaces for communicating both with x_{i-1} and x_{i+1}. Thus, in this case, the cost paid for activating the interfaces used by x_i is less than $c(W_i) + c(W_{i+1})$ and the same holds for solutions W_i^* and W_i^{MK}. It follows that, for each $1 \leq i \leq n-1$ the cost paid for activating interfaces in W_i in nodes x_i and x_{i-1} is at most $2c(W_i)$ and the overall cost of the solution provided by $\mathcal{C}$ is less than or equal to $2\sum_{i=1}^{n-1} c(W_i)$. As from Corollary 3 we are using in each edge a $(1+\epsilon)$-approximation algorithm for the knapsack problem, it follows that: $2\sum_{i=1}^{n-1} c(W_i) \leq 2\sum_{i=1}^{n-1}(1+\epsilon)c(W_i^{MK})$. As W_i^{MK} is an optimal solution for *MinKP* on edge e_i which guarantees a bandwidth of B, $c(W_i^{MK}) \leq c(W_i^*)$, for each $1 \leq i \leq n-1$, and hence: $2(1+\epsilon)\sum_{i=1}^{n-1} c(W_i^{MK}) \leq 2(1+\epsilon)\sum_{i=1}^{n-1} c(W_i^*) \leq$ $2(1+\epsilon)\left(\sum_{i=1}^{n-2} c(W_i^* \cup W_{i+1}^*) + c(W_{n-1}^*)\right) \leq 2(1+\epsilon)\,\text{OPT}$, where the two last inequalities follow from the fact that in an optimal solution the cost of activating interfaces for each node x_i is $c(W_i^* \cup W_{i+1}^*) \geq c(W_i^*)$ and the overall cost is $\text{OPT} = c(W_1^*) + \sum_{i=1}^{n-2} c(W_i^* \cup W_{i+1}^*) + c(W_{n-1}^*)$.

The complexity of $\mathcal{C}$ is $O(n\frac{k^2}{\epsilon})$ as it is composed of $n-1$ executions of algorithm $\mathcal{A}$ of Theorem 4 which requires $O(k)$ time, and $n-1$ executions of algorithm from [10] which requires $O(\frac{k^2}{\epsilon})$ time. By defining $\epsilon' = 2\epsilon$, Algorithm $\mathcal{C}$ provides a $(2+\epsilon')$-approximated solution and requires $O(|V|\frac{k^2}{\epsilon'})$ time. □

When the input graph is a cycle, since there are two paths from s to t, it is not always clear how the bandwidth B must be split among the two possible ways. However, the following theorem can be stated for the bounded case.

Theorem 7. *BCMI is solvable within $O(|V|)$ time in the bounded case when the input graph is a cycle.*

5 Conclusion

We have considered the Bandwidth Constraints in Multi-Interface Networks problem. We focused on problem hardness and approximation factors in general and more specific settings. The obtained results have shown that the problem is *NP*-hard to be optimally or approximately solved. Polynomial algorithms for special cases have been provided. Further investigation for better performing approximation algorithms or heuristics remain challenging problems. Another interesting issue is to study the problem from a distributed point of view.

References

1. Ahuja, R., Magnanti, T., Orlin, J.: Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, Englewood Cliffs (1993)
2. Athanassopoulos, S., Caragiannis, I., Kaklamanis, C., Papaioannou, E.: Energy-efficient communication in multi-interface wireless networks. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 102–111. Springer, Heidelberg (2009)
3. Bahl, P., Adya, A., Padhye, J., Walman, A.: Reconsidering wireless systems with multiple radios. SIGCOMM Comput. Commun. Rev. 34(5), 39–46 (2004)
4. Caporuscio, M., Charlet, D., Issarny, V., Navarra, A.: Energetic Performance of Service-oriented Multi-radio Networks: Issues and Perspectives. In: 6th Int. Workshop on Software and Performance (WOSP), pp. 42–45. ACM Press, New York (2007)
5. Cavalcanti, D., Gossain, H., Agrawal, D.: Connectivity in multi-radio, multi-channel heterogeneous ad hoc networks. In: IEEE 16th Int. Symp. on Personal, Indoor and Mobile Radio Communications (PIMRC), pp. 1322–1326. IEEE, Los Alamitos (2005)
6. D'Angelo, G., Di Stefano, G., Navarra, A.: Minimizing the Maximum Duty for Connectivity in Multi-Interface Networks. In: Zhong, F. (ed.) COCOA 2010, Part II. LNCS, vol. 6509, pp. 254–267. Springer, Heidelberg (2010)
7. Draves, R., Padhye, J., Zill, B.: Routing in multi-radio, multi-hop wireless mesh networks. In: 10th Annual International Conference on Mobile Computing and Networking (MobiCom), pp. 114–128. ACM, New York (2004)
8. Faragó, A., Basagni, S.: The effect of multi-radio nodes on network connectivity—a graph theoretic analysis. In: IEEE Int. Workshop on Wireless Distributed Networks (WDM). IEEE, Los Alamitos (2008)
9. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
10. Gens, G., Levner, E.: Computational complexity of approximation algorithms for combinatorial problems. In: Becvar, J. (ed.) MFCS 1979. LNCS, vol. 74, pp. 292–300. Springer, Heidelberg (1979)
11. Görtz, S., Klose, A.: Analysis of some greedy algorithms for the single-sink fixed-charge transportation problem. Journal of Heuristics 15(4), 331–349 (2009)
12. Güntzer, M.M., Jungnickel, D.: Approximate minimization algorithms for the 0/1 Knapsack and Subset-Sum Problem. Operations Research Letters 26(2), 55–66 (2000)
13. Klasing, R., Kosowski, A., Navarra, A.: Cost Minimization in Wireless Networks with a Bounded and Unbounded Number of Interfaces. Networks 53(3), 266–275 (2009)
14. Kosowski, A., Navarra, A., Pinotti, M.C.: Exploiting Multi-Interface Networks: Connectivity and Cheapest Paths. Wireless Networks 16(4), 1063–1073 (2010)
15. Lawer, E.: Combinatorial Optimization: Networks and Matroids. Holt, Rinehart, and Winston (1976)

A Privacy-Preserving ID-Based Group Key Agreement Scheme Applied in VPAN

Yoni De Mulder, Karel Wouters, and Bart Preneel

Katholieke Universiteit Leuven
Dept. Electrical Engineering-ESAT/SCD/IBBT-COSIC
Kasteelpark Arenberg 10, 3001 Heverlee, Belgium
{yoni.demulder,karel.wouters,bart.preneel}@esat.kuleuven.be

Abstract. In 2008, Wan et al. presented an anonymous ID-based group key agreement scheme for wireless networks, for which they claim that it ensures anonymity and unlinkability of the group members, as well as forward and backward secrecy of the group session key. In this paper, we show that forward and backward secrecy do not hold for the protocol. We propose a correction that introduces a shielding factor that protects each member's input to the group key. we also introduce a new feature that assures the correctness of the key as computed by all group members. This results in an increased computation cost, due to extra public key operations, and a similar communication cost. We also show in which practical setting the protocol can be deployed.

Keywords: Privacy, Group key agreement, ID-based cryptography.

1 Introduction

In this paper, we propose an improved version of the anonymous ID-based group key agreement scheme for wireless networks by Wan *et al.* [1], and present a scenario in which such a scheme can be used. The original scheme claims to provide group member anonymity from outside eavesdroppers, and forward and backward group key secrecy from leaving resp. joining group members. We show that these claims are incorrect and propose an improved protocol. In Sect. 2, we introduce ID-based cryptography, the original protocol by Wan *et al.* and its vulnerabilities. The main problem with the protocol is that most of the shares of the previously agreed group key remain unaltered in the computation of the new group key, such that joining/leaving members can reconstruct the old/new group key. The improvement, presented in Sect. 3 involves the introduction of a session ID to protect the previously established group key shares. In Sect. 4 we analyse the performance loss, and indicate why the improvements fix the original protocol. Finally, we show how the protocol can be used in the practical setting of a Virtual Private Ad Hoc Network (VPAN), and we conclude with ideas for future work.

The setting in which we operate is as follows: a dynamic set of devices wants to establish a shared secret group key in a privacy-preserving way. 'Dynamic' means that devices can join or leave, albeit at a slow pace (a couple of devices per hour). Encryption with an authenticated group key will ensure confidentiality, but we also require the following:

- **Anonymity:** a group member remains anonymous for an outsider;
- **Unlinkability:** an outsider should be unable to link a group member across sessions with different group keys;
- **Backward and forward secrecy:** only the members that were involved in the group key generation should be able to construct the group key. Being part of the group in the past (forward secrecy) or the future (backward secrecy) leaks no information on the 'current' group key.
- **Perfect forward secrecy:** in case of master key compromise, all previous group keys should remain secret.

2 ID-Based Protocol by Wan *et al.*

The group key agreement scheme of Wan *et al.* in [1] is based on *ID-based public-key cryptography* in which the public key of a user is derived from its identity. Identity Based Encryption (IBE) was proposed by Shamir [2] in 1984, and one of the first practical IBE schemes [3, 4] is based on bilinear maps. In IBE, a trusted server S, called a Private Key Generator (PKG), provides private keys for all participating group members using a randomly chosen master secret s and the identity of each user, after checking his ID. The focus on this paper is on how to fix the scheme of Wan *et al.*, but many similar schemes have been proposed; one example of similar work can be found in [5], which also contains some references to other schemes, and a survey [6] of key agreement protocols. The mathematical setup for the protocol in [1] can be summarised as follows: the PKG selects two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of order q for some large prime q, and a bilinear mapping $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2 : (P_1, P_2) \mapsto Q$. The PKG determines the generator $P \in \mathbb{G}_1$, a master secret $s \in_R \mathbb{Z}_q \setminus \{0\}$, and a public value $P_{\text{pub}} = sP$.

The public parameters $\langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, H_1, P, P_{\text{pub}} \rangle$ are distributed to all users in the system, where H_1 is a hash function, $H_1 : \{0,1\}^* \to \mathbb{G}_1$, used to embed identity values in $\mathbb{G}_1$: For a user with identity U_i, the PKG generates the private key $\text{PrK}_i = sH_1(U_i)$, for which the corresponding public key is the user's identity $\text{PuK}_i = U_i$. In the remainder of the text, $u_i = H_1(U_i)$.

The building blocks of the anonymous ID-based group key agreement protocol presented by Wan *et al.*, which is based on [7, 8], are: group initialisation, the join protocol and the leave protocol.

Initialisation Protocol

The initialisation protocol is executed when an initiator U_1 wants to have a private group session with a set of users $\{U_2, \dots, U_n\}$.

Round 1: $U_1 \to U_i : [\text{E}_{\text{PuK}_i}(\mathcal{L}, \text{Sig}_{\text{PrK}_1}(\mathcal{L})),\ r_1P]$, where $r_1 \in_R \mathbb{Z}_q$ and $\mathcal{L} = U_1\| \dots \|U_n\|Nym_1\| \dots \|Nym_n$, the concatenation of the identities U_i and the related pseudonyms Nym_i. $\mathcal{L}$ is signed with U_1's private key, and encrypted with the recipient's (U_i) public key.

Round 2: $U_i \rightarrow U_{i-1}, U_{i+1} : [Nym_i, r_iP]$: The users $U_i, (i \neq 1)$ decrypt the message from Round 1, retrieve their pseudonym Nym_i, which is sent, together with r_iP, $r_i \in_R \mathbb{Z}_q$ to the users U_{i-1} and U_{i+1}:
Round 3: $U_i \rightarrow * : [Nym_i, X_i = \frac{k_i}{k_{i-1}}]$: Each U_i broadcasts his Nym_i with a key share $X_i = k_i/k_{i-1}$, depending on his private key $\mathrm{PrK}_i = su_i$, the random number r_i and the points $r_{i-1}P$ and $r_{i+1}P$ from Round 2:
$k_i = h(\hat{e}(u_{i+1}, \mathrm{PrK}_i)\|r_ir_{i+1}P)$, $k_{i-1} = h(\hat{e}(u_{i-1}, \mathrm{PrK}_i)\|r_ir_{i-1}P)$.[1] The bilinear property of the mapping $\hat{e}$ ensures the consistency of the subkeys k_i: $\hat{e}(u_{i+1}, \mathrm{PrK}_i) = \hat{e}(u_{i+1}, su_i) = \hat{e}(u_i, su_{i+1}) = \hat{e}(u_i, \mathrm{PrK}_{i+1})$.
Group key K generation: Each user U_i receives all $X_j, (j \neq i)$, and computes[2] the 'subkeys' k_{i+1},k_{i+2},...,k_{i+n-1}, from his own subkey k_i:

$$k_{i+1} = k_iX_{i+1}, k_{i+2} = k_{i+1}X_{i+2}, \ldots \; k_{i+n-1} = k_{i-1} = k_{i+n-2}X_{i+n-1} = k_{i-2}X_{i-1} \; .$$

Then U_i verifies $k_{i+n-1}X_{i+n} = k_i$, and forms the group key $K = H(k_1\|k_2\|\ldots\|k_n)$.[3] Finally, each user $U_i, (i \neq 1)$ sends $H(K\|U_1\|U_2\|\ldots\|U_n)$ to the initiator U_1, who checks the consistency of the group key K.

Join Protocol

When the join protocol is executed to add a new user U_{n+1} to the group, the group key is updated to ensure backward secrecy.
Round 1: U_1 generates Nym_{n+1} for U_{n+1} and initiates the protocol:

$$U_1 \rightarrow U_n : \mathrm{E}_{\mathrm{PuK}_n}(\mathcal{L}_1\|\mathrm{Sig}_{\mathrm{PrK}_1}(\mathcal{L}_1)), \quad \mathcal{L}_1 = U_{n+1}\|Nym_{n+1} \tag{1}$$

$$U_1 \rightarrow U_{n+1} : \mathrm{E}_{\mathrm{PuK}_{n+1}}(\mathcal{L}_2\|\mathrm{Sig}_{\mathrm{PrK}_1}(\mathcal{L}_2)), \tag{2}$$

$$\mathcal{L}_2 = U_1\|Nym_1\|r_1P\|U_n\|Nym_n\|r_nP\|\mathcal{L}_1 \;, \tag{3}$$

Round 2: U_{n+1} obtains Nym_{n+1}, chooses $r_{n+1} \in_R \mathbb{Z}_q$ and computes two subkeys $k_{n+1} = h(\hat{e}(u_1, \mathrm{PrK}_{n+1})\|r_{n+1}r_1P)$ and $k'_n = h(\hat{e}(u_n, \mathrm{PrK}_{n+1}))\|r_{n+1}r_nP)$. U_{n+1} then sends his information to U_1 and U_n:

$$U_{n+1} \rightarrow U_1, U_n : Nym_{n+1}, r_{n+1}P, X_{n+1}, \quad \text{where } X_{n+1} = k_{n+1}/k'_n. \tag{4}$$

Round 3: U_1 and U_n compute k_{n+1} and k'_n respectively:
$k_{n+1} = h(\hat{e}(u_{n+1}, \mathrm{PrK}_1)\|r_1r_{n+1}P)$, $k'_n = h(\hat{e}(u_{n+1}, \mathrm{PrK}_n)\|r_nr_{n+1}P)$. Then they compute the new X-values $X'_1 = k_1/k_{n+1}$ and $X'_n = k'_n/k_{n-1}$, and distribute them to the new group:

$$U_n \rightarrow U_1 : X'_n \;,$$

$$U_1 \rightarrow U_{n+1} : \mathrm{E}_{\mathrm{PuK}_{n+1}}(\mathcal{N}_1\|\mathrm{Sig}_{\mathrm{PrK}_1}(\mathcal{N}_1)), \quad \mathcal{N}_1 = X'_1\|X_2\|\ldots\|X_{n-1}\|X'_n \;,$$

$$U_1 \rightarrow * : E_K(\mathcal{N}_2\|\mathrm{Sig}_{\mathrm{PrK}_1}(\mathcal{N}_2)), \quad \mathcal{N}_2 = X'_1\|X_{n+1}\|X'_n \;.$$

Group key K update: Every group member (including U_{n+1}) can now compute all the subkeys k_i, $i = 1, \ldots, n+1$ with the altered k'_n and the new k_{n+1} with the following sequence of calculations[4] (illustrated for user U_{n-1}):

[1] $h : \mathbb{G}_2 \times \mathbb{G}_1 \rightarrow \{0,1\}^k$ is a hash function with security parameter k.
[2] Subscript numbers are considered modulo n.
[3] $H : \{0,1\}^* \rightarrow \{0,1\}^k$ is a hash function.
[4] Note that all subscript numbers are considered modulo $n+1$.

$k'_n = k_{n-1}X'_n$; $k_{n+1} = k'_n X_{n+1}$; $k_{n+2} = k_1 = k_{n+1}X'_1$; $k_{2n} = k_{n-1} = k_{n-2}X_{n-1}$. The updated group key $K' = H(k_1\|k_2\|\ldots\|k'_n\|k_{n+1})$, is formed and checked in the same way as in the initialisation protocol.

Leave Protocol

This protocol ensures forward secrecy when a group member U_i leaves.

Round 1: U_1 assigns new pseudonyms to U_{i-1} and U_{i+1}, using the current group key K:

$$U_1 \rightarrow U_{i-1}, U_{i+1} : E_K(\mathcal{L}\|\mathrm{Sig}_{\mathrm{PrK}_1}(\mathcal{L})), \qquad (5)$$
$$\mathcal{L} = U_i\|Nym_i\|U_{i-1}\|Nym'_{i-1}\|U_{i+1}\|Nym'_{i+1} \ .$$

Round 2: U_{i-1} and U_{i+1} verify the signature of U_1 and exchange new parameters $r'_{i-1}P$ and $r'_{i+1}P$ using their new pseudonyms:

$$U_{i-1} \rightarrow U_{i+1} : Nym'_{i-1}, r'_{i-1}P, \qquad U_{i+1} \rightarrow U_{i-1} : Nym'_{i+1}, r'_{i+1}P \ .$$

Round 3: U_{i-1} and U_{i+1} recompute their (equal) subkeys k'_{i-1} and k'_i:

$$U_{i-1} : k'_{i-1} = h(\hat{e}(u_{i+1}, \mathrm{PrK}_{i-1})\|r'_{i-1}r'_{i+1}P) \ ,$$
$$U_{i+1} : k'_i = h(\hat{e}(u_{i-1}, \mathrm{PrK}_{i+1})\|r'_{i+1}r'_{i-1}P) \ .$$

Next, the updated X-values are computed and distributed:

$$U_{i-1} \rightarrow U_1 : X'_{i-1} = \frac{k'_{i-1}}{k_{i-2}}, \quad U_{i+1} \rightarrow U_1 : X'_{i+1} = \frac{k_{i+1}}{k'_i} = \frac{k_{i+1}}{k'_{i-1}},$$
$$U_1 \rightarrow * : E_K(\mathcal{N}\|\mathrm{Sig}_{\mathrm{PrK}_1}(\mathcal{N})), \quad \mathcal{N} = U_i\|U_{i-1}\|U_{i+1}\|X'_{i-1}\|X'_{i+1}. \qquad (6)$$

Group key K update: The remaining $n-1$ group members can now compute the updated group key $K' = H(k_1\|k_2\|\ldots\|k'_{i-1}\|k_{i+1}\|\ldots\|k_n)$, which is checked in the same way as in the initialisation protocol.

Security Properties

Forward and backward secrecy are not met in the described protocols, contrary to the claims in [1]. In the adversary model, we assume a global, active attacker who is capable of eavesdropping, injecting, modifying or dropping messages within the network at will.

Forward secrecy: In the leave protocol described above, forward secrecy is not guaranteed. In round 3 of the protocol, the leaving member U_i can obtain the values X'_{i-1} and X'_{i+1} in two ways: they are sent unprotected to U_1 AND they are broadcasted under the old group key K in (6). U_i already knows k_i $(i = 1, \ldots, n)$, from the old group key, and only needs to recover the updated subkey $k'_{i-1} = k'_i$, to get the new group key $K' = H(k_1\|\ldots\|k_{i-2}\|\mathbf{k'_{i-1}}\|k_{i+1}\|\ldots\|k_n)$, which can be done from X'_{i-1} and X'_{i+1}:

$$k'_{i-1} = k_{i-2}X'_{i-1}, \quad k'_i = k_{i+1}X'_{i+1}.$$

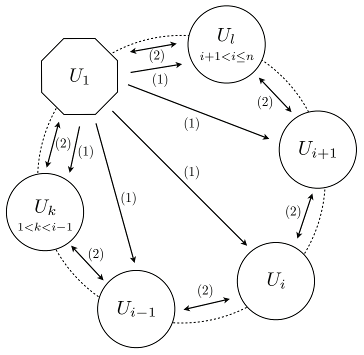
Backward secrecy: Backward secrecy is not ensured by the group member join protocol. At the end of this protocol, user U_{n+1} computes the new group key

$K' = H(k_1\|\ldots\|k_{n-1}\|k'_n\|k_{n+1})$ knowing all subkeys k_i. To be able to compute K, only k_n is missing, which can be computed from $X_1 = k_1/k_n$ or $X_n = k_n/k_{n-1}$, sent around unencrypted in the previous session, which U_{n+1} could have monitored.

3 Improved Protocol

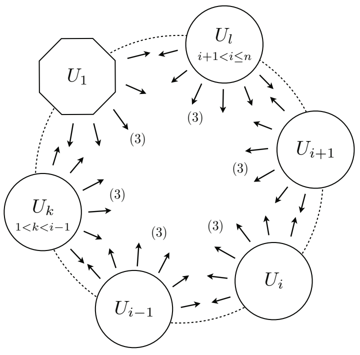
Forward and backward secrecy can be ensured by the following improved leave/join protocols. Our improvement, partially based on ideas of Jung [9], makes use of a session ID, denoted as SID. This random string, unique for each new group session is newly generated and distributed by U_1 in each join/leave protocol. The SID will blind all subkeys k, such that each member will affect the updated group key. Below we describe the difference to the original protocol, for the building blocks described in Sect. 2.

3.1 Initialisation Protocol



(1) $U_1 \to U_i : \mathrm{E}_{\mathrm{PuK}_i}(\mathcal{L}\|\mathrm{Sig}_{\mathrm{PrK}_1}(\mathcal{L})), r_1P$,
$\mathcal{L} = U_1\|\ldots\|U_n\|Nym_1\|\ldots\|Nym_n\|\mathrm{SID}$
(2) $U_i \to U_{i-1}, U_{i+1} : Nym_i, r_iP$

(a) Round-step 1 & 2



(3) $U_i, (i = 1,\ldots,n) \to * : Nym_i, X_i$,
$X_i = H(k_i, \mathrm{SID}) \oplus H(k_{i-1}, \mathrm{SID})$

(b) Round-step 3

Fig. 1. Initialisation Protocol

Round 1: U_1 generates a SID and adds it to the encrypted and signed message, sent to each user U_i, in which $\mathcal{L} = U_1\|\ldots\|U_n\|Nym_1\|\ldots\|Nym_n\|\mathrm{SID}$:

$$U_1 \to U_i : \mathrm{E}_{\mathrm{PuK}_i}(\mathcal{L}\|\mathrm{Sig}_{\mathrm{PrK}_1}(\mathcal{L})), r_1P\,. \tag{7}$$

Round 2: No changes to the original protocol.
Round 3: Each U_i calculates k_i and k_{i-1} as in the original. X_i is different:

$$X_i = H(k_i\|\mathrm{SID}) \oplus H(k_{i-1}\|\mathrm{SID})\,.$$

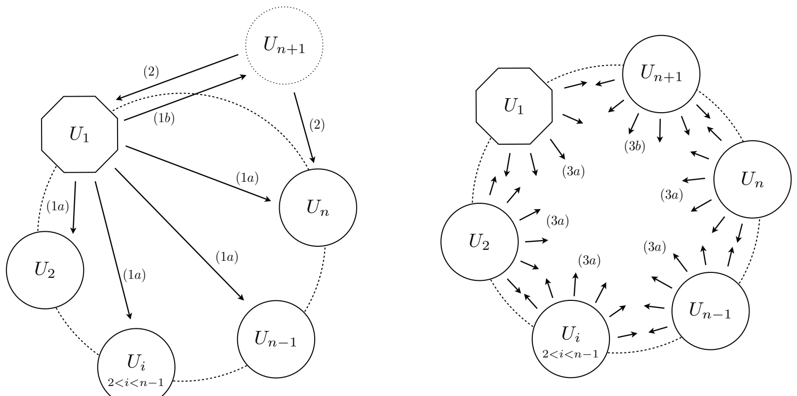
X_i now also depends on SID, and will be updated with each change of the group. As before Nym_i, X_i is broadcasted to all other users.

Group key K generation: Each U_i executes a series of calculations:

$$H(k_{i+1}, \mathrm{SID}) = H(k_i, \mathrm{SID}) \oplus X_{i+1}\ ,H(k_{i+2}, \mathrm{SID}) = H(k_{i+1}, \mathrm{SID}) \oplus X_{i+2}\ ,$$
$$\dots$$
$$H(k_{i+n-1}, \mathrm{SID}) = H(k_{i+n-2}, \mathrm{SID}) \oplus X_{i+n-1}\ .$$

At the end, U_i verifies if $H(k_{i+n}, \mathrm{SID}) = H(k_{i+n-1}, \mathrm{SID}) \oplus X_{i+n} = H(k_i, \mathrm{SID})$, and the group key K is formed: $K = H(H(k_1, \mathrm{SID}) \| H(k_2, \mathrm{SID}) \| \dots \| H(k_n, \mathrm{SID}))$, **Group key K consistency verification:** In the original scheme, each $U_i, (i \neq 1)$ sends the *same* check value to U_1, which means that U_1 cannot verify its origin. In our version of the protocol, the confirmation message can only be generated by a legitimate U_i: $U_i \rightarrow U_1 : E_K(U_i \| \mathrm{Sig}_{\mathrm{PrK}_i}(K))$.

3.2 Join Protocol



(1a) $U_1 \rightarrow * : E_K(\mathcal{L}_1 \| \mathrm{Sig}_{\mathrm{PrK}}(\mathcal{L}_1))$,
$\mathcal{L}_1 = U_1 \| \dots \| U_n \| U_{n+1} \| Nym'_1 \| \dots \| Nym'_n \| Nym_{n+1} \| \mathrm{SID}'$
(1b) $U_1 \rightarrow U_{n+1} : \mathrm{E}_{\mathrm{PuK}_{n+1}}(\mathcal{L}_2 \| \mathrm{Sig}_{\mathrm{PrK}_1}(\mathcal{L}_2))$,
$\mathcal{L}_2 = \mathcal{L}_1 \| r_1 P \| r_n P$
(2) $U_{n+1} \rightarrow U_1, U_n : Nym_{n+1}, r_{n+1} P$

(a) Round-step 1 & 2

(3a) $U_i, (i = 1, \dots, n) \rightarrow * : Nym'_i, X'_i$,
$X'_1 = H(k_1, \mathrm{SID}') \oplus H(k_{n+1}, \mathrm{SID}')$
$X'_i|_{2 \leq i \leq n-1} = H(k_i, \mathrm{SID}') \oplus H(k_{i-1}, \mathrm{SID}')$
$X'_n = H(k'_n, \mathrm{SID}') \oplus H(k_{n-1}, \mathrm{SID}')$
(3b) $U_{n+1} \rightarrow * : Nym_{n+1}, X_{n+1}$,
$X_{n+1} = H(k_{n+1} \| \mathrm{SID}') \oplus H(k'_n \| \mathrm{SID}')$

(b) Round-step 3

Fig. 2. Join Protocol

Round 1: U_1 generates a new session ID, denoted as SID'. Next U_1 informs all members $U_i, (i = 1, \dots, n)$ and U_{n+1} about U_{n+1}'s joining, and assigns new pseudonyms Nym'_i to all U_i and a pseudonym Nym_{n+1} to U_{n+1}[5] :

$$U_1 \rightarrow * : E_K(\mathcal{L}_1 \| \mathrm{Sig}_{\mathrm{PrK}_1}(\mathcal{L}_1)),$$
$$\mathcal{L}_1 = U_1 \| \dots \| U_n \| U_{n+1} \| Nym'_1 \| \dots \| Nym'_n \| Nym_{n+1} \| \mathrm{SID}'\ , \quad (8)$$
$$U_1 \rightarrow U_{n+1} : \mathrm{E}_{\mathrm{PuK}_{n+1}}(\mathcal{L}_2 \| \mathrm{Sig}_{\mathrm{PrK}_1}(\mathcal{L}_2)), \quad \mathcal{L}_2 = \mathcal{L}_1 \| r_1 P \| r_n P\ , \quad (9)$$

[5] Note that the encryption algorithms in (8) and (9) are different.

Round 2: After decryption, U_{n+1} obtains the pseudonyms of all $U_i, (i = 1, \ldots, n+1)$, chooses $r_{n+1} \in_R \mathbb{Z}_q$ and computes k_{n+1} and k'_n as in the original protocol. He also computes his own $X_{n+1} = H(k_{n+1} \| \mathrm{SID}') \oplus H(k'_n \| \mathrm{SID}')$ and sends $Nym_{n+1}, r_{n+1}P$ to U_1 and U_n.
Round 3: Upon reception of $r_{n+1}P$, U_1 and U_n can compute subkeys k_{n+1} and k'_n as before and recompute their X-values:

$$U_1 : X'_1 = H(k_1, \mathrm{SID}') \oplus H(k_{n+1}, \mathrm{SID}') \ ,$$
$$U_n : X'_n = H(k'_n, \mathrm{SID}') \oplus H(k_{n-1}, \mathrm{SID}') \ .$$

The other users U_i need to update their X-value as well:

$$U_i, (i = 2, \ldots, n-1) : X'_i = H(k_i, \mathrm{SID}') \oplus H(k_{i-1}, \mathrm{SID}') \ .$$

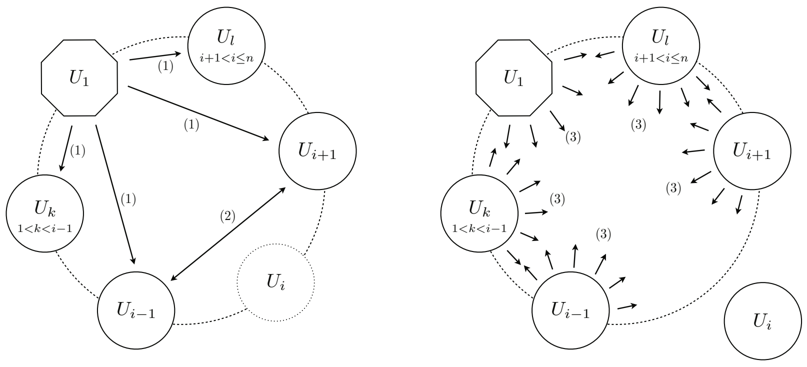
Finally, all group members broadcast their new X-values to all other users along with their new pseudonym:

$$U_i, (i = 1, \ldots, n) \rightarrow * : Nym'_i, X'_i \ , \quad U_{n+1} \rightarrow * : Nym_{n+1}, X_{n+1} \ .$$

Group key K update: Each user can now compute every $H(k_i \| \mathrm{SID}')$, using the X-values, and compute and verify the new group key K' [6]:

$$K' = H(H(k_1, \mathrm{SID}') \| \ldots \| H(k'_n, \mathrm{SID}') \| H(k_{n+1}, \mathrm{SID}')) \ .$$

3.3 Leave Protocol



(1) $U_1 \rightarrow * : E_K(\mathcal{L} \| \mathrm{Sig}_{\mathrm{PrK}_1}(\mathcal{L}))$,
$\mathcal{L} = U_1 \| \ldots \| U_n \| Nym'_1 \| \ldots \| Nym'_{i-1} \| Nym_i \|$
$Nym'_{i+1} \| \ldots \| Nym'_n \| \mathrm{SID}'$

(2) $U_{i-1} \rightarrow U_{i+1} : Nym'_{i-1}, r'_{i-1}P$
$U_{i+1} \rightarrow U_{i-1} : Nym'_{i+1}, r'_{i+1}P$

(a) Round-step 1 & 2

(3) $U_i, (i = 1, \ldots, i-1, i+1, \ldots, n) \rightarrow * : Nym'_i, X'_i$,
$X'_{i-1} = H(k'_{i-1}, \mathrm{SID}') \oplus H(k_{i-2}, \mathrm{SID}')$
$X'_{i+1} = H(k_{i+1}, \mathrm{SID}') \oplus H(k'_i, \mathrm{SID}')$
$X'_i|_{i=1,\ldots,i-2,i+2,\ldots,n} = H(k_i, \mathrm{SID}') \oplus H(k_{i-1}, \mathrm{SID}')$

(b) Round-step 3

Fig. 3. Leave Protocol

Round 1: When U_i wants to leave the current session, U_1 generates a new SID' and informs all remaining group members by broadcasting:

[6] The consistency verification is identical to the one in the initialisation protocol.

$$U_1 \to * : E_K(\mathcal{L} \| \mathrm{Sig}_{\mathrm{PrK}_1}(\mathcal{L})),$$
$$\mathcal{L} = U_1 \| \dots \| U_n \| Nym'_1 \| \dots \| Nym'_{i-1} \| Nym_i \| Nym'_{i+1} \| \dots \| Nym'_n \| \mathrm{SID}' \ ,$$

in which new pseudonyms Nym'_i are assigned.
Round 2: Upon reception of their new pseudonym, U_{i-1} and U_{i+1} exchange their new values $r'_{i-1}P$ and $r'_{i+1}P$, as in the original protocol.
Round 3: Then, U_{i-1} and U_{i+1} recompute their subkeys k'_{i-1} and k'_i, again as in the original protocol, and compute their X-values:

$$U_{i-1} : X'_{i-1} = H(k'_{i-1}, \mathrm{SID}') \oplus H(k_{i-2}, \mathrm{SID}') \ ,$$
$$U_{i+1} : X'_{i+1} = H(k_{i+1}, \mathrm{SID}') \oplus H(k'_i, \mathrm{SID}') \ .$$

The other users U_i need to update their X-value as well:

$$U_i, (i = 1, \dots, i-2, i+2, \dots, n) : X'_i = H(k_i, \mathrm{SID}') \oplus H(k_{i-1}, \mathrm{SID}') \ .$$

Finally, all remaining group members broadcast their new X-value:

$$U_i, (i = 1, \dots, i-1, i+1, \dots, n) \to * : Nym'_i, X'_i \ .$$

Group key K update: Each user can now compute every $H(k_i \| \mathrm{SID}')$[7], using the X-values, and compute and verify the new group key K' [8]:

$$K' = H(H(k_1, \mathrm{SID}') \| \dots \| H(k'_{i-1}, \mathrm{SID}') \| H(k_{i+1}, \mathrm{SID}') \| \dots \| H(k_n, \mathrm{SID}')).$$

4 Security and Performance Analysis

4.1 Security

Here we informally show that our scheme ensures all the requirements, including forward/backward secrecy. In the security analysis below, we assume that the adversary knows the current SID, unless otherwise stated.
Anonymity: The identities of the users U_i are encrypted at all times in the protocols, either by anonymous ID-based encryption using their public keys PuK_i or by symmetric encryption using the group key K. In all protocols, identities are masked by pseudonyms constructed by the initiator U_1. The use of identities and private keys in the calculation of the values X_i is masked by hashing with a random point in $\mathbb{G}_1$; identity information cannot be retrieved without the master secret s or the user's private key PrK_i.
Unlinkability: Anonymity alone only hides the real identities of the group members. To prevent an adversary from tracing a user by his pseudonym, pseudonyms or X-values are never re-used when the group changes and a new group key needs to be established. Therefore, a user U_i is able to join or leave a private group session anonymously.
Group key secrecy and perfect forward secrecy: If an attacker knows SID, the group key $K = H(H(k_1, \mathrm{SID}) \| H(k_2, \mathrm{SID}) \| \dots \| H(k_n, \mathrm{SID}))$ remains secure: to retrieve K, he needs access to at least one user's shared subkey k_i, to compute the other subkeys k_j from the broadcasted X-values. The subkey $k_i = h(\hat{e}(u_{i+1}, \mathrm{PrK}_i) \| r_i r_{i+1} P)$ cannot be computed without knowing U_{i+1}, the private key PrK_i and the element $r_i r_{i+1} P$:

[7] The equality $k'_{i-1} = k'_i$ is necessary for the correctness of the sequence of calculations.
[8] The consistency verification is identical to the one in the initialisation protocol.

– The IDs U_i are encrypted, and only known to the participating members;
– The private key of the users is never shared and can only be computed with the master secret s;
– The element $r_i r_{i+1} P$ cannot be computed from the exchanged $r_i P$ and $r_{i+1} P$ because of the ECDHP assumption [10].

Knowing the identities U_i next to SID, is also insufficient to compute k_i without having knowledge of the master secret s or the private keys PrK_i, since $\hat{e}(u_{i+1}, \mathrm{PrK}_i) = \hat{e}(u_{i+1}, su_i)$. The argumentation above shows that the group key K cannot be retrieved by the adversary in our threat model.

Even in case when the long-term master secret s is compromised and the identities of the participating members are revealed, group keys from previous stages remain uncompromised, because of the last argument in the list above, such that the protocol is also perfectly forward secret.

Forward and backward secrecy: Forward and backward secrecy should be ensured for the leaving and joining protocol respectively. Each time the group membership changes, the initiator U_1 introduces a new SID' and the group key is updated. While each user U_i still computes both subkeys k_i and k_{i-1}, he will share $H(k_i \| \mathrm{SID}')$, instead of the unprotected k_i. This is done by broadcasting $X'_i = H(k_i \| \mathrm{SID}') \oplus H(k_{i-1} \| \mathrm{SID}')$. A joining/leaving member is unable to compromise a past/future group key K:

1. the only subkeys k a joining/leaving member knows are his own, which are both newly generated or updated during the join/leave protocol such that the knowledge of the previous SID/updated SID' cannot be used to compute a previous/updated group key;
2. each user's shared contribution and the corresponding X-value are SID-dependent, and are updated and re-broadcasted in the join/leave protocol.

Forward secrecy for the leaving protocol: In round 1, the leaving member U_i gains knowledge of the updated SID' since it is encrypted using the old group key K. This is no problem for the forward secrecy, as U_i can only generate $H(k_i \| \mathrm{SID}')$ and $H(k_{i-1} \| \mathrm{SID}')$; these are exactly the values that are updated by U_{i-1} and U_{i+1} in round 3, and shared through updated X'-values:

$$U_{i-1} : X'_{i-1} = H(k'_{i-1}, \mathrm{SID}') \oplus H(k_{i-2}, \mathrm{SID}') \ ,$$
$$U_{i+1} : X'_{i+1} = H(k_{i+1}, \mathrm{SID}') \oplus H(k'_i, \mathrm{SID}') \ ,$$

where each shared contribution is updated with the new SID'. Because U_i only knows $H(k_{i-1}, \mathrm{SID}'), H(k_{i-2}, \mathrm{SID}), H(k_{i+1}, \mathrm{SID}), H(k_i, \mathrm{SID}')$, he is unable to retrieve any useful information from X'_{i-1} and X'_{i+1} to retrieve the new K'.

Backward secrecy for the joining protocol: In round 2, the joining member U_{n+1} computes both subkeys k_{n+1} and k'_n and shares this through $X_{n+1} = H(k_{n+1} \| \mathrm{SID}') \oplus H(k'_n \| \mathrm{SID}')$. If U_{n+1} knows the previous session ID, i.e. SID, he can compute $H(k_{n+1} \| \mathrm{SID})$ and $H(k'_n \| \mathrm{SID})$. However, this does not give him an advantage in recovering any past group key K since the subkey k_n has been updated. The shared contributions of all other group members are now masked with a new SID', hence knowing $H(k_1 \| \mathrm{SID}')$ in combination with $X_1 = H(k_1 \| \mathrm{SID}) \oplus H(k_n \| \mathrm{SID})$ does not help to retrieve K.

4.2 Performance

In Tables 1 and 2, we give an overview of the communication and computation cost of the new protocol, compared to the original protocol by Wan *et al.*

Communication cost: The number of broadcast messages in the join and leave protocol is higher: each time the group (of size n) changes, each U_i needs to broadcast his updated share $H(k_i \| \mathrm{SID}')$ to all other users.

Table 1. Communication Cost Comparison

Protocols		Rounds	Messages	Unicast	Broadcast
Wan *et al.*	Initialise	3	$4n$	$3n$	n
	Join	3	7	6	1
	Leave	3	7	6	1
Our protocol	Initialise	3	$4n$	$3n$	n
	Join	3	$n+5$	3	$n+2$
	Leave	3	$n+2$	2	n

Computation cost: The increase in signature generations and verifications is due to (1) the initiator U_1 broadcasts his signed message containing the newly reassigned pseudonyms Nym'_i and updated SID$'$ during the join/leave protocol, and (2) the improved group key consistency verification process at the end of each protocol.

Table 2. Computation Cost Comparison

Protocols		ID-based Encryption	Pairing Computation ($\hat{e}$)	Signature Gen./Verif.		Point Multiplication (in $\mathbb{G}$)
Wan *et al.*	Initialise	$n-1$	$2n$	1	$n-1$	$3n$
	Join	3	4	4	$n+3$	5
	Leave	0	2	2	$n+1$	4
Our protocol	Initialise	$n-1$	$2n$	n	$2n-2$	$3n$
	Join	1	4	$n+2$	$2n+1$	5
	Leave	0	2	$n-1$	$2n-4$	4

5 Application: Virtual Private Ad Hoc Network

As more and more mobile devices interconnect though largescale IP networks, new network architectures become important. Virtual Private Ad Hoc Network (VPAN) is a concept that aims to establish a secure virtual network on top of the existing insecure IP base network by combining network virtualisation and ad hoc networking techniques. This concept was proposed and introduced in [11] and [12].

Due to geographical distribution of VPAN entities, clusters of entities – *VPAN Nodes* – that are able to connect to each other directly are formed, in which a special node, the *Gateway Node*, has connectivity to an IP-based access network. Within the same VPAN, clusters are interconnected through their Gateway Nodes. The VPAN membership is self-organising: members need to be able to discover each other and form a secure overlay without user intervention. Additionally, ad hoc routing techniques are used for efficient internal routing. A VPAN is identified by a VPAN prefix, which is prepended to the VPAN Node ID for every node in a VPAN, such that one node can be active in multiple VPANs.

Our privacy-preserving ID-based group key agreement scheme described in Sect. 3 can be applied to the VPAN setting to protect the privacy of VPAN cluster nodes to the outside world as well as to obtain a shared group session key within each cluster while supporting dynamic cluster membership. More specifically, VPAN Nodes remain anonymous and an outside eavesdropper is unable to trace or monitor activities of a specific VPAN Node, or to link the same VPAN Node in clusters of different VPANs.

The initiator U_1, which should have an IP connection, assumes the role of *Gateway Node* GN, while the remaining users $U_i|_{2\leq i\leq n}$ act as *VPAN Nodes*. Hence a group of users $\{U_i|_{1\leq i\leq n}\}$ is here referred to as a cluster of VPAN members $\{GN, U_i|_{2\leq i\leq n}\}$. To ensure that VPAN Nodes can form new clusters or join an existing cluster, the GN of each VPAN cluster broadcasts periodically the following beacon message: $GN \rightarrow * : GN \| \mathrm{Sig}_{\mathrm{PrK}_{GN}}(GN)$, where GN denotes the Gateway Node's public key.

Initialisation Protocol: Cluster initialisation occurs when setting up a new VPAN or forming a new cluster within an existing VPAN. Each VPAN Node U_i receiving GN's beacon, responds with the following encrypted message: $U_i \rightarrow GN : \mathrm{E}_{\mathrm{PuK}_{GN}}(\mathcal{L} \| \mathrm{Sig}_{\mathrm{PrK}_i}(\mathcal{L}))$ with $\mathcal{L} = U_i \| \mathrm{VPAN}_{\mathrm{prefix}}$. At the end, GN has a set of VPAN Nodes $\{U_i|_{2\leq i\leq n}\}$ to form a cluster and to agree upon a shared group session key.

Join Protocol: When a new VPAN Node U_{n+1} wants to join an existing cluster $\{GN, U_i|_{2\leq i\leq n}\}$, he waits for the beacon and responds with the following encrypted message: $U_{n+1} \rightarrow GN : \mathrm{E}_{\mathrm{PuK}_{GN}}(\mathcal{L} \| \mathrm{Sig}_{\mathrm{PrK}_{n+1}}(\mathcal{L}))$ with $\mathcal{L} = U_{n+1} \| \mathrm{VPAN}_{\mathrm{prefix}}$. During the join protocol, the shared group session key is updated to provide backward secrecy.

Leave Protocol: When a VPAN Node U_i wants to leave an existing cluster, the leave protocol is executed to update the group session key and thus to provide forward secrecy.

The protocols themselves remain exactly the same as described in Sect. 3, all initiated by the Gateway Node GN.

6 Conclusion and Future Work

In this paper, we showed that the key agreement protocol by Wan *et al.* does not offer forward and backward secrecy, contrary to their claims. We adjusted the protocol such that these requirements are met and added an extra safeguard at the end of the protocol. The cost for these improvements is a increased computation cost and a moderately higher communication cost.

Future work includes a thorough investigation of the role of the network in which our protocol will operate; this can be done in a VPAN setting, for which test infrastructures exist already. The designers of the referenced VPAN were involved in this work, and collaboration seems possible.

The cost of running an anonymous routing mechanism in a multihop ad hoc network, as suggested by the original protocol authors, cannot be neglected. Furthermore, privacy-preserving ID-based protocols in which multiple members can join and leave simultaneously and in which groups can merge and split are still to be developed.

Acknowledgements

This work was supported in part by Research Council K.U.Leuven (GOA TENSE), by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy), and by IBBT (Interdisciplinary institute for Broad-Band Technology) of the Flemish Government.

References

1. Wan, Z., Ren, K., Lou, W., Preneel, B.: Anonymous id-based group key agreement for wireless networks. In: IEEE WCNC, Network Track (2008)
2. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1985)
3. Boneh, D., Franklin, M.: Identity-Based Encryption from the Weil Pairing. SIAM Journal on Computing 32, 586 (2003)
4. Sakai, R., Ohgishi, K., Kasahara, M.: Cryptosystems based on pairing. In: The 2000 Symposium on Cryptography and Information Security, pp. 26–28 (2000)
5. Dutta, R., Dowling, T.: Secure and efficient group key agreements for cluster based networks. Transactions on Computational Science 4, 87–116 (2009)
6. Dutta, R., Barua, R.: Overview of key agreement protocols. Cryptology ePrint Archive, Report 2005/289 (2005), http://eprint.iacr.org/
7. Burmester, M., Desmedt, Y.: A secure and efficient conference key distribution system (extended abstract). In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 275–286. Springer, Heidelberg (1995)
8. Smart, N.P.: Identity-based authenticated key agreement protocol based on Weil pairing. Electronics Letters 38(13), 630–632 (2002)
9. Jung, B.E.: An efficient group key agreement protocol. IEEE Communications Letters 10(2), 106–107 (2006)
10. Brown, D.: Standards for efficient cryptography, SEC 1: elliptic curve cryptography. Technical report, Certicom Research (2009)
11. Hoebeke, J.: Adaptive Ad Hoc Routing and Its Application to Virtual Private Ad Hoc Networks. PhD thesis, Universiteit Gent (2007)
12. Hoebeke, J., Holderbeke, G., Moerman, I., Dhoedt, B., Demeester, P.: Virtual Private Ad Hoc Networking. Wireless Personal Communications 38(1), 125–141 (2006)

White Space Regions

Shayan Ehsani, MohammadAmin Fazli, Mohammad Ghodsi, MohammadAli Safari*, Morteza Saghafian, and Mohammad Tavakkoli

Dept. of Computer Engineering, Sharif University of Technology
{fazli,ehsani,mtavakoli,saghafian}@ce.sharif.edu,
{ghodsi,msafari}@sharif.edu

Abstract. We study a classical problem in communication and wireless networks called *Finding White Space Regions*. In this problem, we are given a set of antennas (points) some of which are noisy (black) and the rest are working fine (white). The goal is to find a set of convex hulls with maximum total area that cover all white points and exclude all black points. In other words, these convex hulls make it safe for white antennas to communicate with each other without any interference with black antennas. We study the problem on three different settings (based on overlapping between different convex hulls) and find hardness results and good approximation algorithms.

1 Introduction

The problem of finding white space regions arises in the context of *Cognitive Radio* systems. According to FCC Task Force report [1] in 2004 on the under-utilization of the wireless radio environment, primary users did not use their licensed spectrum from 15% to 85% of the time. Therefore, these silence ranges in time and frequency can be exploited by unlicensed users provided that these users do not make any interference with the primary licensed users.

Consider a primary network whose Base Stations(BSs) are fixed and communicate occasionally in a licensed band along with spatially random distributed spectrum sensing base stations of the cognitive radio networks distributed in a large region. Such a network is depicted in Fig. 1 in which primary users are shown by long antennas and the others by short antennas. The goal is to find the largest area in which no primary user transmission is detected. From now on, we call this area the *white space region* or WSR for short. Cognitive radio users can then safely use this WSR to communicate with each other without affecting primary users. Moreover, the more the area of WSR is the more non-licensed users can communicate with each other.

Abachi et al [3] first proposed a model based on the spatial information of the antennas. In their model, each antenna is a point with cartesian coordinates. They colored points corresponding to primary users as black and the rest as

* The research was in part supported by grant no. CS1389-4-09 from the institute for Research in Fundamental Sciences (IPM).

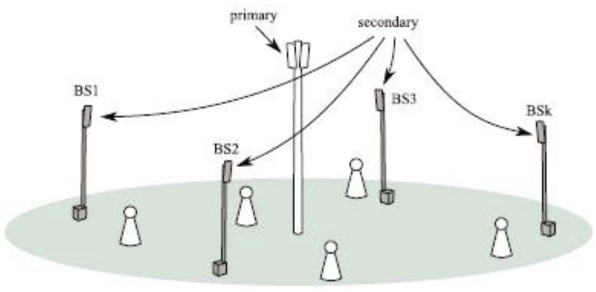


Fig. 1. Sensing Scenario [2]

white. In this model, WSRs are convex regions around white points whose circumcircle does not contain any black point. They proposed a pseudo polynomial algorithm for finding good WSRs.

In another related work, Fischer [4,5] focuses on finding just one WSR with the maximum area. In [4], he proposes a dynamic programming based $\mathcal{O}\left(n^4 log(n)\right)$-time algorithm for finding the maximum area convex polygon which does not contain any black points. In [5] he improved his result by learning techniques.

In this paper, we let more than one convex hull be chosen. Our aim is to find a set of WSRs which covers all white points and whose total area is maximum (We assume a single point or two points connected via a segment as a convex hull of area 0; so, it is always possible to cover all white points with convex hulls). Moreover, no convex hull should contain any black point inside.

1.1 Formal Problem Definition

As mentioned earlier we have a set of n points; some are black (n_b of them) and others are white (n_w of them), and we are looking for a set of convex hulls of the white points that:

- cover all the white points (a single point or a line segment between two points is assumed a convex hull).
- are free from black points.
- the area of their union is maximum.

One question is whether these convex hulls are allowed to have intersection. Depending on wether the convex hulls can overlap, we have three different problems(see Fig. 2):

Definition 1. *Three different problems of finding the white space regions are:*

- **Totally Disjoint Convex Covering (TDCC)***: In this problem WSRs cannot touch each other, i.e. they are not allowed to have common vertices or common area or one's edge lying on another's edge(Fig. 2 part (a))*

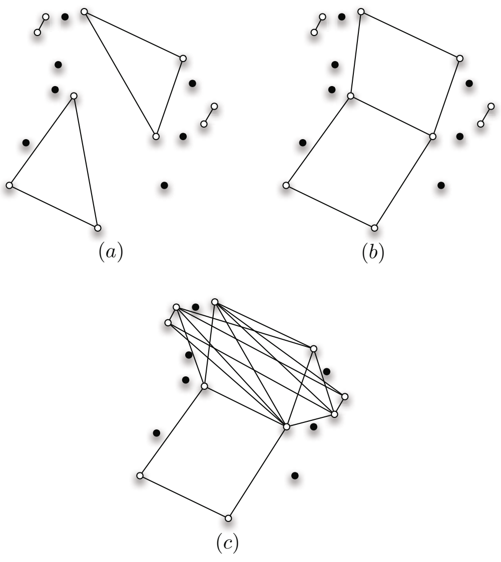


Fig. 2. Three different types of the problem

- **Nonoverlapping Convex Covering (NOCC)**: *In this problem WSRs are not allowed to have common areas but having common vertices is fine (Fig. 2 part (b)).*
- **Convex Covering with no Restriction (CCR)**: *There are no restrictions in regards to intersection (Fig. 2 part (c))*

Our main results are as follows. First, we obtain some hardness results.

Theorem 1. *Both NOCC and TDCC problems are NP-hard.*

Next we obtain two different, but similar approximation algorithms for both of the problems (OPT is the area of the optimal solution).

Theorem 2. *There is an approximation algorithm that computes a set of convex hulls with total area of at least* $(\frac{OPT}{2\log(2n/OPT)+2\log(n)})^{1/4}$ *for the* NOCC *problem.*

Theorem 3. *There is an approximation algorithm that computes a set of convex hulls with total area of at least* $\frac{3\sqrt{3}}{4.\pi}(\frac{OPT}{2\log(2n/OPT)+2\log(n)})^{1/4}$ *for the* TDCC *problem.*

Finally, we propose heuristic algorithms based on Fischer's algorithm and provide experimental and theoretical analysis for them.

For the CCR problem, it's straightforward to find an exact polynomial time algorithm.

Theorem 4. *CCR can be solved in polynomial time.*

Proof. It is enough to consider all possible triangles of white vertices that contain no black vertex and output their union. It is easy to see that this output is the maximum area convex covering. Since the number of possible triangles is polynomial ($\leq \binom{n}{3}$) the running time will be polynomial.

The structure of our paper is as follows. In the next section, we prove the hardness results. Then, in section 3, we propose good approximation algorithms for the problems. In section 4, we propose greedy algorithms based on Fischer's works [4,5]. We show by experimental evaluations that these greedy algorithms work well in practice, but may behave arbitrarily bad in the worst case.

2 Hardness Results

In this section we will prove the NP-Hardness of NOCC. The hardness of TDCC follows similarly. The idea is to reduce the independent set problem on triangle-free planar graphs. A proof for the hardness of the latter problem can be found at [6].

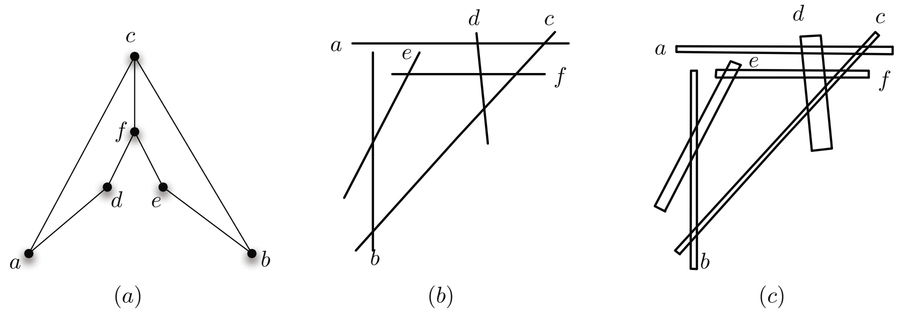


Fig. 3. Forming an intersection graph of rectangles with equal area from G

Definition 2. Intersection Graph *of a set of geometric objects S, is defined as a graph $G = (S, E)$ in which there is an edge between two nodes $s, s' \in S$ if $s \cap s' \neq \emptyset$.*

We use the fact that every planar graph is an intersection of line segments in 2D space, e.g. we can put line segments corresponding to vertices of every planar graph such that two vertices are connected if and only if their corresponding line segments have an intersection. Construction of this equivalent graph can be done in polynomial time[7].

Let G be a planar graph whose maximum independent set is I and $|I| = k$. We make an instance of NOCC whose optimal solution has area ϵk for some constant ϵ. That concludes the reduction as having a polynomial algorithm for NOCC would reveal the size of I.

To do the reduction, we first transform G into an intersection of line segments using the polynomial construction in [7]. It's easy to make the construction such that no two line segments are co-linear and intersections does not happen on the end-points.

Assume s_i is the corresponding line segment to vertex i. Next, we replace s_i with a rectangle r_i whose area is ϵ (i.e. one edge length is $|s_i|$ and the other edge length is $\frac{\epsilon}{|s_i|}$, where $|s_i|$ is the length of s_i). ϵ is chosen so small that every two lines s_i and s_j intersect if and only if their corresponding rectanbles r_i and $_j$ intersect.

Finally, for every two pairs $i \neq j$, we put a black point on the line between every vertex of r_i and every vertex of r_j. These black points must not be placed inside any of the rectangles. This latter step ensures that only r_i's will be selected when solving the NOCC problem. The value of ϵ is chosen so small that it does not interfere with the construction, i.e. it's smaller than the distance of every two points in line segment intersection graph. The process of constructing the intersection graph of the rectangles is shown in Fig. 3.

How big is the area of the optimal solution in this NOCC instance? It's easy to see that the optimal solution must be a set of intersecting-free r_i's which means the optimal answer is $|I|\epsilon$. This completes the proof of the Theorem 1.

3 Approximation Algorithms

In this section we propose an approximation algorithm for NOCC. One can assume that the optimal solution is a set of non-intersecting triangles because every polygon in the optimal solution can be triangulated. So the problem reduces to the problem of finding the maximum weighted independent set in the intersection graph of all the black point-free triangles with white vertices whose weights are equal to their area. That is, for every triangle of three white points that does not contain any black point, we put a vertex with weight equal to the area of the trinagle. Two vertices are adjacent if their corresponding triangles interset in area.

Agarwal et al. [8] consider a non-weighted version of this problem for the intersection graphs of n convex shapes in 2D space. They proposed an approximation algorithm that computes a solution whose output is at least $\sqrt[3]{\frac{\alpha}{2\log(2n/\alpha)}}$, where α is the cardinality of the optimum solution. We have manipulated their algorithm to solve our problem. Let's first outline the algorithm of Agarwal et al. and then state our manipulations.

3.1 Non-weighted Version

The algorithm in [8] gets n convex shapes in 2D space as input and computes the maximum independent set of their intersection graph. Their algorithm has a divide and conquer approach and is outlined below(The algorithm is stated for a set of triangles Δ instead of polygons).

Algorithm 1. ApproxNOCC(Δ: a set of triangles)

```
Find a line $\ell$ which divides $\Delta$ into two almost equal cardinality sets.
$C_\ell$ = all the triangles that intersect $\ell$.
$L_\ell$ = all the triangles whose position is completely to the left of $\ell$.
$R_\ell$ = all the triangles that are completely positioned to the right of $\ell$.
Compute $\zeta(C_\ell)$ by a separate approximation algorithm
return $max\{ApproxNOCC(L_\ell) + ApproxNOCC(R_\ell), \zeta(C_\ell)\}$
```

Given a vertical line ℓ every triangle is either completely to the left of ℓ, completely to the right of ℓ, or intersects ℓ. Based on this we have three sets of triangles L_ℓ, R_ℓ and C_ℓ. By the way, ℓ is chosen in such a way that $||R_\ell| - |L_\ell||$ is minimized. For C_ℓ they provide a separate algorithm that computes a solution whose size is $\zeta(C_\ell)$, for some function ζ. Notice that this problem is different from and simpler than the original problem as all triangles in C_ℓ intersect the vertical line ℓ. At the end their algorithm returns the maximum of $\zeta(C_\ell)$ and their approximation algorithm solution on the reduced set $L_\ell \cup R_\ell$.

Let $\mu(\Delta)$ be the size of the solution returned by their algorithm. It's easy to prove that

$$\mu(\Delta) = \max\{\mu(L_\ell) + \mu(R_\ell), \zeta(C_\ell)\} \tag{1}$$

Given the above recursive equation, Agarwal et al. prove the following lower bound for $\mu(\Delta)$.

$$\mu(\Delta) \geq \zeta(\frac{OPT}{2log(2t/OPT)}) \tag{2}$$

where $t = |\Delta|$ and OPT is the size of the optimal solution. For C_ℓ they propose a dynamic programming based algorithm that computes a solution that has a cubic approximation factor:

$$\zeta(C_\ell) \geq \sqrt[3]{OPT}$$

where OPT is the size of the optimal solution on C_ℓ. This yields the above $\sqrt[3]{\frac{\alpha}{2log(2n/\alpha)}}$ approximation factor.

3.2 Our Modification

Our algorithm has the same structure except that our triangles are weighted and consequently, our approximation algorithm for the maximum independent set on C_ℓ is different as explained below.

First we need the following useful lemma for our construction.

Lemma 1. *Suppose that we are given a set I of weighted objects such that for each $x \in I$ we have $w(x) \geq 1$. Assume a partial order $\preceq$ over the members of I is given. Let $S = \sum_{x \in I} w(x)$. Then there exists a chain or anti-chain over the members of I whose total weight is at least $\sqrt{S}$.*

Proof. Let $\mathcal{A}$ be a maximum cardinality anti-chain and $\mathcal{C}$ be a partition of objects into minimum number of chains. By Dilworth's theorem [9], the cardinality of $\mathcal{A}$ equals the number of chains in $\mathcal{C}$.

If $\mathcal{A}$ has cardinality more than $\sqrt{S}$ then we are done as every object has weight more than one. Otherwise, $\mathcal{C}$ has at most $\sqrt{S}$ chains which means the heaviest chain must have total weight at least $\frac{S}{\sqrt{S}} = \sqrt{S}$.

Let ℓ be a vertical line. Like before, let C_ℓ be all the triangles that intersect ℓ. Suppose $OPT(C_\ell)$ is the total area of all the triangles in the maximum weighted independent set of C_ℓ. For each $s_i \in C_\ell$ define $r(s_i)$(resp. $l(s_i)$) to be the x-coordinate of the rightmost (resp. leftmost) point in s_i. Also define $c(s_i)$ to be the maximum y-coordinate of the intersection of s_i with line ℓ. The following lemma is obtained from [8] by minor modifications.

Lemma 2. *There exists a sequence $I = \prec s_{i_1}, s_{i_2}, ..., s_{i_m} \succ$ of the members of C_ℓ such that $\sum_{j=1}^{m} w(s_{i_j}) \geq \sqrt[4]{|\sum_{x \in C_\ell} w(x)|}$, where $w(x)$ is the area of the triangle x and I is a chain according to one of the following order conditions:*

- *$O_1(s_i, s_j)$: $r(s_i) < r(s_j)$ and $c(s_i) < c(s_j)$.*
- *$O_2(s_i, s_j)$: $r(s_i) < r(s_j)$ and $c(s_i) > c(s_j)$.*
- *$O_3(s_i, s_j)$: $r(s_i) > r(s_j)$.*

Proof. Let $S_l = |\sum_{x \in C_\ell} w(x)|$. First we apply lemma 1 on C_ℓ with partially order defined by O_1. So, there must be a chain or anti-chain D_1 with total weight at least $\sqrt{S_l}$. If it's a chain then we are done. Otherwise, apply lemma 1 to D_1 on partially order defined by O_2. According to this lemma, there exist a chain or anti-chain D_2 whose total weight is at least $\sqrt{\sqrt{S_l}} = \sqrt[4]{S_l}$. If it's a chain then it satisfies condition O_2. Otherwise, as it is an anti-chain by both O_1 and O_2, it must satisfy O_3.

The interesting fact is that the optimal answer that satisfies either of O_i's can be computed in polynomial time by a dynamic-programming algorithm.

For each $x \in \{1, 2, 3\}$, we shall compute the sequence with maximum total area that satisfies O_x. Let $s_1, \cdots, s_p$ be a topological ordering based on the order O_x on C_ℓ. Let i, j be two vertices such that $i < j$ and $s_i \cap s_j = \emptyset$. Define $\phi^x(i, j)$ to be the size of the maximum total area sequence from the set $\{s_i, s_{i+1}, \cdots, s_j\}$ which satisfies O_x. $\phi^x(i, j)$ equals

$$max_{\substack{i \leq k \leq j \\ s_k \cap s_i = \emptyset \\ s_k \cap s_j = \emptyset}} \phi^x(i, k-1) + \phi^x(k+1, j) + w(s_k)$$

which means one can use dynamic programming and compute ϕ^x in polynomial time. At the end, $\zeta(C_\ell))$ will be the maximum of three values,

$$\zeta(C_\ell) = max\{\phi^1(1, p), \phi^2(1, p), \phi^3(1, p)\} \geq \sqrt[4]{\sum_{x \in C_\ell} w(x)} \geq \sqrt[4]{OPT(C_\ell)}$$

Given the above lower bound for $\zeta(C_\ell)$ we plug it into Equation 2 and obtain a lower bound for the approximation factor of our algorithm.

$$\mu(\Delta) \geq \sqrt[4]{\frac{OPT}{2log(2t/OPT)}} \geq \sqrt[4]{\frac{OPT}{2log(2n/OPT) + 2log(n)}}$$

The last inequality follows from the fact that $t \leq \binom{n}{3}$.

The algorithm is outlined in Algorithm 1. The input of this algorithm is the set of all triangles of white vertices which contain no black points (Δ). This completes the proof of Theorem 2.

3.3 TDCC

The idea for approximating TDCC is similar to that of NOCC with some modifications. In NOCC we could safely assume that the optimal answer is a set of triangles and then found an approximate weighted independent set on the set of feasible triangles. This assumption, however, does not hold for the case of TDCC as a polygon cannot necessarily be partitioned into a set of totally disjoint triangles.

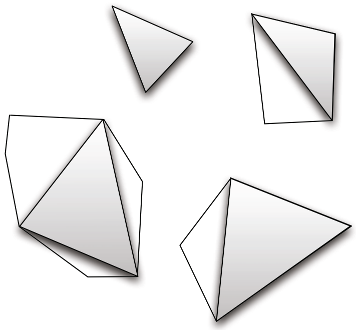


Fig. 4. Inscribed triangles with area of $\frac{3\sqrt{3}}{4\pi}$. OPT in TDCC problem.

To overcome this difficulty, we use the following useful lemma.

Lemma 3. *([10]) Let B be a compact convex body in the plane and B_k be the largest area k-edge polygon inscribed in B. Then $area(B_k) \geq area(B).\frac{k}{2\pi}sin\frac{2\pi}{k}$, where equality holds if and only if B is an ellipse.*

This means if we only focus on triangles for TDCC then the optimal solution would be less by no more than a factor of $\frac{3}{2\pi}sin\frac{2\pi}{3} = \frac{3\sqrt{3}}{4\pi}$.

Therefore, we can run the same algorithm as the one for NOCC (with the change that two triangles are considered connected even if they share an edge or a vertex) and obtain a solution whose total area is at least

$$\frac{3\sqrt{3}}{4\pi}\sqrt[4]{\frac{OPT}{2log(2n/OPT) + 2log(n)}}$$

This completes the proof of Theorem 3.

Algorithm 2. GreedyTDCC

```
$\mathcal{C} \leftarrow \emptyset$
$C$ = maximum convex hull of white points by the Fischer's algorithm[4]
**while** *The area of $C$ is greater than 0* **do**
    $\mathcal{C} \leftarrow \mathcal{C} \cup C$
    Remove all the white points inside $C$
    Replace $C$'s vertices with black points
    **foreach** *pair $(A, B)$ of points which coincide on the outside or boundary of $C$* **do**
        **if** *segment $\overline{AB}$ has intersection with $C$* **then**
            Add a dummy black point on $\overline{AB}$ which resides inside $C$.
        **end**
    **end**
    Add a dummy black point inside $C$.
    $C$ = maximum convex hull of white points.
**end**
**return** $\mathcal{C}$
```

3.4 A Note on Measurement Units and the Root Function

It's clear that if we change our measurement unit(e.g. use meter instead of inch) then it affects our approximation factor which is a root function. To avoid this problem the area of triangles (and hence, the wight of vertices) are computed relative to the smallest triangle. Thus, all weights are at least one (which is needed for Lemma 1) and our approximation factor would not change by scaling.

4 Heuristic Approach

The algorithm of Fischer[4] for finding one single region with maximum area can be easily transformed into a greedy algorithm for our problem by repeatedly finding a region and deleting it.

In this section, we show that while this algorithm behaves well for random data, it can be arbitrarily bad in the worst case. First we explain the algorithm in detail and then provide experimental results. Finally we prove a lower bound for its behavior in the worst case. We explain the algorithm and experimental results for TDCC. The results for NOCC are essentially similar but they are more complex and need more details and is left to the journal version of our work.

4.1 Algorithm Description

The algorithm works as follows. It iteratively computes a maximum area convex hull of white regions using Fischer's algorithm. In order to prevent intersection, it puts enough black points around and inside the selected convex hulls to prevent future convex hulls to overlap existing ones.

The details of the algorithms is depicted in Algorithm 2.

Once a convex hull C is chosen, for any line segment AB of two white points A and B that intersect with C, the algorithm puts an arbitrary black point on the intersection of AB and the interior of C. That prevents line segment AB to be chosen as part of any future convex hull. It also replaces C's vertices with black points so as to prevent future convex hulls to have intersection with C's vertices.

4.2 Analysis

As we see shall see shortly this greedy algorithm could work arbitrarily bad, however, our experiments show its good behavior for random data.

First we build an example on which the solution of GreedyTDCC and the optimal differ by a factor of $\Omega(n)$. Our worst case example is depicted in Fig. 5.

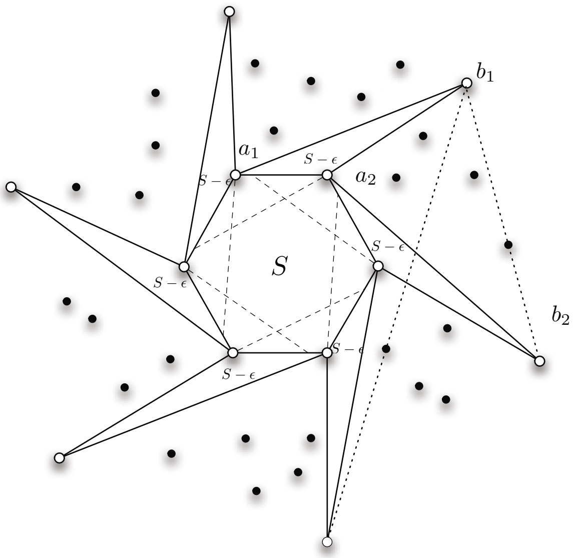


Fig. 5. Worst case example of GreedyTDCC

Let C be an n-regular polygon with area S and vertices $a_1, a_2, \cdots, a_n$. For each $1 \leq i < n$ we create a vertex b_i outside C such that the triangle $b_i a_i a_{i+1}$ has area $S - \epsilon$ (for an arbitrarily small ϵ). Assume $a_1 = a_{n+1}$ and $n \overset{2}{\equiv} 0$ for convenience.

Our aim is to let only C and triangles of the form $b_i a_i a_{i+1}$ be chosen in any solution. Therefore we only allow the following three types of line segments: $a_i a_{i+1}, b_i a_i$, and $b_i a_{i+1}$. For any other line segment we put a black point on it in a suitable place (i.e. is not contained in any of the triangle $b_i a_i a_{i+1}$) to prevent it from being chosen.

In this instance the GreedyTDCC algorithm will pick C as the first convex hull and then it can not find any other convex hull with area more than 0. So the area of the output of GreedyTDCC is S. However, the optimal answer will pick all the $\frac{n}{2}$ triangles of type $b_{2k}a_{2k}a_{2k+1}$, so $OPT = \frac{n}{2}.(S - \epsilon)$. This completes our claim that GreedyTDCC works arbitrarily bad in the worst case.

Theorem 5. *There is an instance L of n points such that*

$$\frac{OPT(L)}{GreedyTDCC(L)} \in \Omega\,(n)$$

where $GreedyTDCC(L)$ is the output of our greedy algorithm on the instance L and $OPT(L)$ is the optimal answer.

The algorithm, however, works well on random data. For this, we implemented the GreedyTDCC algorithm and tested it on seven different random distribution of the points. For each case, e.g. RAN1, we created 10 different random instances and computed the average and the worst (least) ratio of our algorithm to the optimal algorithm. The area of the optimal covering is obtained via a simple exhaustive search algorithm. The results are shown in table 1). The experimental result suggests that the output of GreedyTDCC algorithm is a good approximation of the optimal answer in practice.

Table 1. The ratio of GreedyTDCC's solution to optimal

Test Data	**#White Points**	**#Black Points**	**Worst case Ratio**	**Average Ratio**
RAN_1	10	3	0.64	0.7
RAN_2	12	12	0.49	0.66
RAN_3	16	10	0.48	0.59
RAN_4	14	20	0.41	0.72
RAN_5	15	16	0.6	0.7
RAN_6	20	40	0.56	0.68
RAN_7	20	20	0.48	0.63

5 Conclusion and Further Works

We considered three different versions of the problem of covering a set of white points without touching black points . We solved one and found hardness result as well as approximation algorithms for the other two.

The approximation factor of our algorithms are dependent to the result of Lemma 1. Any improvement in that lemma would improve the approximation factor. One may hope to find a bound like $\frac{S}{\sqrt{n}}$ (which is identical to Dilworth's bound for uniform weights), but we haven't been able to achieve it.

There are many variants of the problem that are worth studying. One particular problem that we are interested in is the weighted version of our problem. Black points have weights between zero and one and the aim is to find a set of convex hulls around white points so that the total weight of black points inside convex hulls is less than a threshold.

Acknowledgment
The authors are thankful to Abbas Mehrabian for his many useful comments throughout the preparation of this paper.

References

1. Federal Communications Commission, Notice of proposed rule making: Unlicensed operation in the TV broadcast bands. ET Docket No. 04-186 (FCC 04-113) (May 2004)
2. Cardoso, L.S., Debbah, M., Bianchi, P., Najim, J.: Cooperative spectrum sensing using random matrix theory. In: IEEE ISWPC, pp. 334–338 (May 2008)
3. Abachi, T., Hemmatyar, M.A., Fazli, M.A., Izadi, M.: Centralized Spectrum Sensing Using New Algorithmic Techniques. In: CRNet 2010, Bijing, China, August 25-27 (2010)
4. Fischer, P.: Finding Maximum Convex Polygons. In: Proceedings of the 9th International Symposium on Fundamentals of Computation Theory, pp. 234–243 (1993)
5. Fischer, P.: More or less efficient agnostic learning of convex polygons. In: Proceedings of the Eighth Annual Conference on Computational Learning Theory, pp. 337–344 (1995)
6. Veni Madhavan, C.E.: Approximation algorithm for maximum independent set in planar traingle-free graphs. In: Joseph, M., Shyamasundar, R.K. (eds.) FSTTCS 1984. LNCS, vol. 181, pp. 381–392. Springer, Heidelberg (1984)
7. de Castro, N., Cobos, F.J., Dana, J.C., Marquez, A., Noy, M.: Triangle-free planar graphs as segment intersection graphs. J. Graph Algorithms Appl. 6(1), 7–26 (2002) (electronic); Graph drawing and representations (Prague, 1999)
8. Agarwal, P.K., Mustafa, N.H.: Independent Set of Intersection Graphs of Convex Objects in 2D. In: Computational Geometry: Theory and Applications, pp. 83–95. Elsevier Science Publishers, Amsterdam (2006)
9. Dilworth, P.: A decomposition thorem for partially ordered sets. Annals of Mathematics 51, 161–166 (1950)
10. Sás, E.: On a certain extremum-property of the ellipse (in Hungarian, English summary). Mat. Fiz. Lapok 48, 533–542 (1941)

New Results on the Complexity of the Max- and Min-Rep Problems

Robert Ganian*

Faculty of Informatics, Masaryk University
Botanická 68a, Brno, Czech Republic
ganian@mail.muni.cz

Abstract. This paper deals with the Max-Rep and Min-Rep problems, both of which are related to the famous Label Cover problem. These are of notable theoretical interest, since they are often used to prove hardness results for other problems. In many cases new complexity results for these problems may be preserved by the reductions, and so new results for Max-Rep and Min-Rep could be applicable to a wide range of other problems as well.

Both Max- and Min-Rep are strongly inapproximable, and the best approximation algorithms have a ratio of $O(n^{1/3})$ and $O(n^{1/3}\log^{2/3} n)$ respectively. Thus, other approaches are desperately needed to tackle these hard problems. In our paper we use the very successful parameterized complexity paradigm and obtain new complexity results for various parameterizations of the problems.

1 Introduction

Both Max-Rep and Min-Rep are natural problems first introduced by Kortsarz [14] related to the famous Label Cover problem of Arora, Babai, Stern and Sweedyk [3]. Indeed, Max-Rep can be expressed as a maximization variant of Label Cover, however Min-Rep is slightly different from its minimization variant. Both of these problems are often used in hardness reductions, which provides strong motivation for obtaining new complexity results for these two problems.

The reductions from Min-Rep and Max-Rep to other problems include Directed Steiner Forest [9], l-Round Power Dominating Set [2], Min-Power k-Edge-Disjoint Paths [12], Red-Blue Set Cover [17], Set Cover with Pairs [13], Sparsest k-Transitive-Closure-Spanner [5], Stochastic Steiner Tree with Non-uniform Inflation [11], Target Set Selection [7] and Vertex Connectivity Survivable Network Design [15].

Previous studies of Min-Rep and Max-Rep focused mainly on their approximability and on approximation algorithms for these problems. Kortsarz proved that both are strongly inapproximable [14], and only recently Charikar, Hajiaghayi and Karloff [6] managed to improve the ratio from $O(n^{1/2})$ to $O(n^{1/3})$

* This research has been supported by the Czech research grants 201/09/J021 and MUNI/E/0059/2009.

for Max-Rep and $O(n^{1/3}log^{2/3}n)$ for Min-Rep. In light of these (fairly discouraging) results, one has to ask whether it would be possible to tackle these two problems by using some other approach than approximation.

In our article, we study Max-Rep and Min-Rep from a parameterized-algorithmics point of view and obtain several new results by various parameterizations of these problems. In this case negative results are of particular interest – hardness results may in the future be used to immediately obtain hardness results for other problems by reductions. We show that both Max-Rep and Min-Rep admit XP-time algorithms when parameterized by the number of sets, that both are $W[1]$-hard with respect to this parameter and that FPT algorithms are possible for both when the structure of the graph is restricted by popular width measures (see section 2 for an explanation of these terms). The final part of the article provides separate hardness proofs for Max-Rep and Min-Rep when parameterizing by the size of sets, and shows that Max-Rep remains hard even when additional very restrictive constraints are present.

2 Definitions

Definition 2.1 (Max-Rep)
Instance: *A bipartite graph* $G = (A, B, E)$ *with* $n = |A| = |B| = m \cdot k$ *and* $2m$ *cardinality-k disjoint subsets* $A_1 \dots A_m$, $B_1 \dots B_m$ *such that* $A_1 \cup A_2 \cup \dots \cup A_m = A$ *and* $B_1 \cup B_2 \cup \dots \cup B_m = B$.

Objective: *Select sets of representatives* $A' \subseteq A$, $B' \subseteq B$ *such that* $|A' \cap A_i| = 1$, $|B' \cap B_i| = 1$ *for all* $1 \leq i \leq m$ *and the subgraph induced by* $A' \cup B'$ *has the maximum number of edges.*

Another view of the Max-Rep problem is to consider a bipartite "supergraph" $\mathcal{H}$ with $2m$ vertices, the vertices of $\mathcal{H}$ being the sets A_i and B_j. A_i and B_j are then adjacent in $\mathcal{H}$ iff there exist $a_i \in A_i$ and $b_j \in B_j$ such that $(a_i, b_j) \in E(G)$ (in this case we say that (a_i, b_j) covers the superedge (A_i, B_j)). In this case, the goal of Max-Rep is to select a single representative for each of the sets A_i, B_j in a way maximizing the edges of $\mathcal{H}$.

Definition 2.2 (Min-Rep)
Instance: *A bipartite graph* $G = (A, B, E)$ *with* $n = |A| = |B| = m \cdot k$ *and* $2m$ *cardinality-k disjoint subsets* $A_1 \dots A_m$, $B_1 \dots B_m$ *such that* $A_1 \cup A_2 \cup \dots \cup A_m = A$ *and* $B_1 \cup B_2 \cup \dots \cup B_m = B$.

Objective: *Select sets* $A' \subseteq A$, $B' \subseteq B$ *such that each superedge of* $\mathcal{H}$ *is covered by some edge* (a, b) *of* G *where* $a \in A'$, $b \in B'$. *The goal is to minimize* $|A'| + |B'|$.

It is important to notice that although both problems work with the same input, their objectives differ significantly. While in Max-Rep one is restricted to selecting only a single representative for each subset of the input with a goal of maximizing superedges in $\mathcal{H}$, in Min-Rep the requirement is to actually cover all the superedges of $\mathcal{H}$ while minimizing the number of selected vertices in $A \cup B$.

Finally, we provide some basic terms used in parameterized complexity theory. A parameter (sometimes referred to as a width or structural parameter) is a function $f : I \to N$ where I is the input for our problem. The idea is to select a parameter which is low on some large and interesting instances of the given problem, and then design parameterized algorithms which run in polynomial time when the parameter is bounded. A good example of a parameter used for many problems is the tree-width of a graph, which is low on graphs that are structurally similar to trees and allows the design of efficient parameterized algorithms for many problems. It is also often the case that the input of the problem itself provides a natural choice for the parameter – e.g. when computing the minimum dominating set of a graph we may take its size as the parameter.

With respect to parameterized algorithms, we distinguish between those which run in time (considerng a parameter k and an input of size n) $O(f(k) \cdot poly(n))$ and those with a runtime of $O(poly(n)^{f(k)})$. Algorithms of the first type are called Fixed-Parameter Tractable (FPT in short) since the exponent of the polynomial in n does not grow with the parameter. On the other hand, algorithms of the second type are called XP algorithms. An FPT algorithm is typically much more practical than an XP one, however having an XP algorithm is still preferable to exponential algorithms when the parameter is bounded.

3 New Results

The problem definitions themselves allow two natural approaches to parameterizing the input: either selecting m as the parameter, thus restricting the number of subsets, or bounding k, which leads to a restriction on the size of the subsets. We will show that while one case leads to relatively positive results, the other remains NP-hard even under very restricted circumstances.

3.1 Parameterization by m

If we set m as the parameter, it is a trivial observation that an XP algorithm exists for computing Max-Rep. The algorithm simply runs through all $\frac{n}{m}$ choices of selecting representatives for each of the m subsets in A and B, and then for each choice counts the number of covered edges.

Corollary 3.1. *There exists an algorithm computing Max-Rep in time* $(\frac{n}{m})^{2m} \cdot n^2$.

For Min-Rep, an XP algorithm exists as well, however the situation is slightly more complicated. Since it is now admissible to select several representatives for each subset, the previous approach will not work directly. On the other hand, there may be at most $\frac{n^2}{m^2}$ edges between any two chosen subsets A_i, B_j and we know that for each adjacent pair of sets it is sufficient to have a single edge cover the superedge between them. So, it is possible to run through all $\frac{n^2}{m^2}$ choices of edges between any pair of subsets (with m^2 pairs in total) and select A', B' as those vertices which are incident to the chosen edges. In this way, we obtain the optimal choices of A', B' for each pre-selected edge set, and all that remains is to compute the cardinality of $A' \cup B'$.

Corollary 3.2. *There exists an algorithm computing Min-Rep in time* $O((\frac{n^2}{m^2})^{m^2} \cdot n)$.

In light of the negative approximability results on these problems, any new exact algorithms may actually be considered good news. Still, the running time of XP algorithms typically grows very quickly with the parameter and so in practical applications it is usually only possible to use them for very low values of the parameter. So, could these parameterized problems be solved in FPT time? Unfortunately, the answer is no – we prove that they are $W[1]$-hard, which means that they do not admit FPT algorithms unless the exponential time hypothesis fails.

Theorem 3.3. *Max-Rep is* $W[1]$*-hard when parameterized by* m.

Proof. We prove the theorem by reduction from the well-known $W[1]$-hard independent set problem [8]. Given is a graph $G = (V, E)$ and a parameter d, and the question is whether there exists an independent set of size d in G. We construct our Max-Rep instance as follows:

For all $1 \leq i \leq m = 3d$, $|A_i| = |B_i| = |V|$ and there exists a bijection j_i^A between each A_i and V and j_i^B between each B_i and V (so that in each set there is a vertex corresponding to any vertex in G). Then for all $1 \leq g, h \leq d$ we add edges between each $a \in A_g$ and $b \in B_h$ iff $(j_g^A(a), j_h^B(b)) \notin E(G)$. At this point, notice that if we could somehow ensure that:

1. Max-Rep choses vertices in A_i and B_i which correspond to the same vertices in G, and
2. No two sets $A_z \neq A_x$ have selected the same vertex in G,

an optimal solution of Max-Rep with d^2 edges would represent an independent set in G, while the non-existence of one would on the other hand indicate that no independent set of size d exists in G.

We will use A_q, B_q with $d + 1 \leq q \leq 2d$ to fulfill point 1. For each A_i, B_i, $1 \leq i \leq d$, and for each $a \in A_i, b \in B_i$ s.t. $j_i^A(a) = j_i^B(b)$, we create edges (a, b'), (b, a'), (a', b'), where $a' \in A_{i+d}$, $b' \in B_{i+d}$ and $j_{i+d}^A(a') = j_{i+d}^B(b') = j_i^B(b)$. Informally, we have created "links" between a and b in A_i, B_i corresponding to the same vertex. These links may increase the optimal solution of Max-Rep by $3d$, but only if for each i we choose vertices in A_i, B_i which correspond to the same vertex in G.

Next, we will use A_p, B_p with $2d + 1 \leq q \leq 3d$ to fulfill point 2, i.e. to ensure that no vertex of G is selected multiple times in A_i (or B_i), $1 \leq i \leq d$. For every $a \in A_i$ we create an edge between a and $b'' \in B_{i+2d}$ which corresponds to the same vertex in G. Then, for each $b'' \in B_{i+2d}$ we add edges to all elements of $A_j, 1 \leq j \neq i \leq d$ such that they correspond to a *different* vertex of G than b''. For $a'' \in A_i$ we then do the same as for $b'' \in B_i$, by symmetry. In total, this may increase the optimal solution of Max-Rep by $2d^2$, but only if each vertex in G appears at most once in A' and at most once in B'.

To summarize: in our instance of Max-Rep, an optimal solution of value $3d^2 + 3d$ exists iff there exist two k-selections of vertices in G (a) which are identical, (b) where no vertex occurs more than once in each selection, and (c) which are pairwise independent; however this is actually only a complicated way of saying that G contains an independent set of cardinality k. ∎

Theorem 3.4. *Min-Rep is $W[1]$-hard when parameterized by m.*

Proof. The proof utilizes the same reduction as in Theorem 3.3. The number of superedges in $\mathcal{H}$ in the reduction is $3d^2 + 3d$:

1. The number of superedges between $A_1 \dots A_d$ and $B_1 \dots B_d$ is d^2.
2. The number of superedges between $A_1 \dots A_d$ and $B_{2d+1} \dots B_{3d}$ and between $A_{2d+1} \dots A_{3d}$ and $B_1 \dots B_d$ is $2 \cdot d^2$.
3. Each of $\{(A_i, B_{(d+i)}), (B_{(d+i)}, A_{(d+i)}), (A_{(d+i)}, B_i)\}$ constitutes 3 new superedges in $\mathcal{H}$.
4. No other superedges are present in $\mathcal{H}$.

Since we already know that the bound of $3d^2 + 3d$ superedges is attainable by selecting $6d$ vertices iff I is independent, this concludes our proof. ∎

Before we move forward to parameterization by k, it is an interesting question to ask whether these results could be improved by adding an additional graph-structural parameter constraint to the input graphs for Max-Rep and Min-Rep. Indeed, in both cases the result turns out to be positive.

Theorem 3.5. *There exists an algorithm computing Max-Rep in FPT time parameterized by the tree-width of the input graph and m.*

Proof. It suffices to show that the problem may be captured by a so-called *LinEMSOL* formula. It is a known result that on graphs of bounded tree-width it is possible to compute *LinEMSOL* optimization problems in FPT time [1]. The formula for Max-Rep has the following form:

$$Max\, E' : \exists A', B' : \forall_{1 \leq i,j \leq m} A_i, B_j : |A_i \cap A'| = 1 \wedge |B_j \cap B'| = 1 \wedge (\forall e \in E' : \exists a' \in A', b' \in B' : inc(e, a') \wedge inc(e, b'))$$

The formula finds two vertex sets A' and B' which fulfill the listed conditions such that the number of edges in $|E'|$ is maximized, and all edges in E' need to be incident to a vertex in A' and B'. ∎

Theorem 3.6. *There exists an algorithm computing Min-Rep in FPT time parameterized by the rank-width of the input graph and m.*

Proof. The proof is similar to the one for Max-Rep. However, here it is possible to generalize the result from tree-width to the much less restrictive rank-width parameter [16]. Rank-width, like tree-width, allows efficient solution of *LinEMSOL*-expressible problems [10], however only with respect to the less powerful MSO_1 language which does not allow variables to represent edges or edge sets. The formula for Min-Rep has the following form:

$$Min\, V' : \forall_{1 \leq i,j \leq m} A_i, B_j : (\exists a_i \in A_i, b_j \in B_j : edge(a_i, b_j)) \implies (\exists a'_i \in A_i \cap V', b'_j \in B_j \cap V' : edge(a'_i, b'_j))$$

3.2 Parameterization by *k*

Parameterization of Max-Rep and Min-Rep yielded, to a certain extent, at least some positive results. The next natural step is to try a different approach – instead of fixing the number of sets as the parameter, we parameterize their size k. Both Max-Rep and Min-Rep become trivial for $k = 1$, so the first meaningful value of k is 2.

Surprisingly, things get much worse now. The key result is that both Max-Rep and Min-Rep remain NP-hard even for $k = 2$. For Max-Rep, the problem remains hard even when restricted to forests with paths of length at most 2 and degree at most 3. It is interesting that for this problem restricting the structure of the graph helped significantly when parameterizing by m but does not help even for very low values of k.

Theorem 3.7. *Max-Rep is NP-hard to compute when $k \geq 2$, even if G is a forest with maximum path length* 2 *and maximum degree* 3*.*

Proof. We prove hardness by reduction from the NP-hard E3OCC-MAX-2SAT problem, which is even hard to approximate within some constant factor [4]. Given is a 2SAT instance where each variable occurs exactly three times, and the goal is to evaluate the variables so that the number of satisfied clauses is maximized.

The first part of our reduction gadget is simple – for each clause c we have a set $B_c = \{c1, c2\}$. We may safely assume that each variable occurs twice positively and once negatively or twice negatively and once positively. Now, say we had some vertices of A represent literals and each was connected to the clause it belonged to by a single edge so that each clause vertex had exactly one neighbour. Then, if we could somehow ensure that Max-Rep would only select positive or negative values for each variable, solving it would immediately translate to an optimal solution solution for E3OCC-MAX-2SAT; indeed, each edge in the optimal solution leads to a unique clause which is satisfied by the valuation found by Max-Rep.

We will construct our gadget with this idea in mind. For each variable x with two positive and one negative occurrences, we have $A_{x1} = \{x_1^t, x_1^f\}$ and $A_{x2} = \{x_2^t, x_2^\emptyset\}$. x_1^t, x_1^f and x_2^t will be adjacent to an arbitrary clause vertex of the clause they occur in (where f represents false and t true), while keeping each clause vertex of degree 1.

To make sure it is never optimal to choose x_1^f and x_2^t in the solution, we add two special sets $B_{xp} = \{x_{tp}, x_{fp}\}$ and $B_{xq} = \{x_{tq}, x_{fq}\}$ and make x_{tp}, x_{tq} adjacent to x_1^t and x_{fp}, x_{fq} adjacent to $x_2^\emptyset$ (see Figure 1 for an illustration).

Consider what would happen if A' contained both x_1^f and x_2^t. This choice of A' for A_{x1} and A_{x2} could at best induce two superedges in $\mathcal{H}$ – both adjacent to clause vertices. On the other hand, for example a choice of x_1^f and $x_2^\emptyset$ certainly induces three superedges in $\mathcal{H}$ thanks to x_{fp} and x_{fq}, and so an optimal solution for Max-Rep will never contain an invalid choice of SAT variables.

For formal reasons, it is necessary to add some degree 0 vertices to make sure $|A| = |B|$, however otherwise we are done. Given the optimal solution

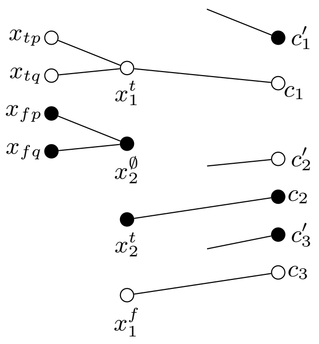


Fig. 1. An illustration of the Max-Rep reduction gadget for variable x

for such a Max-Rep instance, it suffices to subtract $2 \cdot |var|$ (var being the set of variables) and the result is the maximum number of obtainable edges between clause vertices and variable vertices. Since each clause may only add a single edge to the sum and any optimal choice of variable vertices translate to proper valuations, this number exactly equals the maximum number of satisfiable clauses in the E3OCC-MAX-2SAT instance. ∎

At this moment the distinction between Max-Rep and Min-Rep becomes more apparent. While the reduction in the previous section was applicable to both Min-Rep and Max-Rep, here there is no simple way of translating the proof to Min-Rep and a full separate proof is required.

Theorem 3.8. *Min-Rep is NP-hard to compute even when $k \geq 2$.*

Proof. The proof is again based on a reduction from E3OCC-MAX-2SAT. Notice that in any optimal solution to an E3OCC-MAX-2SAT instance each variable appears in at most one unsatisfied clause (otherwise its value could be switched to obtain a better solution). Thus, solving E3OCC-MAX-2SAT is actually equivalent to being allowed to select *universal* variables which will "cheat" and fulfill every clause they appear in, with the goal of minimizing the number of universal variables.

For the reduction itself, assume that the input is an E3OCC-MAX-2SAT instance with n variables. A and B are completely symmetric in the reduction gadget, so we will only explicitly describe A. For each literal z we create n sets $A_z^i = \{z_t^{A,i}, z_f^{A,i}\} \in A$, $1 \leq i \leq n$. Next, edges are added between all $z_t^{A,i}$ and $z_t^{B,j}$ (*true vertices*) and between all $z_f^{A,i}$ and $z_f^{B,j}$ (*false vertices*), $1 \leq i, j \leq n$. Now for each variable x we create an additional *control set* $A_x = \{x_t^A, x_f^A\}$ with edges from x_t^A to all the true vertices of x in B and from x_f^A to all the false vertices of x in B. An optimal Min-Rep solution at this moment would be $6n^2 + 2n$ and is obtained by selecting either only true or only false vertices for each variable (see Fig. 2 for an illustration).

Next, for each pair of literals which form a clause edges are added based on their positive/negative signs in the clause so that the only non-adjacent vertices

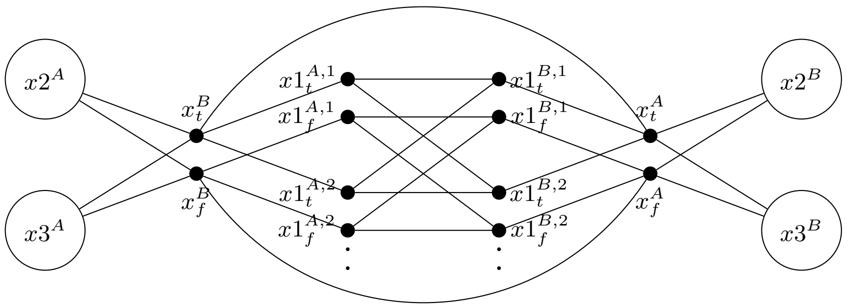


Fig. 2. Min-Rep reduction gadget for variable x, appearing as three literals $x1, x2, x3$

are those with opposite signs than in the clause (i.e. assignments which satisfy the clause are adjacent). For instance, if $x2$ and $\neg y3$ form a clause then the created edges are $1 \leq i, j \leq n : (x2_t^{A,i}, y3_f^{B,j})$, $(x2_f^{A,i}, y3_f^{B,j})$, $(x2_t^{A,i}, y3_t^{B,j})$ and their symmeric copies.

If the input 2SAT instance is satisfiable, Min-Rep will still output $6n^2 + 2n$ since it is possible to select only true or false vertices for each variable and still cover all superedges – those belonging to the variable gadgets are covered simply by selecting all vertices as true or false, and those belonging to clauses are covered since all clauses are satisfied. On the other hand, if the instance is not satisfiable, then it is not possible to cover all the clause superedges in this way and in some sets both vertices are selected. The goal is to show that the minimum number of unsatisfied clauses (or, equivalently, universal variables) is $\frac{k-(6n^2+2n)}{2}$ where k is the optimal solution of Min-Rep.

First assume that the optimal solution only contains double selections of vertices from control sets. It is easy to see that any symmetric double selection in A and B then allows the variable to independently change the truth assignment of one of its literals, and so the number of double selections (obtained by subtracting the number of sets, $6n^2 + 2n$) is equal to twice the number of universal variables.

On the other hand, assume that an optimal solution contains at least one double selection of vertices from a non-control set, say some $x3_t^{A,i}$ and $x3_f^{A,i}$. If the control set of x is double-selected then it is possible to satisfy all superedges adjacent to $x3^{A,i}$ without any additional double selections, contradicting the optimality of our solution. Yet if the control set of x is not double-selected then to satisfy all the superedges between $x3$ vertices and the control set it is necessary for all $x3$ sets to have the corresponding truth assignment selected. But then the double selection of $x3^{A,i}$ is either unnecessary or it satisfies some superedges which – due to symmetry - also need to be satisfied for all the $n-1$ other sets of $x3$, which contradicts the optimality of our solution in comparison with a solution which only double selects control sets. ∎

4 Conclusions

We have provided several new results on the parameterized complexity of the theoretically important Max-Rep and Min-Rep problems. These problems are of particular interest since they are often used in hardness proofs, and the provided results may help future authors obtain stronger results when using reductions to Max-Rep and Min-Rep.

Furthermore, our hardness proofs illustrate the importance of using the right parameter in algorithms: while using m as the parameter leads to some positive results (particularly if the graph has bounded tree-width), parameterizing by k in fact does not help at all and it does not seem that this could be improved by restricting the structure of the graph. Future research could be aimed at determining the hardness of Min-Rep on graphs with structural restrictions.

References

1. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. J. Algorithms 12(2), 308–340 (1991)
2. Aszami, A., Stilp, M.D.: Approximation algorithms and hardness for dominating with propagation. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) RANDOM 2007 and APPROX 2007. LNCS, vol. 4627, pp. 1–15. Springer, Heidelberg (2007)
3. Arora, S., Babai, L., Stern, J., Sweedyk, Z.: The hardness of approximate optima in lattices, codes and systems of linear equations. J. Comput. System Sci. 54, 317–331 (1997)
4. Berman, P., Karpinski, M.: Improved Approximation Lower Bounds on Small Occurrence Optimization. Electronic Colloquium on Computational Complexity, TR Report 03-008 (2003)
5. Bhattacharyya, A., Grigorescu, E., Jung, K., Raskhodnikova, S., Woodruff, D.P.: Transitive-Closure Spanners. In: SODA 2009, pp. 531–540 (2009)
6. Charikar, M., Hajiaghayi, M., Karloff, H.: Improved Approximation Algorithms for Label Cover Problems. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 23–34. Springer, Heidelberg (2009)
7. Chen, N.: On the approximability of influence in social networks. In: SODA 2008, pp. 1029–1037 (2008)
8. Downey, R.G., Fellows, M.R.: Parameterized complexity (Monographs in Computer Science). Springer, Heidelberg (1999)
9. Feldman, M., Kortsarz, G., Nutov, Z.: Improved approximation for the directed steiner forest problem. In: SODA 2009, pp. 922–931 (2009)
10. Ganian, R., Hliněný, P.: On Parse Trees and Myhill-Nerode-type Tools for handling Graphs of Bounded Rank-width. Discrete Applied Mathematics 158, 851–867 (2010)
11. Gupta, A., Hajiaghayi, M.T., Kumar, A.: Stochastic steiner tree with non-uniform inflation. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) RANDOM 2007 and APPROX 2007. LNCS, vol. 4627, pp. 134–148. Springer, Heidelberg (2007)
12. Hajiaghayi, M.T., Kortsarz, G., Mirrokni, V.S., Nutov, Z.: Power optimization for connectivity problems. Math. Program. 110, 195–208 (2007)

13. Hassin, R., Segev, D.: The set cover with pairs problem. In: Sarukkai, S., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 164–176. Springer, Heidelberg (2005)
14. Kortsarz, G.: On the hardness of approximating spanners. Algorithmica 30, 432–450 (2001)
15. Kortsarz, G., Krauthgamer, R., Lee, J.R.: Hardness of approximability for vertex-connectivity network design problems. SIAM Journal on Computing 33, 185–199 (2004)
16. Oum, S., Seymour, P.: Approximating clique-width and branch-width. J. Combin. Theory Ser. B 96(4), 514–528 (2006)
17. Peleg, D.: Approximation algorithms for the Label-Cover$_{MAX}$ and Red-Blue Set Cover problems. J. Discrete Algorithms 5, 55–64 (2007)

In-Place Sorting

Viliam Geffert[1,⋆] and Jozef Gajdoš[2]

[1] Department of Computer Science – P. J. Šafárik University
Jesenná 5 – 04154 Košice – Slovakia
[2] College of International Business ISM Slovakia in Prešov
Duchnovičovo nám. 1 – 08001 Prešov – Slovakia
viliam.geffert@upjs.sk, gajdos@ismpo.sk

Abstract. We present an algorithm for asymptotically efficient sorting. Our algorithm sorts the given array $\mathcal{A}$ by the use of $n{\cdot}\lg n + O(n{\cdot}\lg\lg n)$ comparisons and $O(n)$ element moves. Moreover, this algorithm works in-place, using only a constant auxiliary workspace. This shrinks the gap between the known information-theoretic lower bound and the existing algorithms to $O(n{\cdot}\lg\lg n)$ comparisons, even if we require the algorithm to use only a constant auxiliary memory and a linear number of moves.

Keywords: in-place algorithms, sorting, computational complexity.

1 Introduction

Sorting is one of the oldest and most fundamental problems in computer science. The efficiency of a comparison-based sorting algorithm is given by two quantities: the number of pairwise element comparisons and the number of element moves carried out in the worst case, both expressed as a function of n, which is the number of elements to be sorted. It is well known that any comparison-based sorting algorithm must perform at least $\lg n! \approx n{\cdot}\lg n - 1,443n$ comparisons to sort an array consisting of n elements, see [4]. Moreover, we assume that the algorithm works *in-place*, that is, only one extra storage location (in addition to the input array) is available for storing elements aside. To store array indexes, counters, etc., only $O(1)$ integer storage locations, of $O(\lg n)$ bits each, are available. As was shown in [6], the lower bound for the number of moves performed by any in-place sorting algorithm is $\lfloor 3/2{\cdot}n \rfloor$.

The first algorithm that asymptotically matches these lower bounds and reduces simultaneously, the number of comparisons, moves, and storage is [2], which works in-place and uses $2n{\cdot}\lg n + o(n{\cdot}\lg n)$ comparisons and $(13{+}\varepsilon){\cdot}n$ element moves, where $\varepsilon > 0$ denotes an arbitrarily small, but fixed, real constant. However, the ultimate goal, stated in [5], is an in-place sorting algorithm that performs $n{\cdot}\lg n + O(n)$ comparisons and $O(n)$ element moves.

In this paper we present a sorting algorithm that works in-place and comes very close to this ultimate goal, with $n{\cdot}\lg n + O(n{\cdot}\lg\lg n)$ comparisons and $(34{+}\varepsilon){\cdot}n$

⋆ Supported by VEGA, the Slovak Grant Agency, under contract 1/0035/09.

element moves, performed in the worst case. The structure is similar to [2] and, moreover, we use [2] as a subroutine to sort segments of limited length.

The main difference between our algorithm and [2] is the introduction of *cache memory* and the subsequent usage of *pointer memory*. The cache enables us to save a significant number of comparisons, by reading the pointers not for individual input elements, but for the groups of elements stored in cache blocks. This allows to accomplish in-place sorting with a number of comparisons only $O(n \cdot \lg \lg n)$ above $n \cdot \lg n$ (the information-theoretic lower bound), keeping the number of moves linear. This result is mainly of theoretical interest, showing that the information-theoretic lower bound for comparisons cannot be asymptotically raised even by adding such drastic restrictions as a constant auxiliary memory together with linear number of moves.

2 Sorting with Additional Memory

First, let us concentrate on a simpler task. Assume that we are given a contiguous block A consisting of m *active elements*, which are to be sorted. Assume also that we are given an extra work space B called *buffer memory* of size at least $3m-1$ and a *pointer memory* Π, capable of storing at least $\lfloor 4m/(\lg m)^2 \rfloor$ bits. To enable movement of elements, we have one empty location called a *hole*. An assignment $a_j := a_i$ transports not only one element from the location i to j, but also the hole from location j to i.

2.1 Structure of the Memory

The buffer memory forms a contiguous block B, containing at least $3m-1$ *buffer elements*. All buffer elements are greater than or equal to a given *buffer separator* $b^{\preceq}$, placed in an extra location. All active elements from A are strictly smaller than $b^{\preceq}$ and therefore, by using a single comparison, it is possible to determine whether any given element is an active element or a buffer element.

The buffer memory B consists of three parts. At the left end of B, starting with its leftmost element, is a high level *frame memory* consisting of blocks which will be called *frames*. Right next to the last frame is a *cache memory* consisting of *cache blocks*. Both, the frame and the cache memory, are of fixed length. Finally at the right end of B is a *segment memory*, consisting of *segments* that are dynamically allocated from the right end of B and growing to the left.

Cache memory is used as a temporary storage for active elements before transportation to segment memory. All cache blocks are of fixed length c, where

$$c = \begin{cases} \lceil \lg m \rceil \\ \lceil \lg m \rceil + 1 \end{cases} \quad \text{so that } c \text{ is odd.} \tag{1}$$

The number of cache blocks $c_{\#}$ corresponds to the number of frame elements in R, which will be defined later. For the time being, let us just assume that $c_{\#} = R + 1 \leq 4m/(\lg m)^3$. The length of the cache memory is then

$$C = c_{\#} \cdot c = (R+1) \cdot c \leq (1 + o(1)) \cdot 4m/(\lg m)^2. \tag{2}$$

Every cache block is associated with a particular frame element, except the leftmost cache block, denoted by γ_0, not associated with any frame element. Thus, the starting position of the ith cache block, associated with the ith frame element, is $R + i \cdot c + 1$, where R is the total size of the frame memory.

At the beginning of computation, all cache blocks are *free*, containing buffer elements only. However, during the computation, any cache block might contain some active elements. The general structure of a cache block is $t_1 \ldots t_h b_{h+1} \ldots b_c$, for some $h \in \{0, \ldots, c\}$, where $t_1 \ldots t_h$ are some active elements and $b_{h+1} \ldots b_c$ are buffer elements. Neither $t_1 \ldots t_h$ nor $b_{h+1} \ldots b_c$ are sorted. In addition, the algorithm does not keep any information about the boundary h separating active and buffer elements, if the cache block is not being manipulated at the present moment. Since all active elements are strictly smaller than $b^{\preceq}$ and all buffer elements are greater than or equal to $b^{\preceq}$, we can quickly determine the number of active elements, by using a binary search with $b^{\preceq}$ over the c locations of the block, which costs only $1 + \lfloor \lg c \rfloor \leq O(\lg \lg m)$ comparisons, by (1).

The blocks in the segment memory have a similar structure as the blocks in the cache memory. All segments are of a fixed length s, defined as follows:

$$\begin{aligned} \kappa &= \left\{ \begin{array}{l} \lceil (\lg m)^3 \rceil \\ (\lceil (\lg m)^3 \rceil + 1) \end{array} \right. \qquad \text{so that } \kappa \text{ is odd, and} \\ s &= \kappa \cdot c. \end{aligned} \tag{3}$$

Since cache block size c is also an odd number, the size s of every segment is an *odd* integer multiple of c. Therefore, each segment can be divided into κ *subsegments*, where κ is odd, and the size of each subsegment is exactly c, the size of a single cache block. The number of active segments is bounded by

$$s_{\#} = \lfloor 2m/s \rfloor \leq 2m/(\lg m)^4. \tag{4}$$

The size of workspace reserved for the segment memory is therefore bounded by

$$S = s_{\#} \cdot s \leq 2m\,. \tag{5}$$

Here we assume that m is "sufficiently large," such that $s \leq m$, and hence $s_{\#} \geq 2$. A "short" block A is sorted in a different way, by the use of a modified heapsort.

All segments are *free* at the beginning of computation, containing buffer elements only. The starting position of the last segment that has been allocated is kept in a global index variable $\bar{s}$. Initially, $\bar{s}$ points to the right end of B, that is, the initial number of allocated segments is zero. To allocate a new segment, the procedure sets $\bar{s} := \bar{s} - s$, and returns $\bar{s}$ as the starting position of the new segment. Immediately after allocation, some $\lfloor \kappa/2 \rfloor$ subsegments containing active elements only (smaller than $b^{\preceq}$) are transported to the first $\lfloor \kappa/2 \rfloor$ subsegments of the new segment. The corresponding buffer elements are saved in the locations released by the active elements. From this point forward, the segment becomes *active.*

The structure of an active segment is $t_1 \ldots t_h b_{h+1} \ldots b_s$, where h is an integer multiple of κ ranging between 0 and s. The first h elements $t_1 \ldots t_h$ are active elements stored in the segment, while $b_{h+1}\dots b_s$ are some buffer elements. The value of h is kept between $\lfloor\kappa/2\rfloor\cdot c$ and $s-c$, so that at least one half (roughly) of elements in each active segment is active, and there is still a room for storing one more subsegment. Active and buffer elements in the segment are not sorted. Nevertheless, the boundary separating active and buffer elements can be easily computed by a binary search with $b^{\preceq}$ over the leftmost elements in the κ subsegments, which costs $1+\lfloor\lg\kappa\rfloor \leq 3\cdot\lg\lg m+1$ comparisons, by (3). Here we utilize the fact that each subsegment is either full (consisting of active elements only), or empty (containing only buffer elements), and that all nonempty subsegments form a contiguous zone starting from the left.

The frame memory is placed at the left end of B and consists of $r_{\#}$ *frame blocks* of size r, where

$$\begin{aligned} r &= 1+\lceil\lg(2m/s)\rceil && \leq 2+\lg(2m/8) \leq \lg m-1\,,\\ r_{\#} &= 2^{r-1} = 2^{\lceil\lg(2m/s)\rceil} && \leq 2\cdot 2m/s \qquad \leq 4m/(\lg m)^4, \end{aligned} \tag{6}$$

using (1), (3) and $m\geq 8$. That is, the total length of frame memory is

$$R = r_{\#}\cdot r \leq 4m/(\lg m)^3 - 1\,. \tag{7}$$

Using (2), (5), (7) and $m\geq 8$, we get that the total space requirements for the cache, segment and frame memories does not exceed the size of the buffer B, since $C+S+R \leq 4m/(\lg m)^2 + 2m + 4m/(\lg m)^3 \leq 3m-1$. A frame block can be either *free*, containing buffer elements only, or *active*, containing some active elements followed by some buffer elements. Initially, all frame blocks are free. During the computation, active frame blocks are concentrated in a contiguous left part of the frame, followed by some free frame blocks. However, there are some important differences from the cache and segment memory structure:

First, let us denote the elements in the frame memory by $x_1,\dots,x_R$.

Second, the active elements, forming a left part of a frame block, are in sorted order. So are the active frame blocks, forming a left part of the frame memory. More precisely, let $a_1,\dots,a_f$ denote the sequence of all active elements stored in the frame memory, obtained by reading active elements from left to right, ignoring buffer elements. Then $a_1,\dots,a_f$ is a sorted sequence of elements. Consequently, a subsequence of these, stored in the first (leftmost) positions of active frame blocks, denoted here by $a_{i_1}, a_{i_2},\dots,a_{i_g}$, must also be sorted. Here f denotes the total number of active elements in the frame, while g the number of active frame blocks, at the given moment. Similarly, $a_{i_j}a_{i_j+1}a_{i_j+2}\dots a_{i_{j+1}-1}$, the sequence of active elements stored in the jth frame block, is also sorted.

Third, the number of active elements in an active frame block can range between 1 and $r-1$. That is, we keep room for potential storing of one more active element in each active frame block. The only restriction follows from the fact that there are no free blocks in between some active blocks.

Relationship between the frame, cache and segments: For every element x_ℓ in the frame (no matter whether active or not) there is one cache block associated with it. Moreover, with each active frame element a_k we associate one segment σ_k. Thus, if we skip cache blocks associated with buffer elements in the frame, we get a sequence $\gamma_0, \gamma_1, \ldots, \gamma_f$ corresponding to active frame elements $a_1, \ldots, a_f$. Besides active cache blocks we also have a sequence of allocated segments $\sigma_0, \sigma_1, \ldots, \sigma_f$. Both, the cache block γ_k and the segment σ_k, for any $k \in \{1, \ldots, f\}$, contain some active elements satisfying $a_k \leq a \leq a_{k+1}$, taken from the array A and stored in the structure so far. The active elements satisfying $a_f \leq a$ are stored in γ_f or in σ_f, similarly, those satisfying $a \leq a_1$ are stored in γ_0 or σ_0. Note that no frame element is associated with the cache block γ_0 and the segment σ_0. Chronologically, γ_0 is the first cache block, and σ_0 is the first active segment that has been allocated. If $f = 0$, i.e., no active elements have been stored in the frame yet, all active elements are transported from A to γ_0 and, later if necessary, from γ_0 to σ_0.

Recall that we also maintain the invariant that the total number of active elements in each active cache/segment pair is at least $\lfloor s/2 \rfloor$. More precisely, either the number of active subsegments (all full) is at least $\lceil \kappa/2 \rceil$, or the number of such subsegments is $\lfloor \kappa/2 \rfloor$, but the associated cache block contains at least $\lfloor c/2 \rfloor$ active elements. Thus, if the frame contains f active elements at the given moment, namely, $a_1, \ldots, a_f$, for some $f \geq 1$, the total number of active elements, stored both in the frame and in the cache/segment pairs $\gamma_0/\sigma_0, \gamma_1/\sigma_1, \ldots, \gamma_f/\sigma_f$ is at least $f + (f+1) \cdot \lfloor s/2 \rfloor$. However, the total number of all active elements is exactly equal to m, which gives $m \geq (f+1) \cdot s/2$, and hence also $f+1 \leq 2m/s$. But $f+1$, the number of active segments, is an integer number, which gives that $f+1 \leq \lfloor 2m/s \rfloor$. Therefore, using (4) and (6),

$$f+1 \leq \lfloor 2m/s \rfloor = s_{\#}\,, \quad f \leq 2m/s \leq 2^{\lceil \lg(2m/s) \rceil} = r_{\#}\,. \tag{8}$$

As a consequence, we get that $f+1$, the number of active segments, does not exceed $s_{\#}$, the capacity of the segment memory. Second, f, the number of active elements in the frame, will never exceed $r_{\#}$, the total number of blocks in the frame, and hence there is enough room to store all active frame elements, even if each active frame block contained only a single element of the sequence $a_1, \ldots, a_f$. Recall that each cache block is associated with one element in the frame, and hence the requirement for cache memory is also sufficient.

Structure of the pointer memory: The relative order of active frame elements in the sequence $a_1, \ldots, a_f$ and the corresponding order of active cache blocks does not reflect the chronological order, in which the segments σ_0 and $\sigma_1, \ldots, \sigma_f$ are allocated. Therefore, with each element position in the frame, we associate a *pointer* to the starting position of the corresponding segment. More precisely, if the frame is viewed as a single contiguous zone of elements $x_1 \ldots x_R$, then the corresponding zone of pointers is $\pi_1 \ldots \pi_R$. If, for some ℓ, the element x_ℓ is a buffer element, then $\pi_\ell = 0$, which represents a *NIL* pointer. Conversely, if x_ℓ is an active element, then the value of π_ℓ represents the starting position of the segment associated with x_ℓ. (The pointer π_0 to the segment σ_0, having no "parent" in the frame, is stored separately, in a global index variable).

Since there are at most $s_{\#}$ segments, all of equal length, a pointer to a segment can be represented by an integer value ranging between 0 and $s_{\#} = \lfloor 2m/s \rfloor \leq m/2$, by (4). Thus, a single pointer can be represented by a block of p bits, where

$$p = 1 + \lfloor \lg s_{\#} \rfloor \leq \lg m \,. \tag{9}$$

The number of pointers is equal to R, the size of the frame. Therefore, $p_{\#} = R$. Thus, the pointer memory Π can be viewed as a contiguous array consisting of $p_{\#}$ bit blocks, of p bits each, and hence, by (7), its total length is at most

$$P = p_{\#} \cdot p = R \cdot p \leq \lfloor 4m/(\lg m)^2 \rfloor \,, \tag{10}$$

using also the fact that P must be an integer number.

Since an in-place algorithm can store only a limited amount of information in index variables, the pointer memory Π is actually simulated by two separate contiguous blocks Π_{L} and Π_{R}, each containing at least $\lfloor 4m/(\lg m)^2 \rfloor$ elements. Initially, Π_{L} and Π_{R} are sorted, and the largest (rightmost) element in Π_{L} is strictly smaller than the smallest (leftmost) element in Π_{R}. This allows us to encode the value of the jth bit, for any j ranging between 1 and $\lfloor 4m/(\lg m)^2 \rfloor$, by swapping the jth element of Π_{L} with the jth element of Π_{R}. Testing the value of the jth bit is equivalent to comparing the relative order of the corresponding elements in Π_{L} and Π_{R}, which costs only a single comparison. Setting a single bit value requires a single comparison and, optionally, 3 element moves.

2.2 Inserting Elements in the Structure

In the first phase, we take all m active elements and insert them into the structure described above. Throughout the process the structure is kept "balanced" and we also save all buffer elements from B in the locations released by inserted active elements. In the second phase, the active elements are collected back to A in sorted order. The goal of the first phase is to transport every active element from A to one of the segments $\sigma_0, \sigma_1, \ldots, \sigma_f$ in buffer memory B. For each active element a, we perform the following steps.

First, we find the "proper" block in the frame, which means finding the index j satisfying $a_{i_j} < a \leq a_{i_{j+1}}$. This is done by a binary search with the given element a over $a_{i_2}, \ldots, a_{i_g}$, that is, over the leftmost locations in the active frame blocks. Note that the element a_{i_1} is excluded from the range of the binary search. If $a \leq a_{i_2}$, the binary search will return $j = 1$, i.e., the first frame block. Similarly, for $a_{i_g} < a$, the binary search returns $j = g$, i.e., the last frame block. If $g < 2$, we can go directly to the first (and only) active frame block, that is, $j := 1$.

Second, by the use of a binary search with the given element a over the r locations in the jth active frame block, we find the "proper" active frame element for the element a, i.e., the index k satisfying $a_k < a \leq a_{k+1}$. Note that, since $a_{i_j} < a \leq a_{i_{j+1}}$, the elements a_k and a_{k+1} are between a_{i_j} and $a_{i_{j+1}}$ in the sequence $a_1, \ldots, a_f$ of all frame elements, not excluding the possibility that $a_{i_j} = a_k$, and/or $a_{k+1} = a_{i_{j+1}}$. Recall that the jth active frame block begins with the active elements $a_{i_j} a_{i_j+1} a_{i_j+2} \ldots a_{i_{j+1}-1}$, followed by some buffer elements, so that the length of the block is equal to r. These buffer elements are not sorted, however, they are all greater than or equal to $b^{\preceq}$. On the other hand, the element a, being active, is strictly smaller than $b^{\preceq}$. This allows us to use the binary search with the given element a in the standard way, which returns the index k satisfying $a_k < a \leq a_{k+1}$. For $a_{i_{j+1}-1} < a$, the binary search returns correctly $k = i_{j+1}-1$. If $j = 1$, that is, we are in the first frame block, the binary search may end up with $k = 0$, indicating that $a \leq a_1 = a_{i_1}$.

Third, after finding the frame element a_k we compute the starting position of the cache block γ_k associated with a_k. This value is obtained simply by computing $R + k \cdot c + 1$, which takes constant time, with no comparisons or moves.

Fourth, by the use of a binary search with the buffer separator $b^{\preceq}$ over the c elements in the current cache block, find the boundary h dividing the cache block into two parts, namely, $t_1 \dots t_h$, the active elements stored in the block, and $b_{h+1} \dots b_c$, some buffer elements, filling up the room.

Fifth, we save the buffer element b_{h+1} aside, to the current location of the hole, and store the given element a in the cache block. If $h+1 < c$, then we can repeat the process and insert the next active element from A into the structure. Otherwise, if $h + 1 = c$, then the current cache block γ_k has become full and cannot absorb any more elements. In this case we empty γ_k by transporting its content into the segment σ_k associated with the same element a_k in the frame.

The above process is repeated until all m active elements have been inserted in the structure. At this moment, the cache memory contains at most $(f{+}1) \cdot (c{-}1)$ active elements, since there are f active elements in the frame, hence, $f{+}1$ active cache blocks, each containing at most $c{-}1$ active elements. The inserting phase is finalized by emptying the entire cache memory. All active elements residing in cache blocks are moved to the corresponding segments. This is done simply by scanning through all the cache blocks $\gamma_0, \gamma_1, \dots, \gamma_R$. For each cache block γ_k, we determine whether it is active by comparing its leftmost element with $b^{\preceq}$. If γ_k is active, we empty the entire block γ_k, even if it contains some buffer elements.

Let us now determine the standard cost of inserting a single element. The binary search looking for a proper frame block inspects a range of $g - 1 < r_{\#}$ elements, and hence it performs at most $1 + \lfloor \lg r_{\#} \rfloor \leq \lg m$ comparisons, by (6). The second binary search, looking for a proper active element within the given frame block, inspects a range of r elements, performing at most $1 + \lfloor \lg r \rfloor \leq O(\lg\lg m)$ comparisons, by (6). Computing the starting position of the associated cache block takes constant time and the subsequent binary search with $b^{\preceq}$ over c locations of this block uses at most $1 + \lfloor \lg c \rfloor \leq O(\lg\lg m)$ comparisons, by (1). Finally, saving one buffer element and transporting the element a to the current cache block uses 2 element moves. Since the total of m elements is inserted in this way, the number of comparisons is bounded by $m{\cdot}\lg m + O(m{\cdot}\lg\lg m)$ and the number of moves by $2m$, not counting the costs for emptying active cache blocks. It can be shown that, in the course of the entire computation, $m + O(m{\cdot}\lg\lg m)$ comparisons and $2m$ element moves are sufficient for emptying cache blocks.

In addition, the final dumping of the entire cache memory consists of scanning through the leftmost positions of all cache blocks (comparing them with $b^{\preceq}$), which requires $R{+}1 \leq 4m/(\lg m)^3$ comparisons, by (7), and dumping of $f{+}1 \leq 2m/s$ active cache blocks, by (8). It is easy to show that emptying a single cache block consumes $\lg m + O(\lg\lg m)$ comparisons and $2c$ element moves. Thus the number of comparisons is at most $2m/(\lg m)^3 + O(m{\cdot}\lg\lg m)$ and the number of moves $4m/(\lg m)^3$. Summing up, the final dumping of cache memory consumes at most $6m/(\lg m)^3 + O(m{\cdot}\lg\lg m)$ comparisons and $4m/(\lg m)^3$ moves.

Lemma 1. *Inserting m elements in the structure requires $m{\cdot}\lg m + O(m{\cdot}\lg\lg m)$ comparisons and $(4+o(1)){\cdot}m$ element moves.*

2.3 Extracting in Sorted Order

In the second phase, we transport all active elements back to A in sorted order. Let f_{m} denote the maximal value of f, corresponding to the number of active elements in the frame at the moment when the last active element has been inserted. Thus, the frame memory contains the sorted sequence of active elements $a_1, \ldots, a_{f_{\mathrm{m}}}$, intertwined with some buffer elements, so the total size of the frame is R, consisting of elements $x_1, \ldots, x_R$. The rest of the active elements is in segments $\sigma_0, \sigma_1, \ldots, \sigma_{f_{\mathrm{m}}}$, with σ_k containing active elements that satisfy $a_k \leq a \leq a_{k+1}$. Thus, to produce the sorted order of all active elements, it is sufficient to move, back to A, the sequence $\sigma'_0, a_1, \sigma'_1, a_2, \sigma'_2, \ldots, a_{f_{\mathrm{m}}}, \sigma'_{f_{\mathrm{m}}}$, where σ'_k denotes the block of sorted active elements contained in σ_k.

The procedure begins with moving the block σ'_0 to A. (The problem of sorting a given segment σ_k is described later.)

Then, in a loop iterated for $\ell = 1, \ldots, R$, check whether x_ℓ is an active element. This requires only a single comparison, comparing x_ℓ with $b^{\preceq}$. If x_ℓ is a buffer element, it is skipped, and we can go to the next element in the frame.

If x_ℓ is an active element, i.e., $x_\ell = a_k$, for some k, the procedure saves the leftmost buffer element, not moved yet from the output block A, in the current location of the hole and, after that, moves $x_\ell = a_k$ to A. (The first free position in A, i.e., the position of the leftmost buffer element, is kept in a separate global index variable, and incremented each time a new active element is transported back to A). Then we read the value encoded in the pointer π_ℓ and compute the starting position of the segment σ_k. We sort σ_k by the procedure described bellow, creating σ'_k. During this process we obtain the value h_k, which is the number of active elements in the given segment. After that, we move the h_k active elements from σ'_k back to A in the same fashion as described above, i.e., we save the leftmost buffer element in A to the current location of the hole, transport the leftmost active element from σ'_k to A and increment the indices. Again, only two moves per element are sufficient.

Let us derive computational costs of the above procedure, not including the cost of sorting σ_k. Testing whether x_ℓ is an active element, for $\ell = 1, \ldots, R$, requires $R \leq O(m/(\lg m)^3)$ comparisons, by (7). Transporting m elements back to A requires only $2m$ moves in total, since only active elements are moved. Reading the values of f_{m} pointers, of length p bits each, can be done with $f_{\mathrm{m}}{\cdot}p \leq r_{\#}{\cdot}p \leq O(m/(\lg m)^3)$ comparisons, using (6), (8), and (9). By summing up we see that, if we exclude the costs of sorting the segments, extracting the elements back to A in sorted order uses $O(m/(\lg m)^3)$ comparisons and $2m$ moves.

Let us now return to the problem of sorting a single segment σ_k. The number of active elements in this segment is h_k, and clearly $h_k \leq s \leq (1+o(1))\cdot(\lg m)^4$, using (1) and (3). Initially, we determine the value of h_k by the use of a binary search with $b^{\preceq}$ over the s locations of the segment. This costs $1+\lfloor \lg s \rfloor \leq O(\lg\lg m)$ comparisons. Then we sort the h_k active elements by an in-place sorting algorithm [2], using $(2+o(1))\cdot h_k\cdot\lg h_k$ comparisons and $(13+\varepsilon)\cdot h_k$ moves. A total of $f_{\mathrm{m}}+1$ segments is sorted this way, while extracting all m active elements. Therefore the total number of comparisons performed by [2] is bounded by $(f_{\mathrm{m}}+1)\cdot(2+o(1))\cdot h_k\cdot\lg h_k \leq O(m\cdot\lg\lg m)$, using (3) and (8). The number of moves is bounded by $(f_{\mathrm{m}}+1)\cdot(13+\varepsilon)\cdot h_k \leq (26+\varepsilon)\cdot m$. This gives:

Lemma 2. *Transporting m active elements from the structure back to A in sorted order requires $O(m\cdot\lg\lg m)$ comparisons and $(28+\varepsilon)\cdot m$ element moves, where $\varepsilon > 0$ is an arbitrarily small, but fixed, real constant.*

At this moment we are not yet able to determine the cost for the entire algorithm by simply adding the costs in Lems. 1 and 2. This is due to the fact that, while inserting elements in the structure some segments might become full, and need to be rebalanced. Furthermore, after each segment rebalancing some frame block might also become full and trigger rebalancing at the frame level, which is yet another special situation. The costs associated with these rebalancing operations must also be taken into account. It can be shown that, in the course of the entire computation, $O(m\cdot\lg\lg m)$ comparisons and $(4/(\lg m)^3+\varepsilon)\cdot m$ moves are sufficient to keep the segment memory balanced. The cost of keeping the frame memory balanced is $O(m/\lg m)$ comparisons and moves. (For details, see [2,8].)

2.4 Summary

Theorem 3. *The cost of sorting the given block A of size m is $m\cdot\lg m + O(m\cdot\lg\lg m)$ comparisons and $(32+\varepsilon)\cdot m$ moves, where $\varepsilon > 0$ is an arbitrarily small, but fixed, real constant, provided we can use additional buffer and pointer memories, of respective sizes $3m-1$ and $\lfloor 4m/(\lg m)^2 \rfloor$.*

The algorithm assumes that m is "sufficiently large", so that the following conditions hold true. First, the segment size s, defined by (3), satisfies $s \leq m$. Second, the space requirement for cache, segment and frame memories, stated in Sect. 2.1, satisfies $C+S+R \leq 3m-1$. And finally, the number of active elements in the frame f, defined by (8), satisfies $f \geq 1$. All of these assumptions hold for each $m > 2^{17} = 131\,072$. Shorter blocks are handled in a different way, by the use of a modified heapsort, keeping the bounds presented in Thm. 3.

3 In-Place Sorting

Now we can present an in-place algorithm sorting the given array $\mathcal{A}$ consisting of n elements. If $n \leq 2^{17}$, the array is sorted directly by a modified heapsort. For $n > 2^{17}$, the task of the main program is to provide sufficiently large pointer and buffer memories for the procedure presented in Sect. 2. The size of the largest block ever sorted by the procedure of Sect. 2 will not exceed $m = n/4$. Using (10) and the fact that the function $4x/(\lg x)^2$ is monotone increasing for $x \geq 8$, we get the size of the pointer memory bounded by $P = \lfloor n/(\lg(n/4))^2 \rfloor$.

The pointer memory is built by collecting two contiguous blocks $\mathit{\Pi}_{\mathrm{L}}$ and $\mathit{\Pi}_{\mathrm{R}}$. The block $\mathit{\Pi}_{\mathrm{L}}$, placed at the left end of $\mathcal{A}$, will contain the smallest P elements of the array $\mathcal{A}$, while $\mathit{\Pi}_{\mathrm{R}}$, placed at the right end, the largest P elements. To create the blocks $\mathit{\Pi}_{\mathrm{L}}$ and $\mathit{\Pi}_{\mathrm{R}}$, we can use a generalized version of heapsort, which employs a modified heap-like structure. Building the pointer memory then requires $O(n)$ comparisons and $O(n/\lg n)$ moves.

Now the configuration of the array $\mathcal{A}$ has changed to $\mathit{\Pi}_{\mathrm{L}}\mathcal{A}'\mathit{\Pi}_{\mathrm{R}}$, where $\mathcal{A}'$ denotes the elements to be sorted. Before proceeding further, the algorithm verifies, with a single comparison, whether the rightmost element in $\mathit{\Pi}_{\mathrm{L}}$ is strictly smaller than the leftmost element in $\mathit{\Pi}_{\mathrm{R}}$. If this is not the case, all elements in $\mathcal{A}'$ must be equal to these two elements. Therefore, the algorithm terminates, the entire array $\mathcal{A}$ has already been sorted. Conversely, if $\mathit{\Pi}_{\mathrm{L}}$ and $\mathit{\Pi}_{\mathrm{R}}$ pass the test above, they can be used to imitate a pointer memory consisting of P bits.

Once the blocks $\mathit{\Pi}_{\mathrm{L}}$ and $\mathit{\Pi}_{\mathrm{R}}$ have been created, the zone $\mathcal{A}'$ is kept in the form $\mathcal{A}_{\mathrm{S}}\mathcal{A}_{\mathrm{U}}$, where $\mathcal{A}_{\mathrm{S}}$ and $\mathcal{A}_{\mathrm{U}}$ represent the sorted and unsorted parts of $\mathcal{A}'$, respectively. Each element in $\mathcal{A}_{\mathrm{S}}$ is strictly smaller than the smallest element of $\mathcal{A}_{\mathrm{U}}$. The routine described here is a partition-based loop. In the course of the ith iteration, the length of $\mathcal{A}_{\mathrm{U}}$ is n_i, with $n_i < n_{i-1}$. Initially, for $i = 0$, $\mathcal{A}_{\mathrm{S}}$ is empty, $\mathcal{A}_{\mathrm{U}} = \mathcal{A}'$, and $n_0 = n - 2P < n$. The loop proceeds as follows:

First, find $b^{\preceq}$, an element of rank $\lceil n_i/4 \rceil$ in $\mathcal{A}_{\mathrm{U}}$. The selection procedure places this element at the right end of $\mathcal{A}_{\mathrm{U}}$, so the configuration of $\mathcal{A}'$ changes to $\mathcal{A}_{\mathrm{S}}\mathcal{A}'_{\mathrm{U}}b^{\preceq}$. Here $\mathcal{A}'_{\mathrm{U}}$ denotes a mix of elements in $\mathcal{A}_{\mathrm{U}}$, of length $n_i - 1$.

Second, $\mathcal{A}'_{\mathrm{U}}$ is partitioned into $A_<$ and $B_\geq$ consisting, respectively, of elements strictly smaller than $b^{\preceq}$ and of those greater than or equal to $b^{\preceq}$. The configuration of the array thus changes to $\mathcal{A}_{\mathrm{S}}A_<B_\geq b^{\preceq}$. The lengths of $A_<$ and $B_\geq$ will be denoted by $n_{i,<}$ and $n_{i,\geq}$. Note that, even for a large block $\mathcal{A}_{\mathrm{U}}$, we may obtain a very short (or empty) block $A_<$, since many elements may be equal to $b^{\preceq}$.

Third, sort the block $A_<$ by the procedure described in Sect. 2, using some initial segments of $\mathit{\Pi}_{\mathrm{L}}$ and $\mathit{\Pi}_{\mathrm{R}}$ as a pointer memory and of $B_\geq$ as a buffer memory, with $b^{\preceq}$ as a buffer separator. This is possible, since $b^{\preceq}$ has been selected as an element of rank $\lceil n_i/4 \rceil$, and hence $n_{i,<} \leq \lceil n_i/4 \rceil - 1 \leq n_i/4$, with $n_{i,<} + n_{i,\geq} + 1 = n_i$. But the required size of buffer is only $3n_{i,<} - 1 \leq 3/4 \cdot n_i - 1 = n_i - 1 - n_i/4 \leq n_i - 1 - n_{i,<} = n_{i,\geq}$. Therefore, the block $B_\geq$ of length $n_{i,\geq}$ is sufficiently long. Similarly, the required number of bits for pointers is $\lfloor 4n_{i,<}/(\lg n_{i,<})^2 \rfloor \leq \lfloor 4(n/4)/(\lg(n/4))^2 \rfloor = P$, and hence the pointer memory is also sufficiently large. (If $n_{i,<} \leq 2^{17}$, $A_<$ is sorted as a short block).

Fourth, restore the sorted order in $\mathit{\Pi}_{\mathrm{L}}$ and $\mathit{\Pi}_{\mathrm{R}}$, by clearing all bits to zero.

Fifth, after sorting $A_<$, the configuration of $\mathcal{A}'$ is $\mathcal{A}_{\mathrm{S}}A_{<,\mathrm{S}}B'_\geq b^{\preceq}$, where $A_{<,\mathrm{S}}$ denotes the sorted version of the block $A_<$ and $B'_\geq$ a mixed up version of $B_\geq$. Now put the first element in $B'_\geq$ aside and move $b^{\preceq}$ to the first position after $A_{<,\mathrm{S}}$.

After that, collect all elements smaller than or equal to $b^{\preceq}$ to the left part of $B'_{\geq}$, processing also the element put aside. This actually partitions $B'_{\geq}$ into two blocks $A_{=}$ and $B_{>}$ consisting, respectively, of elements equal to $b^{\preceq}$ and of those strictly greater than $b^{\preceq}$, of respective lengths $n_{i,=}$ and $n_{i,>}$. Clearly, $n_{i,=}+n_{i,>}=n_{i,\geq}$. The configuration has changed to $\mathcal{A}_{\mathrm{S}}A_{<,\mathrm{S}}b^{\preceq}A_{=}B_{>}$.

Sixth, observe that $\mathcal{A}_{\mathrm{S}}A_{<,\mathrm{S}}b^{\preceq}A_{=}$ and $B_{>}$ can be viewed as "new" variants of blocks $\mathcal{A}_{\mathrm{S}}$ and $\mathcal{A}_{\mathrm{U}}$. Thus, we can start a new iteration, with $B_{>}$ as a new block $\mathcal{A}_{\mathrm{U}}$, of length $n_{i+1}=n_{i,>}$. The above process is iterated until the length of unsorted part drops to 2^{17}, or below. This residue is then sorted as a short block, without using a buffer or pointers.

Now we can derive computational costs. First, recall that $b^{\preceq}$ has been selected as an element of rank $\lceil n_i/4\rceil$, and hence $n_{i+1}=n_{i,>}\leq n_i-\lceil n_i/4\rceil\leq 3/4\cdot n_i$. Taking into account that $n_0\leq n$, we get $n_i\leq(3/4)^i\cdot n$, for each $i\geq 0$. Second, observe that in different iterations, the final locations occupied by $A_{<,\mathrm{S}}$, $b^{\preceq}$, and $A_{=}$, do not overlap. This gives:

$$\begin{aligned} \textstyle\sum_{i=0}^{\mathcal{I}-1} n_i &\leq 4n\,,\\ \mathcal{I} &\leq O(\lg n)\,,\\ \textstyle\sum_{i=0}^{\mathcal{I}-1}(n_{i,<}+1+n_{i,=})+n_{\mathcal{I}} &\leq n\,, \end{aligned} \tag{11}$$

where $\mathcal{I}$ denotes the number of iterations and $n_{\mathcal{I}}$ the length of the residual block.

Let us now present the costs for the ith iteration. Selection of $b^{\preceq}$ in a block of length n_i, costs $O(n_i)$ comparisons and $\varepsilon\cdot n_i$ moves, by [3]. Partitioning of $\mathcal{A}'_{\mathrm{U}}$ into blocks $A_{<}$ and $B_{\geq}$ can be done with n_i comparisons and $2n_{i,<}+1$ moves, since the length of $\mathcal{A}'_{\mathrm{U}}$ is n_i-1, and the number of collected elements, strictly smaller than $b^{\preceq}$, is $n_{i,<}$. The cost of sorting the block $A_{<}$ is bounded by $n_{i,<}\cdot\lg n_{i,<}+O(n_{i,<}\cdot\lg\lg n_{i,<})\leq n_{i,<}\cdot\lg n+O(n_{i,<}\cdot\lg\lg n)$ comparisons and $(32+\varepsilon)\cdot n_{i,<}$ moves, by Thm. 3. Restoring the sorted order in Π_{L} and Π_{R}, by clearing all bits, costs $O(P)\leq O(n/(\lg n)^2)$ comparisons and moves. Positioning $b^{\preceq}$ to the right of $A_{<,\mathrm{S}}$ requires only 2 element moves. Finally, the ith iteration is concluded by partitioning $B'_{\geq}$ into blocks $A_{=}$ and $B_{>}$, with at most $n_{i,\geq}\leq n_i$ comparisons and $2n_{i,=}+1$ moves, since the length of $B'_{\geq}$ is $n_{i,\geq}$, and the number of collected elements, equal to $b^{\preceq}$, is $n_{i,=}$. The cost of sorting the residual short block does not exceed the bounds for the standard case; $n_{\mathcal{I}}\cdot\lg n_{\mathcal{I}}+\lg\lg n_{\mathcal{I}}+3\leq n_{\mathcal{I}}\cdot\lg n+O(n_{\mathcal{I}}\cdot\lg\lg n)$ comparisons and $20n_{\mathcal{I}}\leq(32+\varepsilon)\cdot n_{\mathcal{I}}$ moves. Now we can sum the above costs over all iterations, using (11):

$$\begin{aligned} C(n) &\leq \textstyle\sum_{i=0}^{\mathcal{I}-1} n_i\cdot O(1)+\sum_{i=0}^{\mathcal{I}-1} n_{i,<}\cdot(\lg n+O(\lg\lg n))+\sum_{i=0}^{\mathcal{I}-1} O(n/(\lg n)^2)\\ &\quad + n_{\mathcal{I}}\cdot(\lg n+O(\lg\lg n))\\ &\leq O(n)+n\cdot(\lg n+O(\lg\lg n))+O(n/\lg n)\leq n\cdot\lg n+O(n\cdot\lg\lg n)\,,\\ M(n) &\leq \textstyle\sum_{i=0}^{\mathcal{I}-1}\varepsilon\cdot n_i+\sum_{i=0}^{\mathcal{I}-1}(34+\varepsilon)\cdot n_{i,<}+\sum_{i=0}^{\mathcal{I}-1}2n_{i,=}+\sum_{i=0}^{\mathcal{I}-1}O(n/(\lg n)^2)\\ &\quad +(32+\varepsilon)\cdot n_{\mathcal{I}}\\ &\leq \varepsilon\cdot n+n\cdot(34+\varepsilon)+O(n/\lg n)\leq(34+\varepsilon)\cdot n\,, \end{aligned}$$

where $\varepsilon > 0$ is an arbitrarily small, but fixed, real constant. The above analysis did not include the costs of the initial building of pointer memory. However, this can be done with only $O(n)$ comparisons and $O(n/\lg n)$ moves, and hence the bounds displayed above represent the total computational costs of the algorithm.

Theorem 4. *The given array, consisting of n elements, can be sorted in-place by performing at most $n \cdot \lg n + O(n \cdot \lg \lg n)$ comparisons and $(34+\varepsilon) \cdot n$ element moves, where $\varepsilon > 0$ denotes an arbitrarily small, but fixed, real constant. The number of auxiliary arithmetic operations with indices is bounded by $O(n \cdot \lg n)$.*

4 Concluding Remarks

We have shown that sorting can be accomplished with almost optimal number of comparisons, i.e., $n \cdot \lg n + O(n \cdot \lg \lg n)$, even if we keep the number of moves bounded by $O(n)$ and sorting in-place. Our algorithm is mainly of theoretical interest, because of its complexity. A captivating problem would be to devise a practical in-place sorting algorithm that matches the upper bounds of Thm. 4. However, an algorithm with $\varepsilon \cdot n \log n$ moves is relatively easy to implement [7].

Note that our algorithm does not sort stably, since the buffer elements can be mixed up quite arbitrarily and the original order of equal buffer elements cannot be recovered. Whether there exists a stable in-place sorting algorithm performing, in the worst case, $n \cdot \lg n + o(n \cdot \lg n)$ comparisons and $O(n)$ element moves is left as an open problem. So far, for stable in-place sorting, the best known upper bounds are $O(n \cdot \lg n)$ comparisons and $O(n)$ moves [1].

The upper bounds of Thm. 4 are not optimal and can be improved, which is left as another open problem.

References

1. Franceschini, G.: Sorting Stably, In-Place, with $O(n \lg n)$ Comparisons and $O(n)$ Moves. Theory of Computing Systems 40, 327–353 (2007)
2. Franceschini, G., Geffert, V.: An In-Place Sorting with $O(n \lg n)$ Comparisons and $O(n)$ Moves. J. Assoc.Comput. Mach. 52, 515–537 (2005)
3. Geffert, V., Kollár, J.: Linear-time in-place selection with $\varepsilon \cdot n$ element moves. Comput. & Informatics 25, 333–350 (2006)
4. Knuth, D.: The Art of Computer Programming, Sorting and Searching, 2nd edn., vol. 3. Addison-Wesley, Reading (1973, 1998)
5. Munro, J., Raman, V.: Sorting with minimum data movement. J. Algorithms 13, 374–393 (1992)
6. Munro, J., Raman, V.: Selection from read-only memory and sorting with minimum data movement. Theoret. Comput. Sci. 165, 311–323 (1996)
7. Reinhardt, K.: Sorting in-place with a worst case complexity of $n \log n - 1.3n + O(\log n)$ comparisons and $\varepsilon n \log n + O(1)$ transports. In: Proc. Internat. Symp. Algorithms and Comput, vol. 650, pp. 489–498. Springer, Heidelberg (1992)
8. Willard, D.: Maintaining dense sequential files in a dynamic environment. In: Proc. Symp. Theory of Comput., pp. 114–121. ACM Press, New York (1982)

On d-Regular Schematization of Embedded Paths

Andreas Gemsa[1], Martin Nöllenburg[1,2,⋆], Thomas Pajor[1], and Ignaz Rutter[1]

[1] Institute of Theoretical Informatics, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany

[2] Department of Computer Science, University of California, Irvine, USA

Abstract. In the d-regular path schematization problem we are given an embedded path P (e.g., a route in a road network) and an integer d. The goal is to find a *d-schematized* embedding of P in which the orthogonal order of all vertices in the input is preserved and in which every edge has a slope that is an integer multiple of $90°/d$. We show that deciding whether a path can be d-schematized is NP-hard for any integer d. We further model the problem as a mixed-integer linear program. An experimental evaluation indicates that this approach generates reasonable route sketches for real-world data.

1 Introduction

Angular or $\mathscr{C}$-oriented schematizations of graphs refer to a class of graph drawings, in which the admissible edge directions are limited to a given set $\mathscr{C}$ of (usually evenly spaced) slopes. This includes the well-known class of orthogonal drawings and extends more generally to k-linear drawings, e.g., octilinear metro maps. Applications of schematic drawings can be found in various domains such as cartography, VLSI layout, and information visualization.

In many schematization scenarios the input is not just a graph but a graph with an initial drawing that has to be schematized according to the given set of slopes. This is the case, e.g., in cartography, where the geographic positions of network vertices and edges are given [6], in sketch-based graph drawing, where a sketch of a drawing is given and the task is to improve or schematize that sketch [3], or in dynamic graph drawing, where each drawing in a sequence of drawings must be similar to its predecessor [5]. For such a redrawing task it is crucial that the mental map [13] of the user is preserved, i.e., the output drawing must be as similar as possible to the input. Misue et al. [13] suggested preserving the *orthogonal order* of the input drawing as a simple criterion for maintaining a set of basic spatial properties of the input, namely the relative above/below and left/right positions of all pairs of input nodes. The orthogonal order has been used successfully as a means for maintaining the mental map [4,7,9,11].

The motivation behind the work presented here is the visualization of routes in road networks as sketches for driving directions. An important property of a route sketch is that it focuses on road changes and important landmarks rather than exact geography and distances. Typically the start and destination lie in populated areas that are locally reached via a sequence of relatively short road segments. On the other hand, the majority

⋆ Supported by grant NO 899/1-1 of the German Research Foundation (DFG).

of the route typically consists of long highway segments with no or only few road changes. This property makes it difficult to display driving directions for the whole route in a single traditional map since some areas require much smaller scales than others. The strength of route sketches for this purpose is that they are not drawn to scale but rather use space proportionally to the route complexity.

Related Work. Geometrically, we can consider a route to be an embedded path in the plane. The simplification of paths (or polylines) in cartography is well studied and the classic line simplification algorithm by Douglas and Peucker [8] is one of the most popular methods. Two more recent algorithms were proposed for $\mathscr{C}$-oriented line simplification [12,14]. These line simplification algorithms, however, are not well suited for drawing route sketches since they keep the positions of the input points fixed or within small local regions around the input points and thus edge lengths are more or less fixed. On the other hand, Agrawala and Stolte [1] presented a system called *LineDrive* that uses heuristic methods based on simulated annealing to draw route sketches. Their method allows distortion of edge lengths and angles. It does not, however, restrict the set of edge directions and does not give hard quality guarantees for the mental map such as the preservation of the orthogonal order.

A graph drawing problem that has similar constraints as drawing route sketches is the metro-map layout problem, in which an embedded graph is to be redrawn octilinearly. The problem is known to be NP-hard [15] but it can be solved successfully in practice by mixed-integer linear programming [16]. The existing methods covered in a survey by Wolff [17] do aim to keep the mental map of the input but no strict criterion like the orthogonal order is applied. Brandes and Pampel [4] studied the path schematization problem in the presence of orthogonal order constraints in order to preserve the mental map. They showed that deciding whether a rectilinear schematization exists that preserves the orthogonal order of the input path is NP-hard. They also showed that schematizing a path using arbitrarily oriented unit-length edges is NP-hard. Delling et al. [7] gave an efficient algorithm to compute $\mathscr{C}$-oriented drawings of monotone paths that preserve the orthogonal order and have minimum schematization cost. The schematization cost counts the number of edges that are not drawn with their closest $\mathscr{C}$-oriented direction. The authors also sketch a heuristic approach for schematizing non-monotone paths.

Contributions. In this paper we close the complexity gap of the path schematization problem that remained open between the hardness result of Brandes and Pampel [4] for rectilinear paths and the efficient algorithm of Delling et al. [7] for monotone $\mathscr{C}$-oriented paths. We prove that deciding whether a $\mathscr{C}$-oriented orthogonal-order preserving drawing of an embedded input path exists is NP-hard, even if the path is simple. This is true for every *d-regular* set $\mathscr{C}$ of slopes that have angles that are integer multiples of $90°/d$ for any integer d. The case $d = 1$ is covered by Brandes and Pampel [4] but their proof relies on the absence of diagonal edges and hence does not extend to other values of d. We show the hardness in the octilinear case $d = 2$ in Section 3 and subsequently, in Section 4, how this result extends to the general d-regular case for $d > 2$. We finally design and evaluate a mixed integer-linear program (MIP) for solving the d-regular path schematization problem in Section 5. Our experimental results show that routes in practice usually consist of only a small number of relevant road segments and that our MIP is indeed able to quickly generate reasonable sketches for those routes. Omitted proofs and details about the MIP are found in the appendix.

2 Preliminaries

A plane embedding of a graph $G = (V,E)$ is a mapping $\pi : G \rightarrow \mathbb{R}^2$ that maps every vertex $v \in V$ to a distinct point $\pi(v) = (x_\pi(v), y_\pi(v))$ and every edge $e = uv \in E$ to the line segment $\pi(e) = \overline{\pi(u)\pi(v)}$ such that no two edges e_1, e_2 cross in π except at common endpoints. For simplicity we also use the terms vertex and edge to refer to their images under an embedding.

We measure the slope of an edge uv as the counterclockwise angle formed between the horizontal line through $\pi(u)$ and the line segment $\pi(uv)$. For a set of angles $\mathscr{C}$ we say that a drawing is *$\mathscr{C}$-oriented* if the slope of every edge $e \in E$ is contained in $\mathscr{C}$. A set of slopes $\mathscr{C}$ is called *d-regular* for an integer d if $\mathscr{C} = \mathscr{C}_d = \{i \cdot 90^\circ/d \mid i \in \mathbb{Z}\}$.

Let π and ρ be two embeddings of the same graph G. We say that ρ respects the *orthogonal order* [4] of π if for any two vertices u and $v \in V$ it holds that $x_\rho(u) \leq x_\rho(v)$ if $x_\pi(u) \leq x_\pi(v)$ and $y_\rho(u) \leq y_\rho(v)$ if $y_\pi(u) \leq y_\pi(v)$. In other words, the orthogonal order defines the relative above-below and left-right positions of any two vertices.

Let (G,π) be a graph G with a plane input embedding π. A *d-regular schematization* (or *d-schematization*) of (G,π) is a plane embedding ρ that is $\mathscr{C}_d$-oriented, that preserves the orthogonal order of π and where no two vertices are embedded at the same coordinates. We also call ρ *valid* if it is a d-schematization of (G,π).

3 Hardness of 2-Regular Path Schematization

In this section we show that the problem of deciding whether there is a 2-schematization for a given embedded graph (G,π) is NP-hard, even if G is a simple path. In the latter case we denote the problem as the (d-regular) PATH SCHEMATIZATION PROBLEM (PSP). We fix $d = 2$. To prove that 2-regular PSP is NP-hard we first show hardness of the closely related 2-regular UNION OF PATHS SCHEMATIZATION PROBLEM (UPSP), where (G,π) is a set $\mathscr{P}$ of k embedded disjoint paths $\mathscr{P} = \{P_1, \ldots, P_k\}$.

We show that 2-regular UPSP is NP-hard by a reduction from MONOTONE PLANAR 3-SAT, which is known to be NP-hard [2]. MONOTONE PLANAR 3-SAT is a special variant of PLANAR 3-SAT where each clause either contains exactly three positive literals or exactly three negative literals and additionally, the variable-clause graph admits a planar drawing such that all variables are on the x-axis, the positive clauses are embedded below the x-axis and the negative clauses above the x-axis. An example instance of MONOTONE PLANAR 3-SAT is depicted in Fig. 1. In a second step, we show how to augment the set of paths $\mathscr{P}$ to form a single simple path P that has the same properties as $\mathscr{P}$ and thus proves that PSP is NP-hard.

In the following, we assume that φ is a given MONOTONE PLANAR 3-SAT instance with variables $X = \{x_1, \ldots, x_n\}$ and clauses $C = \{c_1, \ldots, c_m\}$.

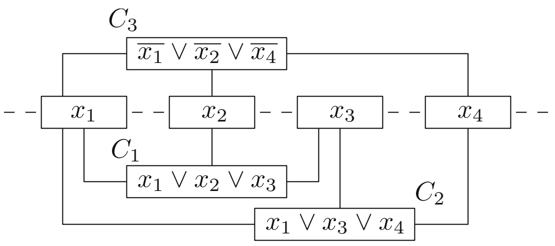


Fig. 1. A MONOTONE PLANAR 3-SAT instance with four variables and three clauses

3.1 Hardness of the UNION OF PATHS SCHEMATIZATION PROBLEM

In the following we first introduce the different types of required gadgets and then show how to combine them in order to prove the hardness of UPSP.

Border Gadget. For all our gadgets we need to control the placement of vertices on discrete positions. Thus our first gadget is a path whose embedding is unique up to scaling and translation. This path will induce a grid on which we subsequently place the remaining gadgets.

Let $\Delta_x(u,v)$ be the x-distance between the two vertices u and v. Likewise, let $\Delta_y(u,v)$ be the y-distance between u and v. We call a simple path P with embedding π *rigid* if a 2-schematization of (P,π) is unique up to scaling and translations. Further, we call an embedding π of P *regular* if there exists a length $\ell > 0$ such that for any two vertices u,v of P it holds that $\Delta_x(u,v) = z_x \cdot \ell$ and $\Delta_y(u,v) = z_y \cdot \ell$ for some $z_x, z_y \in \mathbb{Z}$. Thus in a regular embedding of P all vertices are embedded on a grid whose cells have side length ℓ. For our border gadget we construct a simple path B of appropriate length with embedding π that is both rigid and regular. Hence, π is essentially the unique 2-schematization of (B,π), and after rescaling we can assume that all points of B lie on an integer grid. The border gadget consists of a horizontal component B_h and a vertical component B_v which share a common starting vertex $v_1 = v'_1$, see Fig. 2. The component B_h alternates between a 45° edge to the upper right and a vertical edge downwards. The vertices are placed such that their y-coordinates alternate and hence all odd, respectively all even, vertices have the same y-coordinate. The vertical component consists of a copy of the horizontal component rotated by 90° in clockwise direction around v_1. To form B, we connect B_v and B_h by identifying their starting points v_1 and v'_1.

Lemma 1. *The border gadget B with its given embedding π is rigid and regular.*

In the following we define the length of a grid cell induced by B as 1 so that we obtain an integer grid. We choose B long enough to guarantee that any vertex of our subsequent gadgets lies vertically and horizontally between two pairs of vertices in B.

The grid in combination with the orthogonal order gives us the following properties:

1. If we place a vertex v with two integer coordinates, i.e., on a grid point, its position in any valid embedding is unique;
2. if we place a vertex v with one integer and one non-integer coordinate, i.e., on a grid edge, its position in any valid embedding is on that grid edge;

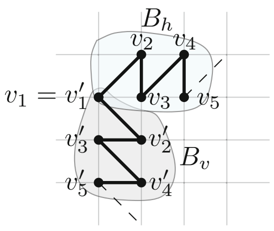


Fig. 2. Border gadget B that is rigid and regular. The horizontal component is denoted by B_h and the vertical one by B_v.

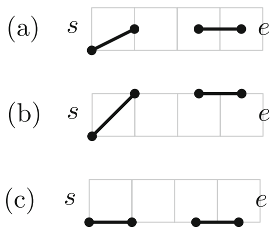


Fig. 3. The switch s is linked with edge e; (a) shows the input embedding, in (b) e is pulled up and in (c) it is pulled down

3. if we place a vertex v with two non-integer coordinates, i.e., in the interior of a grid cell, then its position in any valid embedding is in that grid cell (including its boundary).

Basic Building Blocks. We will frequently make use of two basic building blocks that rely on the above grid properties.

The first one is a *switch*, i.e., an edge that has exactly two valid embeddings. Let $s = uv$ be an edge within a single grid cell where u is placed on a grid point and v on a non-incident grid edge. We call u the *fixed* and v the *free* vertex of s. Assume that u is in the lower left corner and v on the right edge of the grid cell. Then in any valid d-schematization s is either horizontal or diagonal, see Fig. 3 for an example.

The second basic concept is *linking* of vertices. We can synchronize two vertices in different and even non-adjacent cells of the grid by assigning them the same x- or y-coordinate. We call two vertices u and v *linked*, if in π either $x_\pi(u) = x_\pi(v)$ or $y_\pi(u) = y_p(v)$. Then the orthogonal order requires that u and v remain linked in any valid embedding. This concept allows us to transmit information on local embedding choices over distances. Two edges $e_i = uu'$ and $e_j = vv'$ are *linked* if there is a vertex of e_i that is linked to a vertex of e_j. We use linking of edges in combination with switches. Namely, we link a switch s via its free vertex with another edge e, as illustrated in Fig. 3. Then the choice of the embedding of s determines one of the two coordinates of the linked vertices in e. In the case depicted in Fig. 3 the switch s determines the y-coordinate of both vertices of e; we say that s *pulls* e up (Fig. 3(b)) or down (Fig. 3(c)). Such a switch is called a *vertical switch*. Analogously, edges can be pulled to the left and to the right by a *horizontal switch*.

Variable Gadgets. The variable gadget for a variable x is a simple structure consisting of a horizontal switch and a number of linked *connector vertices* on consecutive grid lines below the switch, one for each appearance of x or $\neg x$ in a clause of φ. We denote the number of appearances as $t(x)$. All connector vertices share the same x-coordinate in π. Each one will be connected to a clause with a diagonal *connector edge*. A connector edge spans the same number of grid cells horizontally and vertically and hence can only be embedded at a slope of 45°. In order to have the correct slope, the positions of both

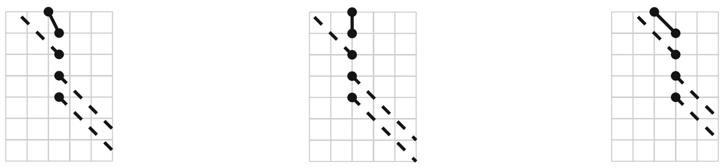

(a) Input embedding (b) State `true` (c) State `false`

Fig. 4. A variable gadget with three connector vertices

endpoints within their grid cell must be the same. The upper vertices will connect to the gadgets of the negative clauses and the lower vertices to the gadgets of the positive clauses. The variable gadget has two states, one in which the switch pulls all vertices to the left (defined as `true`), and one in which it pulls them to the right (defined as `false`). Figure 4 shows an example. A variable gadget g_x for a variable x takes up one grid cell in width and $t(x)+1$ grid cells in height.

Clause Gadgets. The general idea behind the clause gadget is to create a set of paths whose embedding is influenced by three connector edges. Each of those edges carries the truth value of the connected variable gadget. The gadget consists of three diagonal edges e_2, e_3, e_4, two vertical edges e_1 and e_5 and four switches s_1, s_2, s_3, s_4, see Fig. 5. For positive clauses, we want that if all its connector edges are pulled to the right, i.e., all literals are `false`, there is no valid embedding of the clause gadget. This is achieved by placing the edges of the gadget in such a way that there are certain points, called *critical points*, where two different non-adjacent edges can possibly place a vertex. Further, we ensure with the help of switches that each of the three diagonal edges of the clause gadget has exactly one vertex embedded on a critical point in a valid embedding.

Key to the gadget is the *critical edge* e_3 that is embedded with one fixed vertex on a grid point and one loose vertex in the grid cell A to the top left of the fixed vertex. Edge e_3 is linked with switches s_2 and s_4. These switches ensure that the loose vertex must be placed on a free corner of the grid cell. The three free corners are all critical points. It is clear that there is a valid embedding of e_3 if and only if one of the three critical points is available.

We use the remaining edges of the gadget to block one of the critical points of A for each literal that is `false`. The lower vertex of the middle connector edge is placed just to the left of the upper left corner of A such that it occupies a critical point if it is pulled to the right. Both of the left and right connector edges have another linked vertical edge e_1 and e_5 appended each. Now if e_1 is pushed to the right by its connector edge, then edge e_2 is pushed upward since e_1 and e_2 share a critical point. Due to switch s_1 edge e_2 blocks the lower left critical point of A in that case. Similarly, edge e_4 blocks the upper right critical point of A if the third literal is `false`.

The clause gadget for a negative clause works analogously such that a critical point is blocked for each connector edge that is pulled to the left instead of to the right. It corresponds basically to the positive clause gadget rotated by 180°.

Lemma 2. *A clause gadget has a valid embedding if and only if at least one of its literals is* `true`.

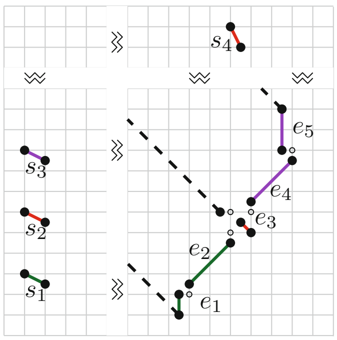


Fig. 5. Sketch of a clause gadget. Critical points are marked as white disks.

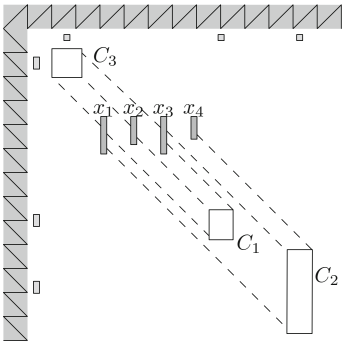


Fig. 6. Sketch of the gadgets for the instance φ from Fig. 1

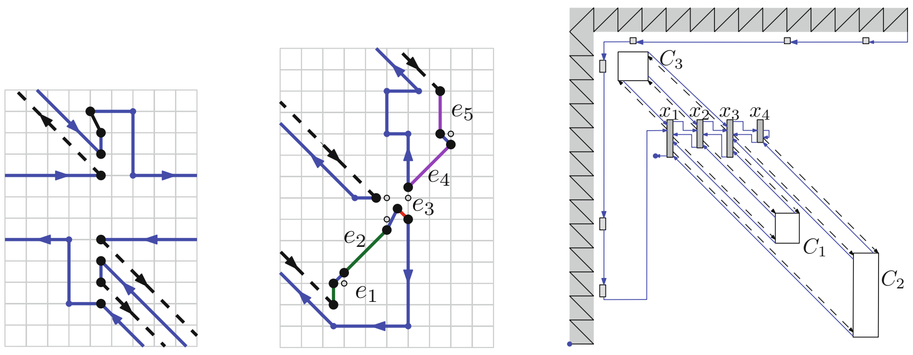
The size of a clause gadget (not considering the switches) depends on the horizontal distances Δ_1, Δ_2 between the left and middle connector edge and between the middle and right connector edge, respectively. The width of a gadget is 6 and its height is $\Delta_1 + \Delta_2 - 5$.

Gadget Placement. The top and left part of our construction is occupied by the border gadget that defines the grid. The overall shape of the remaining construction is similar to a slanted version of the standard embedding of an instance of MONOTONE PLANAR 3-SAT as in Fig. 1.

We place the individual variable gadgets horizontally aligned in the center of the drawing. Since variable gadgets do not move vertically there is no danger of accidentally linking vertices of different variable gadgets by placing them at the same y-coordinates. For the clause gadgets to work as desired, we must make sure that any two connector edges to the same clause gadget have a minimum distance of seven grid cells. This can easily be achieved by spacing the variable gadgets horizontally so that connector edges of adjacent variables cannot come too close to each other.

To avoid linking between different clause gadgets or clause gadgets and variable gadgets, we place each of them in its own x- and y-interval of the grid with the negative clauses to the top left of the variables and the positive clauses to the bottom right. The switches of each clause gadget are placed at the correct positions just next to the border gadget. Figure 6 shows a sketch of the full placement of the gadgets for a MONOTONE PLANAR 3-SAT instance φ. Since the size of each gadget is polynomial in the size of the formula φ, so is the whole construction. The above construction finally yields the following theorem.

Theorem 1. *The* 2*-regular* UNION OF PATHS SCHEMATIZATION PROBLEM *is NP-hard.*



(a) Variable gadget (b) Clause gadget (c) Sketch of full instance

Fig. 7. Augmented gadget versions. Additional path edges are highlighted in blue

3.2 Hardness of the Path Schematization Problem

Finally, to prove that 2-regular PSP is also NP-hard we have to show that we can augment the union of paths constructed above to form a single simple path that still has the property that it has a valid embedding if and only if the corresponding Monotone Planar 3-Sat formula φ is satisfiable.

The general idea is to start the path at the lower end of the border gadget and then collect all the switches next to the border gadget. From there we enter the upper parts of the variable gadgets and walk consecutively along all the connector edges into the negative clause gadgets. Once all negative connectors have been traversed, the path continues along the positive connectors and into the positive clauses. The additional edges and vertices must be placed such that they do not interfere with any functional part of the construction.

The only major change is that we double the number of connector vertices in each variable gadget and add a parallel dummy edge for each connector edge. That way we can walk into a clause gadget along one edge and back into the variable gadget along the other edge. We also need a clear separation between the negative and positive connector vertices, i.e., the topmost positive connector vertices of all variable gadgets are assigned the same y-coordinate and we adjust the required spacing of the variable gadgets accordingly. Figure 7(a) shows an example of an augmented variable gadget with one positive and two negative connector edges.

The edges e_1–e_5 of each clause gadget are inserted into the path in between the leftmost connector edge and its dummy edge and in between the rightmost connector edge and its dummy edge, respectively. The details are illustrated in Fig. 7(b). It is clear that by this construction we obtain a single simple path P that contains all the gadgets of our previous reduction. Moreover, the additional edges are embedded such that they do not interfere with the gadgets themselves. Rather they can move along with the flexible parts without occupying grid points that are otherwise used by the gadgets. Hence we can summarize:

Theorem 2. *The 2-regular* Path Schematization Problem *is NP-hard.*

4 Hardness of d-Regular PSP for $d > 2$

In the previous section we have established that 2-regular PSP is NP-hard. Here we show that d-regular PSP remains NP-hard for all $d > 2$ by modifying the gadgets of our proof for $d = 2$.

An obvious problem for adapting the gadgets is that due to the presence of more than one diagonal slope the switches do not work any more for uniform grid cells. In a symmetric grid, we cannot ensure that the free vertex of a switch is always on a grid point in any valid d-schematization. This means that we need to devise a different grid in order to make the switches work properly. We construct a new border gadget that induces a grid with cells of uniform width but non-uniform heights. We call the cells of this grid *minor cells* and form groups of $d-1$ vertically consecutive cells to form *meta cells*. Then all meta cells again have uniform widths and heights.

The $d-1$ different heights of the minor cells are chosen such that the segments connecting the upper left corner of a meta cell to the lower right corners of its minor cells have exactly the slopes that are multiplies of $(90/d)^\circ$ and lie strictly between 0° and 90°. This restores the functionality of switches whose fixed vertex is placed on the upper left corner of a meta cell and whose free vertex is on a non-adjacent grid line of the same meta cell. Although the underlying grid differs the functionality of variable and clause gadgets as well as the overall structure stays the same. This yields the following theorem. The detailed proof is included in the full version of this paper [10].

Theorem 3. *The d-regular* PATH SCHEMATIZATION PROBLEM *is NP-hard for any $d > 2$.*

5 Design and Evaluation of a MIP for d-Regular PSP

PSP can be formulated as a MIP that is similar to a MIP model for drawing octilinear metro maps [16]. With a MIP we can not only decide PSP but also optimize the output for resemblance with the input path if there is a feasible solution. We achieve this by (i) minimizing for each edge the deviation between its input slope and its output slope, and (ii) by minimizing the total path length subject to a certain minimum length for every edge. See the full version of this paper [10] for the detailed MIP formulation.

We have implemented the MIP for simple input paths and evaluate its performance by schematizing 1 000 quickest routes in the German road network, where we select source and target nodes uniformly at random. The average path length is 1020.7 nodes, hence, a schematization would yield a route sketch with far too much detail. Therefore, in a preprocessing step, we simplify the path using the Douglas-Peucker algorithm [8] but keep all important nodes such as points of road and road category changes. Moreover, we shortcut self-intersecting subpaths whose lengths are below a threshold. This is to remove over- or underpasses near slip roads of highways, which are considered irrelevant for route sketches or would be depicted as an icon rather than a loop. Figure 8 illustrates an example of a route sketch obtained by our MIP. It clearly illustrates how a schematic route sketch, unlike the geographic map, is able to depict both high-detail and

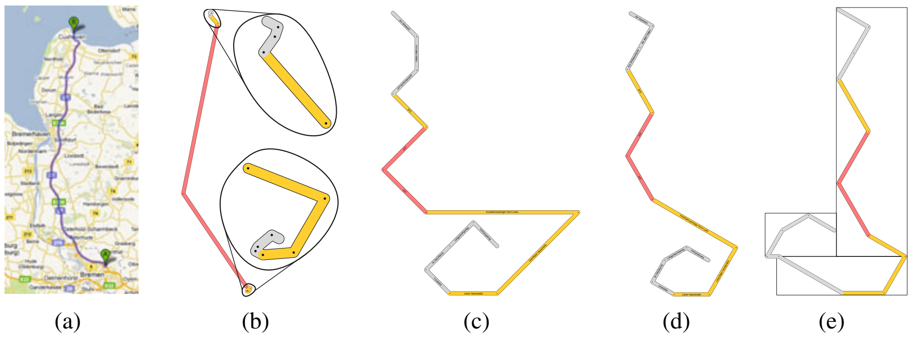


Fig. 8. A sample route from Bremen to Cuxhaven in Germany: (a) output of Google Maps, (b) simplified route after preprocessing, (c) output of our MIP for $d = 2$, (d) output of our MIP for $d = 3$, and (e) output of the algorithm by Delling et al. [7]

low-detail parts of the route in a single picture. Comparing Figures 8(c) and 8(d) indicates that using the parameter $d = 2$ for the schematization process yields route sketches with a very high level of abstraction while using $d = 3$ results in route sketches which resemble the overall shape of the route much better. For example, the route sketch in Fig. 8(c) seems to suggest that there is a 90° turn while driving on the highway but this is not the case. The route sketch depicted in Fig. 8(c) resembles the original route much better. Generally, we found that using the parameter $d = 3$ yields route sketches with a higher degree of readability.

For comparison, we show the output computed by the method of Delling et al. [7] in Fig. 8(e). Recall that their original algorithm takes as input only monotone paths. They describe, however, a heuristic approach to schematize non-monotone paths by subdividing them into maximal monotone subpaths that are subsequently merged into a single route sketch. In order to avoid intersections of the bounding boxes of the monotone subpaths, additional edges of appropriate length must be inserted into the path; the orthogonal order is preserved only within the monotone subpaths. This may lead to undesired effects in sketches of non-monotone routes, see Fig. 8(e).

Evaluation. We performed two experiments that were executed on a single core of an AMD Opteron 2218 processor running Linux 2.6.27.23. The machine is clocked at 2.6 GHz, has 16 GiB of RAM and 2×1 MiB of L2 cache. Our implementation is written in C++ and was compiled with GCC 4.3.2, using optimization level 3. As MIP solver we use Gurobi 3.0.1.

In the first experiment we examine the performance of our MIP for $d = 3$ subject to the amount of detail of the route, as controlled by the distance threshold ε between input and output path in the path simplification step. We also set the threshold for removing self-intersecting subpaths to ε. Table 1(a) reports our experimental results for values of ε between 2^{-1} and 2^{-5}.

We observe that with decreasing ε, the lengths of the paths increase from 20.04 nodes to 32.81 nodes on average. This correlates with the running time of our MIP which

Table 1. Results obtained by running 1 000 random queries in the German road network and schematizing the simplified paths with our MIP. The tables reports average path lengths, percentage of infeasible instances, average number of iterations, and average running times.

(a) Varying ε with $d = 3$.

ε	~length	% inf	~iter	~time [ms]
2^{-1}	20.04	0.1	1.28	801.61
2^{-2}	20.50	0.1	1.29	621.66
2^{-3}	21.81	0.1	1.29	649.49
2^{-4}	25.16	0.2	1.26	772.57
2^{-5}	32.81	0.3	1.27	1102.20

(b) Varying d with $\varepsilon = 2^{-3}$.

d	% inf	~iter	~time [ms]
1	51.3	1.03	107.36
2	0.7	1.19	363.49
3	0.1	1.29	649.49
4	0.1	1.25	1075.82
5	0.1	1.21	1347.86

is between 801.61 ms and 1 102.20 ms. Note that in practice a value of $\varepsilon = 2^{-3}$ is a good compromise between computation time and amount of detail. Further, we observe that adding planarity constraints in a lazy fashion pays off since we need less than 1.3 iterations on average. constraints at all. The number of infeasible instances, i. e., paths without a valid 3-schematization, is 0.1 %. schematization respecting the orthogonal order of the nodes together with the minimum edge lengths, or, if the input path is not simple.

The second experiment evaluates the performance of our MIP when using different values of d, see Table 1(b). We fix $\varepsilon = 2^{-3}$. While for rectilinear drawings with $d = 1$ we need 107.36 ms to compute a solution, the running time increases to 1 347.86 ms when using $d = 5$. Interestingly, more than half of the paths do not have a valid rectilinear schematization. By allowing one additional diagonal slope ($d = 2$), the number of infeasible instances significantly decreases to 0.7 % and for $d = 3$ only 0.1 %, i. e., a single instance, is infeasible.

6 Conclusion

Motivated by drawing route sketches in road networks, we studied the d-regular path schematization problem (PSP), which has two main goals: To preserve the user's mental map through maintaining the orthogonal order, and to reduce the visual complexity using restricted edge slopes given by integer multiples of $(90/d)^\circ$. We analyzed the complexity of the problem and showed that PSP is NP-hard for $d \geq 2$, thus, closing the complexity gap between the hardness result of Brandes and Pampel [4] for $d = 1$, and the efficient algorithm of Delling et al. [7] for monotone paths. In the second part of this work, we modeled and implemented the PSP as a mixed integer linear program (MIP). An experimental evaluation with real-world data of the German road network showed that we are indeed able to compute visually appealing route sketches within approximately one second for values of $d \leq 5$.

Using ideas of Nöllenburg and Wolff [16], our MIP can be further generalized to handle both non-simple paths and general graphs, e. g., a set of alternative routes.

Acknowledgments. We thank an anonymous referee for helpful comments.

References

1. Agrawala, M., Stolte, C.: Rendering effective route maps: Improving usability through generalization. In: Proc. 28th Ann. Conf. Computer Graphics and Interactive Techniques (SIGGRAPH 2001), pp. 241–249. ACM, New York (2001)
2. de Berg, M., Khosravi, A.: Finding perfect auto-partitions is NP-hard. In: Proc. 25th European Workshop Comput. Geom (EuroCG 2008), pp. 255–258 (2008)
3. Brandes, U., Eiglsperger, M., Kaufmann, M., Wagner, D.: Sketch-driven orthogonal graph drawing. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 1–11. Springer, Heidelberg (2002)
4. Brandes, U., Pampel, B.: On the hardness of orthogonal-order preserving graph drawing. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 266–277. Springer, Heidelberg (2009)
5. Branke, J.: Dynamic graph drawing. In: Kaufmann, M., Wagner, D. (eds.) Drawing Graphs. LNCS, vol. 2025, pp. 228–246. Springer, Heidelberg (2001)
6. Cabello, S., de Berg, M., van Dijk, S., van Kreveld, M., Strijk, T.: Schematization of road networks. In: Proc. 17th Annual Symposium on Computational Geometry (SoCG 2001), June 3–5, pp. 33–39. ACM Press, New York (2001)
7. Delling, D., Gemsa, A., Nöllenburg, M., Pajor, T.: Path schematization for route sketches. In: Kaplan, H. (ed.) SWAT 2010. LNCS, vol. 6139, pp. 285–296. Springer, Heidelberg (2010)
8. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica 10(2), 112–122 (1973)
9. Dwyer, T., Koren, Y., Marriott, K.: Stress majorization with orthogonal ordering constraints. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 141–152. Springer, Heidelberg (2006)
10. Gemsa, A., Nöllenburg, M., Pajor, T., Rutter, I.: On d-regular Schematization of Embedded Paths. Tech. Rep. 2010-21, Faculty of Informatics, Karlsruhe Institute of Technology (2010), `http://digbib.ubka.uni-karlsruhe.de/volltexte/1000020426`
11. Hayashi, K., Inoue, M., Masuzawa, T., Fujiwara, H.: A layout adjustment problem for disjoint rectangles preserving orthogonal order. In: Whitesides, S.H. (ed.) GD 1998. LNCS, vol. 1547, pp. 183–197. Springer, Heidelberg (1999)
12. Merrick, D., Gudmundsson, J.: Path simplification for metro map layout. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 258–269. Springer, Heidelberg (2007)
13. Misue, K., Eades, P., Lai, W., Sugiyama, K.: Layout adjustment and the mental map. J. Visual Languages and Computing 6(2), 183–210 (1995)
14. Neyer, G.: Line simplification with restricted orientations. In: Dehne, F., Gupta, A., Sack, J.-R., Tamassia, R. (eds.) WADS 1999. LNCS, vol. 1663, pp. 13–24. Springer, Heidelberg (1999)
15. Nöllenburg, M.: Automated drawing of metro maps. Tech. Rep. 2005-25, Fakultät für Informatik, Universität Karlsruhe (2005), `http://digbib.ubka.uni-karlsruhe.de/volltexte/1000004123`
16. Nöllenburg, M., Wolff, A.: Drawing and Labeling High-Quality Metro Maps by Mixed-Integer Programming. IEEE Transactions on Visualization and Computer Graphics (2010), `http://dx.doi.org/10.1109/TVCG.2010.81` (preprint available online)
17. Wolff, A.: Drawing subway maps: A survey. Informatik - Forschung und Entwicklung 22(1), 23–44 (2007)

Upward Point-Set Embeddability

Markus Geyer[1], Michael Kaufmann[1], Tamara Mchedlidze[2], and Antonios Symvonis[2]

[1] Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Tübingen, Germany
{geyer,mk}@informatik.uni-tuebingen.de

[2] Dept. of Mathematics, National Technical University of Athens, Athens, Greece
{mchet,symvonis}@math.ntua.gr

Abstract. We study the problem of Upward Point-Set Embeddability, that is the problem of deciding whether a given upward planar digraph D has an upward planar embedding into a point set S. We show that any switch tree admits an upward planar straight-line embedding into any convex point set. For the class of k-switch trees, that is a generalization of switch trees (according to this definition a switch tree is a 1-switch tree), we show that not every k-switch tree admits an upward planar straight-line embedding into any convex point set, for any $k \geq 2$. Finally we show that the problem of Upward Point-Set Embeddability is NP-complete.

1 Introduction

A *planar straight-line embedding* of a graph G into a point set S is a mapping of each vertex of G to a distinct point of S and of each edge of G to the straight-line segment between the corresponding end-points so that no two edges cross each other. Gritzmann *et al.* [9] proved that outerplanar graphs is the class of graphs that admit a planar straight-line embedding into every point set in general position or in convex position. Efficient algorithms are known to embed outerplanar graphs [3] and trees [4] into any point set in general or in convex position. From the negative point of view, Cabello [5] proved that the problem of deciding whether there exists a planar straight-line embedding of a given graph G into a point set P is NP-hard even when G is 2-connected and 2-outerplanar. For upward planar digraphs, the problem of constructing upward planar straight-line embeddings into point sets was studied by Giordano et al. [8], later on by Binucci et al. [2] and recently by Angelini et al. [1]. While some positive and negative results are known for the case of upward planar digraphs, the complexity of testing upward planar straight-line embeddability into point sets has not been known.

In this paper we continue the study of the problem of upward planar straight-line embedding of directed graphs into a given point set. Our results include:

– We extend the positive results given in [1,2] by showing that any directed switch tree admits an upward planar straight-line embedding into every point set in convex position.

- We study directed k-switch trees, a generalization of switch trees (a 1-switch tree is exactly a switch tree). From the construction given in [2] (Theorem 5), we know that for $k \geq 4$ not every k-switch tree admits an upward planar straight-line embedding into any convex point set. Then we fill the gap for 2 and 3-switch trees, by showing that, for any $k \geq 2$ there is a class of k-switch trees $\mathcal{T}_n^k$, and a point set S in convex position, such that any $T \in \mathcal{T}_n^k$ does not admit an upward planar straight-line embedding into S.
- We study the computational complexity of the upward embeddability problem. More specifically, given a n vertex upward planar digraph G and a set of n points on the plane S, we show that deciding whether there exists an upward planar straight-line embedding of G so that its vertices are mapped to the points of S is NP-Complete. The decision problem remains NP-Complete even when G has a single source and the longest simple cycle of G has length four and, moreover, S is an m-convex point set, for some integer $m > 0$.

Due to space constraints we sketch or omit some of the proofs; for the detailed version see [7].

2 Preliminaries

We mostly follow the terminology of [2]. Next, we give some definitions that are used throughout this paper.

Let l be a line on the plane, which is not parallel to the x-axis. We say that point *p lies to the right of l* (resp., *to the left of l*) if p lies on a semi-line that originates on l, is parallel with the x-axis and is directed towards $+\infty$ (resp., $-\infty$). Similarly, if l is a line on the plane, which is not parallel to the y-axis, we say that point *p lies above l* (resp., *below l*) if p lies on a semi-line that originates on l, is parallel with the y-axis and is directed towards $+\infty$ (resp., $-\infty$).

A *point set in general position*, or *general point set*, is a point set such that no three points lie on the same line and no two points have the same y-coordinate. The *convex hull $H(S)$* of a point set S is the point set that can be obtained as a convex combination of the points of S. A *point set in convex position*, or *convex point set*, is a point set such that no point is in the convex hull of the others. Given a point set S, we denote by $b(S)$ and by $t(S)$ the lowest and the highest point of S, respectively. A *one-sided convex point set S* is a convex point set in which $b(S)$ and $t(S)$ are adjacent in the border of $H(S)$. A convex point set which is not one-sided, is called a *two-sided convex point set.* In a convex point set S, the subset of points that lie to the left (resp. right) of the line through $b(S)$ and $t(S)$ is called the *left (resp. right) part of S*. A one-sided convex point set S is called *left-heavy* (resp., *right-heavy*)convex point set if all the points of S lie to the left (resp., to the right) of the line through $b(S)$ and $t(S)$. Note that, a one-sided convex point set is either a left-heavy or a right-heavy convex point set.

Consider a point set S and its convex hull $H(S)$. Let $S_1 = S \setminus H(S)$, $\ldots$, $S_m = S_{m-1} \setminus H(S_{m-1})$. If m is the smallest integer such that $S_m = \emptyset$, we say that S is an *m-convex point set.* A subset of points of a convex point set S is called *consecutive* if its points appear consecutive as we traverse the convex hull of S in the clockwise or counterclockwise direction.

The graphs we study in this paper are directed. By (u, v) we denote an arc directed from u to v. A *switch-tree* is a directed tree T, such that, each vertex of T is either a source of a sink. Note that the longest directed path of a switch-tree has length one[1]. Based on the length of the longest path, the class of switch trees can be generalized to that of k-switch trees. A *k-switch tree* is a directed tree, such that its longest directed path has length k. So, a switch tree is a 1-switch tree. A digraph D is called *path-DAG*, if its underlying graph is a simple path. A *monotone path* $(v_1, v_2, \ldots, v_k)$ is a path-DAG containing arcs (v_i, v_{i+1}), $1 \leq i \leq k-1$.

An *upward planar directed graph* is a digraph that admits a planar drawing where each edge is represented by a curve monotonically increasing in the y-direction. An *upward straight-line embedding* (*UPSE* for short) of a graph into a point set is a mapping of each vertex to a distinct point and of each arc to a straight-line segment between its end-points such that no two arcs cross and each arc (u, v) has $y(u) < y(v)$. The following results were presented in [2] and are used in this paper.

Lemma 1 (Binucci at al. [2]). *Let T be an n-vertex tree-DAG and let S be any convex point set of size n. Let u be any vertex of T and let $T_1, T_2, \ldots, T_k$ be the subtrees of T obtained by removing u and its incident edges from T. In any UPSE of T into S, the vertices of T_i are mapped into a set of consecutive points of S, for each $i = 1, 2, \ldots, k$.*

Theorem 1 (Binucci at al. [2]). *For every odd integer $n \geq 5$, there exists a $(3n+1)$-vertex directed tree T and a convex point set S of size $3n+1$ such that T does not admit an UPSE into S.*

3 Embedding a Switch-Tree into a Point Set in Convex Position

In this section we enrich the positive results presented in [1,2] by proving that, any switch-tree has an UPSE into any point set in convex position. During the execution of the algorithms, presented in the following lemmata, which embed a tree T into a point set S, a *free point* is a point of S to which no vertex of T has been mapped yet. The following two lemmata treat the simple case of a one-sided convex point set and are immediate consequences of a result by Heath et al. [10].

Lemma 2. *Let T be a switch-tree, r be a sink of T, S be a one-sided convex point set so that $|S| = |T|$, and p be S's highest point. Then, T admits an UPSE into S so that vertex r is mapped to point p.* □

[1] The *length* of a directed path is the number of arcs in the path.

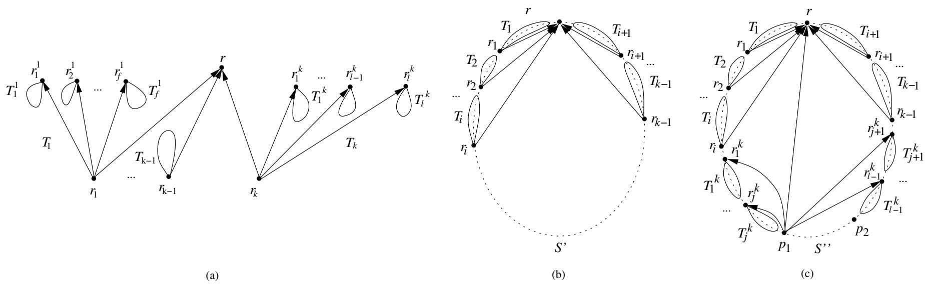


Fig. 1. The construction of Lemma 4

Lemma 3. *Let T be a switch-tree, r be a source of T, S be a one-sided convex point set so that $|S| = |T|$, and p be S's lowest point. Then, T admits an UPSE into S so that vertex r is mapped to point p.* □

Now we are ready to proceed to the main result of the section.

Theorem 2. *Let T be a switch-tree and S be a convex point set such that $|S| = |T|$. Then, T admits an UPSE into S.*

The proof of the theorem is based on the following lemma, which extends Lemma 2 from one-sided convex point sets to convex point sets.

Lemma 4. *Let T be a switch-tree, r be a sink of T, S be a convex point set such that $|S| = |T|$. Then, T admits an UPSE into S so that vertex r is mapped to the highest point of S.*

Proof. Let $T_1, \dots, T_k$ be the sub-trees of T that are connected to r by an edge (Figure 1.a) and let $r_1, \dots, r_k$ be the vertices of $T_1, \dots, T_k$, respectively, that are connected to r. Observe that, since T is a switch tree and r is a sink, vertices $r_1, \dots, r_k$ are sources.

We draw T on S as follows. We start by placing the trees $T_1, T_2, \dots$ on the left side of the point set S as long as they fit, using the highest free points first. This can be done in an upward planar fashion by Lemma 3 (Figure 1.b). Assume that T_i is the last placed subtree. Then, we continue placing the trees $T_{i+1}, \dots, T_{k-1}$ on the right side of the point set S. This can be done due to Lemma 3. Note that the remaining free points are consecutive point of S, denote these points by S'. To complete the embedding we draw T_k on S'. Let $T_1^k, \dots, T_l^k$ the subtrees of T_k, that are connected to r_k by an arc. Let also $r_1^k, \dots, r_l^k$ be the vertices of $T_1^k, \dots, T_l^k$, respectively, that are connected to r_k (Figure 1.a). Note that $r_1^k, \dots, r_l^k$ are all sinks. We start by drawing $T_1^k, T_2^k, \dots$ as long as they fit on the left side of point set S', using the highest free points first. This can be done in an upward planar fashion by Lemma 2. Assume that T_j^k is the last placed subtree (Figure 1.c). Then, we continue on the right side of the point set S' with the trees $T_{j+1}^k, \dots, T_{l-1}^k$. This can be done again by Lemma 2. Note that there are exactly $|T_l^k| + 1$ remaining free points since we have not yet drawn T_l^k and vertex

r_k of T_k. Denote by S'' the remaining free points and note that S'' consists of consecutive points of S. If S'' is a one-sided point set then we can proceed by using the Lemma 2 again and the result follows trivially. Assume now that S'' is a two-sided convex point set and let p_1 and p_2 be the highest points of S'' on the left and on the right, respectively. W.l.o.g., let $y(p_1) < y(p_2)$. Then, we map r_k to p_1. By using the lemma recursively, we can draw T_l^k on $S'' \setminus \{p_1\}$ so that r_l^k is mapped to p_2. The proof is completed by observing that all edges connecting r_k to $r_1^k, \dots, r_l^k$ and $r_1, \dots, r_k$ to r are upward and do not cross each other. □

4 *K*-Switch Trees

Binucci et al. [2] (see also Theorem 1) presented a class of trees and corresponding convex point sets, such that any tree of this class does not admit an UPSE into its corresponding point set.

The $(3n+1)$-size tree T constructed in the proof of Theorem 1[2] has the following structure. It consists of: (i) one vertex r of degree three, (ii) three monotone paths of n vertices: $P_u = (u_n, u_{n-1}, \dots, u_1)$, $P_v = (v_1, v_2, \dots, v_n)$, $P_w = (w_1, w_2, \dots, w_n)$, (iii) arcs (r, u_1), (v_1, r) and (w_1, r).

The $(3n+1)$-convex point set S, used in the proof of Theorem 1[2], consists of two extremal points on the y-direction, $b(S)$ and $t(S)$, the set L of $(3n-1)/2$ points $l_1, l_2, \dots, l_{(3n-1)/2}$, comprising the left side of S and the set R of $(3n-1)/2$ points $r_1, r_2, \dots, r_{(3n-1)/2}$, comprising the right side of S. The points of L and R are located so that $y(b(S)) < y(r_1) < y(l_1) < y(r_2) < y(l_2) < \dots < y(r_{(3n-1)/2}) < y(l_{(3n-1)/2}) < y(t(S))$.

Note that the $(3n+1)$-node tree T described above is a $(n-1)$-switch tree. Hence a straightforward corollary of Theorem 1[2] is the following statement.

Corollary 1. *For $k \geq 4$, there exists a k-switch tree T and a convex point set S of the same size, such that T does not admit an UPSE into S.*

From Section 3, we know that any switch tree T, i.e. a 1-switch tree, admits an UPSE into any convex point set. The natural question raised by this result and Corollary 1 is whether an arbitrary 2-switch or 3-switch tree has an UPSE into any convex point set. This question is resolved by the following theorem.

Theorem 3. *For any $n \geq 5$ and for any $k \geq 2$, there exists a class $\mathcal{T}_n^k$ of $3n+1$-vertex k-switch trees and a convex point set S, consisting of $3n+1$ points, such that any $T \in \mathcal{T}_n^k$ does not admit an UPSE into S.*

Proof. For any $n \geq 5$ we construct the following class of trees. Let P_u be an n-vertex path-DAG on the vertex set $\{u_1, u_2, \dots, u_n\}$, enumerated in the order they are presented in the underlying undirected path of P_u, and such that arcs (u_3, u_2), (u_2, u_1) are present in P_u. Let also P_v and P_w be two n-vertex path-DAGs on the vertex sets $\{v_1, v_2, \dots, v_n\}$ and $\{w_1, w_2, \dots, w_n\}$ respectively, enumerated in the order they are presented in the underlying undirected path of P_v and P_w, and such that arcs $(v_1, v_2), (v_2, v_3)$ and $(w_1, w_2), (w_2, w_3)$ are present in P_v and P_w, respectively. Let $T(P_u, P_v, P_w)$ be a tree consisting of P_u, P_v, P_w, vertex r and arcs $(r, u_1), (v_1, r), (w_1, r)$.

Let $\mathcal{T}_n^k = \{T(P_u, P_v, P_w) \mid$ the longest directed path in P_u, P_v and P_w has length $k\}$, $k \geq 2$. So, $\mathcal{T}_n^k$ is a class of $3n + 1$-vertex k-switch trees. Let S be a convex point set as described in the beginning of the section. Then any $T \in \mathcal{T}_n^k$ does not admit an UPSE into point set S, see [7] for the detailed proof. □

5 Upward Planar Straight-Line Point Set Embeddability is NP-Complete

In this section we examine the complexity of testing whether a given n-vertex upward planar digraph G admits an UPSE into a point set S. We show that the problem is NP-complete even for a single source digraph G having longest simple cycle of length at most 4. This result is optimal for the class of cyclic graphs[2], since Angelini et al. [1] showed that every single-source upward planar directed graph with no cycle of length greater than three admits an UPSE into every point set in general position.

Theorem 4. *Given an n-vertex upward planar digraph G and a planar point set S of size n in general position, the decision problem of whether there exists an UPSE of G into S is NP-Complete. The decision problem remains NP-Complete even when G has a single source and the longest simple cycle of G has length at most 4 and, moreover, S is an m-convex point set for some $m > 0$.*

Proof. The problem is trivially in NP. In order to prove the NP-completeness, we construct a reduction from the 3-Partition problem.

Problem: 3-Partition
Input: A bound $B \in \mathbb{Z}^+$, and a set $A = \{a_1, \dots, a_{3m}\}$ with $a_i \in \mathbb{Z}^+$, $\frac{B}{4} < a_i < \frac{B}{2}$.
Output: m disjoint sets $A_1, \dots, A_m \subset A$ with $|A_i| = 3$ and $\sum_{a \in A_i} a = B$, $1 \leq i \leq m$.

We use the fact that 3-Partition is a strongly NP-hard problem, i.e. it is NP-hard even if B is bounded by a polynomial in m [6]. Let A and B be the set of the $3m$ positive integers and the bound, respectively, that form the instance (A, B) of the 3-Partition problem. Based on A and B, we show how to construct an upward planar digraph G and a point set S such that G has an UPSE on point set S if and only if the instance (A, B) of the 3-partition problem has a solution.

We first show how to construct G (see Figure 2.a for illustration). We start the construction of G by first adding two vertices s and t. Vertex s is the single source of the whole graph. We then add m disjoint paths from s to t, each of length two. The degree-2 vertices of these paths are denoted by u_i, $i = 1, \dots, m$. For each $a \in A$, we construct a monotone directed path P_i of length a that has a new vertices and s at its source. Totally, we have $3m$ such paths $P_1, \dots, P_{3m}$.

[2] A digraph is *cyclic* if its underling undirected graph contains at least one cycle.

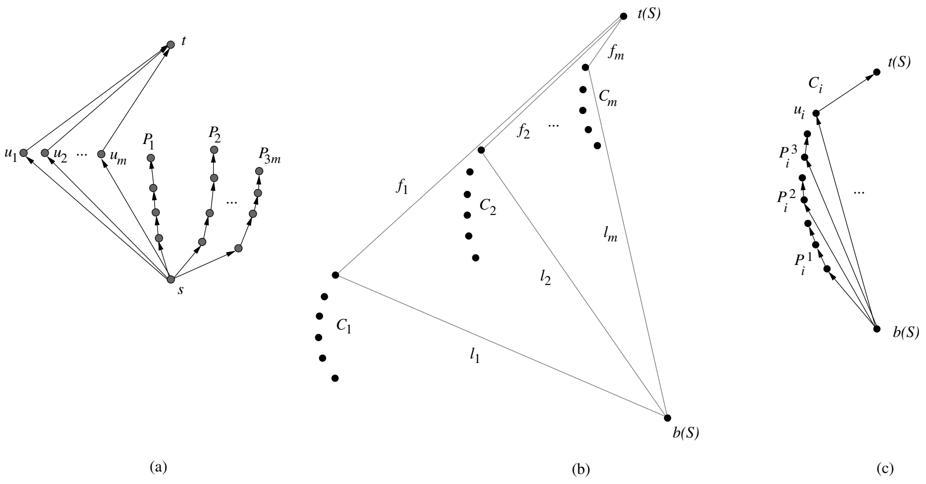


Fig. 2. (a) The graph G of the construction used in the proof of NP-completeness. (b) The point set S of the construction. (c) An UPSE of G on S.

We proceed to the construction of point set S. Let $b(S)$ and $t(S)$ be the lowest and the highest points of S (see Figure 2.b). In addition to $b(S)$ and $t(S)$, S also contains m one-sided convex point sets $C_1, \ldots, C_m$, each of size $B+1$, so that the points of S satisfy the following properties:

- $C_i \cup \{b(S), t(S)\}$ is a left-heavy convex point set, $i \in \{1, \ldots, m\}$.
- The points of C_{i+1} are higher than the points of C_i, $i \in \{1, \ldots, m-1\}$.
- Let l_i be the line through $b(S)$ and $t(C_i)$, $i \in \{1, \ldots, m\}$. $C_1, \ldots, C_i$ lie to the left of line l_i and $C_{i+1}, \ldots, C_m$ lie to the right of line l_i.
- Let f_i be the line through $t(S)$ and $t(C_i)$, $i \in \{1, \ldots, m\}$. C_j, $j \geq i$, lie to the right of line f_i.
- $\{t(C_i) : i = 1, \ldots, m\}$ is a left-heavy convex point set.

The next statement follows from the properties of point set S.

Statement 1. *Let C_i be one of the left-heavy convex point sets comprising S and let $x \in C_j$, $j > i$. Then, set $C_i \cup \{b(S), x\}$ is also a left-heavy convex point set, with $b(S)$ and x consecutive on its convex hull.* □

Statement 2. *We can construct a point set S that satisfies all the above requirements so that the area of S is polynomial on B and m.*

Proof of Statement: For each $i \in \{0, \ldots, m-1\}$ we let C_{m-i} to be the set of $B+1$ points

$$C_{m-i} = \left\{(-j - i(B+2), j^2 - (i(B+2))^2) \mid \; j = 1, 2, \ldots, B+1 \;\right\}$$

Then, we set the lowest point of the set S, called $b(S)$, to be point $(-(B+1)^2 + ((m-1)(B+2))^2, (B+1)^2 - (m(B+2))^2)$ and the highest point of S, called $t(S)$, to be point $(0, (m(B+2))^2)$.

It is easy to verify that all the above requirements hold and that the area of the rectangle bounding the constructed point set is polynomial on B and m. □

Statement 3. $|S| = |V(G)| = m(B+1)+2$. □

We now proceed to show how from a solution for the 3-Partition problem we can derive a solution for the upward point set embeddability problem. Assume that there exists a solution for the instance of the 3-Partition problem and let it be $A_i = \{a_i^1, a_i^2, a_i^3\}$, $i = 1 \ldots m$. Note that $\sum_{j=1}^{3} a_i^j = B$. We first map s and t to $b(S)$ and $t(S)$, respectively. Then, we map vertex u_i on $t(C_i)$, $i = 1 \ldots m$. Note that the path from s to t through u_i is upward and $C_1, \ldots, C_i$ lie entirely to the left of this path, while $C_{i+1}, \ldots, C_m$ lie to the right of this path. Now each C_i has B free points. We map the vertices of paths P_i^1, P_i^2 and P_i^3 corresponding to a_i^1, a_i^2, a_i^3 to the remaining points of C_i in an upward fashion (see Figure 2.c). It is easy to verify that the whole drawing is upward and planar.

Assume now that there is an UPSE of G into S. We prove that there is a solution for the corresponding 3-Partition problem. The proof is based on the following statements.

Statement 4. *In any UPSE of G into S, s is mapped to $b(S)$.* □

Statement 5. *In any UPSE of G into S, only one vertex from set $\{u_1, \ldots, u_m\}$ is mapped to point set C_i, $i = 1 \ldots m$.*

Proof of Statement: For the sake of contradiction, assume that there are two distinct vertices u_j and u_k that are mapped to two points of the same point set C_i (see Figures 3). W.l.o.g. assume that u_k is mapped to a point higher than the point u_j is mapped to. We consider three cases based on the placement of the sink vertex t.

Case 1: t is mapped to a point of C_i (Figure 3.a). It is easy to see that arc (s, u_k) crosses arc (u_j, t), a clear contradiction to the planarity of the embedding.

Case 2: t is mapped to $t(S)$ (Figure 3.b). Similar to the previous case since $C_i \cup \{b(S), t(S)\}$ is a one-sided convex point set.

Case 3: t is mapped to a point of C_p, $p > i$, denote it by p_t (Figure 3.c). By Statement 1 $C_i \cup \{b(S), p_t\}$ is a convex point set and points p_t, $b(S)$ are consecutive points of $C_i \cup \{b(S), p_t\}$. Hence, arc (s, u_k) crosses arc (u_j, t), a contradiction. □

By Statement 5, we have that each $C_i, i = 1 \ldots m$, contains exactly one vertex from set $\{u_1, \ldots u_m\}$. Without lost of generality, we assume that u_i is mapped to a point of C_i.

Statement 6. *In any UPSE of G into S, vertex t is mapped to either a point of C_m or to $t(S)$.*

Proof of Statement: Vertex t has to be mapped higher than any u_i, $i = 1 \ldots m$, and hence higher than u_m, which is mapped to a point of C_m. □

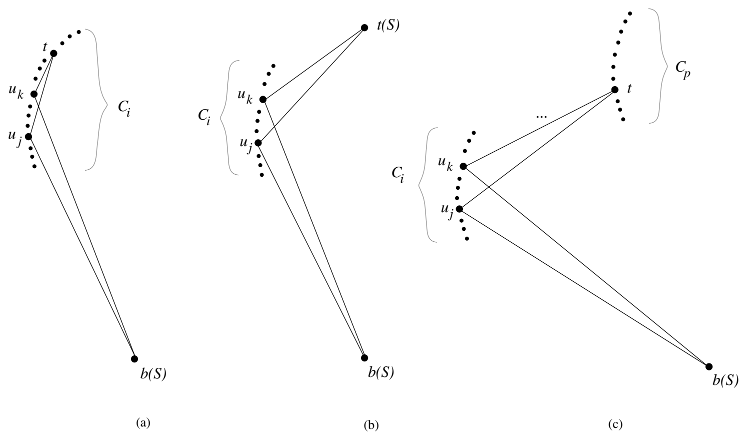


Fig. 3. Mappings used in the proof of Statement 5

Statement 7. *In any UPSE of G into S, vertex u_i is mapped to $t(C_i)$, $1 \leq i \leq m-1$, moreover, there is no arc (v, w) so that v is mapped to a point of C_i and w is mapped to a point of C_j, $j > i$.*

Proof of Statement: We prove this statement by induction on i, $i = 1 \dots m-1$. For the basis, assume that u_1 is mapped to a point p_1 different from $t(C_1)$ (see Figure 4.a). Let p_t be the point where vertex t is mapped. By Statement 6, p_t can be either $t(S)$ or a point of C_m. In both cases, point set $C_1 \cup \{b(S), p_t\}$ is a convex point set, due to the construction of the point set S and the Statement 1. Moreover, the points $b(S)$ and p_t are consecutive on the convex hull of point set $C_1 \cup \{b(S), p_t\}$.

Denote by p the point of C_1 that is exactly above the point p_1. From Statement 5, we know that no u_j, $j \neq 1$ is mapped to the point p. Due to Statement 6, t cannot be mapped to p. Hence there is a path P_k, $1 \leq k \leq 3m$, so that one of its vertices is mapped to p. Call this vertex u. We now consider two cases based on whether u is the first vertex of P_k of not.

Case 1: Assume that there is a vertex v of P_k, such that there is an arc (v, u). Since the drawing of S is upward, v is mapped to a point lower than p and lower than p_1. Since $C_1 \cup \{b(S), p_t\}$ is a convex point set, arc (v, u) crosses arc (u_1, t). A clear contradiction.

Case 2: Let u be the first vertex of P_k. Then, arc (s, u) crosses the arc (u_1, t) since, again, $C_1 \cup \{b(S), p_t\}$ is a convex point set, a contradiction.

So, we have that u_1 is mapped to $t(C_1)$, see Figure 4.b. Observe now that any arc (v, w), such that v is mapped to a point of C_1 and w is mapped to a point $x \in C_2 \cup \ldots \cup C_m \cup \{t(S)\}$ crosses arc (s, u_1), since $C_1 \cup \{b(S), x\}$ is a convex point set. So, the statement is true for $i = 1$.

For the induction step, we assume that the statement is true for C_g and u_g, $g \leq i-1$, i.e. vertex u_g is mapped to $t(C_g)$ and there is no arc connecting a point of C_g to a point of C_k, $k > g$ and this holds for any $g \leq i-1$. We now show that

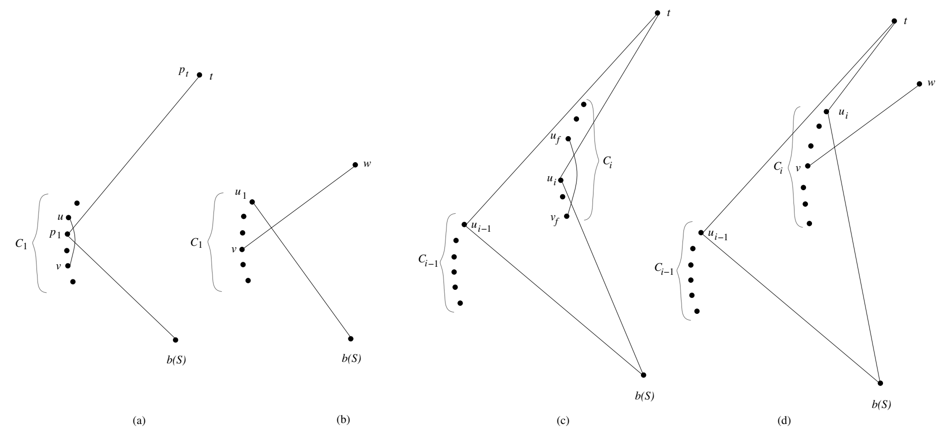


Fig. 4. Mappings used in the proof of Statement 7

it also holds for C_i and u_i. Again, for the sake of contradiction, assume that u_i is mapped to a point p_i different from $t(C_i)$ (see Figure 4.c).

Denote by q the point of C_1 that is exactly above point p_i. From Statement 5, we know that no u_l, $l \neq i$, is mapped to the point q. Due to Statement 6, t can not be mapped to q. Hence, there is a path P_f, so that one of its vertices is mapped to q. Call this vertex u_f. We now consider two cases based on whether u_f is the first vertex of P_f of not.

Case 1: Assume that there is a vertex v_f of P_k such that there is an arc (v_f, u_f). By the induction hypothesis, we know that v_f is not mapped to any C_l, $l < i$. Then, since the drawing of S is upward, v_f is mapped to a point lower than q and lower than p_i. Since $C_i \cup \{b(S), p_t\}$ is a convex point set, arc (v_f, u_f) crosses arc (u_i, t). A clear contradiction.

Case 2: Let u_f be the first vertex of P_k. Then, arc (s, u_f) crosses the arc (u_i, t) since, again, $C_i \cup \{b(S), p_t\}$ is a convex point set, a contradiction.

So, we have shown that u_i is mapped to $t(C_i)$, see Figure 4.d. Observe now that, any arc (v, w), such that v is mapped to a point of C_i and w is mapped to a point $x \in C_{i+1} \cup \ldots \cup C_m \cup \{t(S)\}$ crosses arc (s, u_i), since $C_i \cup \{b(S), x\}$ is a convex point set. So, the statement holds for i. □

A trivial corollary of the previous statement is the following:

Statement 8. *In any UPSE of G into S, any directed path P_j of G originating at s, $j \in \{1, \ldots, 3m\}$, has to be drawn entirely in C_i, for $i \in \{1, \ldots, m\}$.* □

The following statement completes the proof of the theorem.

Statement 9. *In any UPSE of G into S, vertex t is mapped to point $t(S)$.*

Proof of Statement: For the sake of contradiction, assume that t is not mapped to $t(S)$. By Statement 6 we know that t has to be mapped to a point in C_m. Assume first that t is mapped to point $t(C_m)$ (see Figure 5.a). Recall that u_{m-2} and u_{m-1} are mapped to $t(C_{m-2})$ and $t(C_{m-1})$, respectively, and that $\{t(C_i) : i = 1 \ldots m\}$

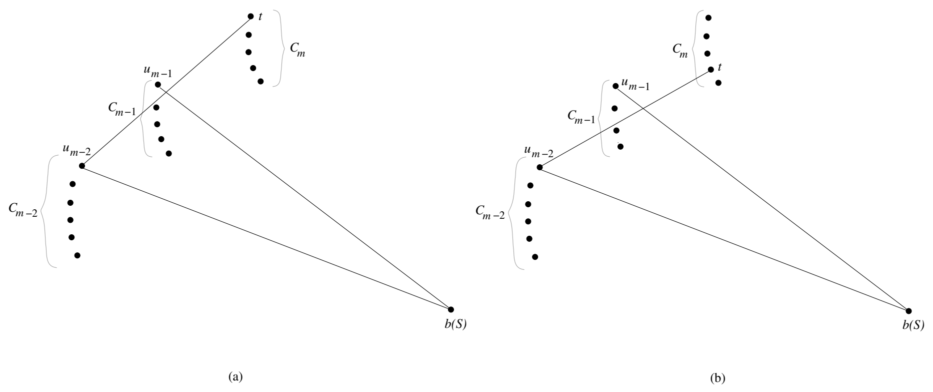


Fig. 5. (a-b) Mappings used in the proof of Statement 9

is a left-heavy convex point set. Hence, points $\{t(C_{m-2}), t(C_{m-1}), t(C_m), b(S)\}$ form a convex point set. It follows that segments $(t(C_{m-2}), t(C_m))$ and $(t(C_{m-1}), b(S))$ cross each other, i.e. edges (s, u_{m-1}) and (u_{m-2}, t) cross, contradicting the planarity of the drawing.

Consider now the case where t is mapped to a point of C_m, say p, different from $t(C_m)$ (see Figure 5.b). Since point p does not lie in triangle $t(C_{m-2})$, $t(C_{m-1})$, $b(S)$ and point $t(C_{m-1})$ does not lie in triangle $t(C_{m-2}), p, b(S)$, points $\{t(C_{m-2}), t(C_{m-1}), p, b(S)\}$ form a convex point set. Hence, segments $(t(C_{m-2}), p)$ and $(t(C_{m-1}), b(S))$ cross each other, i.e. edges (s, u_{m-1}) and (u_{m-2}, t) cross; a clear contradiction. $\square$

Let us now combine the above statements in order to derive a solution for the 3-Partition problem when we are given an UPSE of G into S. By Statement 4 and Statement 9, vertices s and t are mapped to $b(S)$ and $t(S)$, respectively. By Statement 5, for each $i = 1 \ldots m$, point set C_i contains exactly one vertex from $\{u_1, \ldots, u_m\}$, say u_i and, hence, the remaining points of C_i are occupied by the vertices of some paths $P_i^1, P_i^2, \ldots, P_i^c$. By Statement 8, $P_i^1, P_i^2, \ldots, P_i^c$ are mapped entirely to the points of C_i. Since C_i has $B + 1$ points, the highest of which is occupied by u_i, we infer that $P_i^1, P_i^2, \ldots, P_i^c$ contain exactly B vertices. We set $A_i = \{a_i^1, a_i^2, \ldots, a_i^c\}$, where a_i^j is the size of path P_i^j, $1 \leq j \leq c$. Since $\frac{B}{4} < a_i^j < \frac{B}{2}$ we infer that $c = 3$. The subsets A_i are disjoint and their union produces A.

Finally, we note that G has a single source s and the longest simple cycle of G has length 4, moreover the point set S is an m-convex point set for some $m > 1$. This completes the proof. $\square$

6 Open Problems

In this paper, we continued the study of the upward point-set embeddability problem, initiated in [1,2,8]. We showed that the problem is NP-complete, even if some restrictions are posed on the digraph and the point set. We also extended the positive and the negative results presented in [1,2] by resolving the problem for the class of k-switch trees, $k \in \mathbb{N}$. The partial results on the directed trees presented in [1,2] and in the present work, may be extended in two ways: (i) by presenting the time complexity of the problem of testing whether a given directed tree admits an upward planar straight-line embedding (UPSE) to a given general/convex point set and (ii) by presenting another classes of trees, that admit/do not admit an UPSE to a given general/convex point set. It would be also interesting to know whether there exists a class of upward planar digraphs $\mathcal{D}$ for which the decision problem whether a digraph $D \in \mathcal{D}$ admits an UPSE into a given point set P remains NP-complete even for a convex point set P.

References

1. Angelini, P., Frati, F., Geyer, M., Kaufmann, M., Mchedlidze, T., Symvonis, A.: Upward geometric graph embeddings into point sets. In: 18th International Symposium on Graph Drawing (GD 2010), LNCS (2010) (to appear)
2. Binucci, C., Di Giacomo, E., Didimo, W., Estrella-Balderrama, A., Frati, F., Kobourov, S., Liotta, G.: Upward straight-line embeddings of directed graphs into point sets. Computat. Geom. Th. Appl. 43, 219–232 (2010)
3. Bose, P.: On embedding an outer-planar graph in a point set. Computat. Geom. Th. Appl. 23(3), 303–312 (2002)
4. Bose, P., McAllister, M., Snoeyink, J.: Optimal algorithms to embed trees in a point set. J. Graph Alg. Appl. 1(2), 1–15 (1997)
5. Cabello, S.: Planar embeddability of the vertices of a graph using a fixed point set is NP-hard. J. Graph Alg. Appl. 10(2), 353–366 (2006)
6. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
7. Geyer, M., Kaufmann, M., Mchedlidze, T., Symvonis, A.: Upward point-set embeddability. Technical report. arXiv:1010:5937, `http://arxiv.org/abs/1010:5937`
8. Giordano, F., Liotta, G., Mchedlidze, T., Symvonis, A.: Computing upward topological book embeddings of upward planar digraphs. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 172–183. Springer, Heidelberg (2007)
9. Gritzmann, P., Mohar, B., Pach, J., Pollack, R.: Embedding a planar triangulation with vertices at specified positions. Amer. Math. Mont. 98, 165–166 (1991)
10. Heath, L.S., Pemmaraju, S.V., Trenk, A.N.: Stack and queue layouts of directed acyclic graphs: Part I. SIAM J. Comput. 28(4), 1510–1539 (1999)

Cooperative Query Answering by Abstract Interpretation

Raju Halder and Agostino Cortesi

Dipartimento di Informatica
Università Ca' Foscari Venezia, Italy
{halder,cortesi}@unive.it

Abstract. A common problem for many database users is how to formulate and submit correct queries in order to get useful responses from the system, with little or no knowledge of the database structure and its content. The notion of cooperative query answering has been explored as an effective mechanism to address this problem. In this paper, we propose a cooperative query answering scheme based on the Abstract Interpretation framework. In this context, we address three key issues: soundness, relevancy and optimality of the cooperative answers.

Keywords: Databases, Cooperative Query Answering, Abstract Interpretation.

1 Introduction

Traditional query processing system enforces database-users to issue precisely specified queries, while the system provides limited and exact answers, or no information at all when the exact answer is unavailable in the database. Therefore, it is important to the database-users to fully understand problem domain, query syntax, database schema, and underlying database content.

To remedy such shortcomings and to enhance the effectiveness of the information retrieval, the notion of cooperative query answering [5,6,8,15] has been explored. The cooperative query answering system provides users an intelligent database interface that allows them to issue approximate queries independent to the underlying database structure and its content, and provides additional useful information as well as the exact answers.

As an example, in response to the query about "specific flight departing at *10 a.m.* from *Rome Fiumicino* airport to *Paris Orly* airport" the cooperative query answering system may return "all flight information during *morning* time from airports in *Rome* to airports in *Paris*", and thus, the user will be able to choose other flight if the specific flight is unavailable. Such query answering is also known as neighborhood query answering, as instead of providing exact answers it captures neighboring information as well. Cooperative query answering system also gives users the opportunity to issue conceptual or approximate queries where they might ask more general questions, for example, "how to travel from *Rome* to *Paris* at a *reasonable* cost during *morning* time" or "find the flights that fly during *night* only" without knowing the exact database schema and its content. One of the benefits of issuing conceptual queries is to avoid reissuing of the set of concrete queries if the corresponding conceptual query returns empty result.

Cooperative query answering depends on the context in which queries are issued. The context includes the identity of the issuer, the intent or purpose of the query, the requirements that make explicit the answers relevant to the user etc. The following example illustrates it clearly: suppose a user issues a query asking the list of airports that are similar to "*Venice Marcopolo*" airport. Different contexts define the meaning of "*similarity between airports*" differently. For instance, to any surveyor "*similarity*" may refer in terms of the size and facilities provided in the airport, whereas to any flight company "*similarity*" may refer in terms of business point of view *i.e.* flight landing charges or other relevant taxes.

Abstract Interpretation is a well known semantics-based static analysis technique [7,9,11], originally developed by Cousot and Cousot as a unifying framework for designing and then validating static program analysis, and recently it becomes a general methodology for describing and formalizing approximate computation in many different areas of computer science, like model checking, process calculi, security, type inference, constraint solving, etc [9].

In our previous work [11], we introduced Abstract Interpretation framework to the field of database query languages as a way to provide sound approximation of the query languages. In this paper, we extend this to the field of cooperative query answering system: we propose a cooperative query answering scheme based on the Abstract Interpretation framework that consists of three phases - transformation of the whole query system by using abstract representation of the data values, cooperative query evaluation over the abstract domain, and concretization of the cooperative abstract result. The main contributions in this paper are: (*i*) we express the cooperative query evaluation by abstract databases, (*ii*) we express how to deal with cooperative query evaluation in presence of aggregate and negation operations, (*iii*) we address three key issues: soundness, relevancy and optimality of the cooperative answers.

The structure of the paper is as follows: Section 2 discusses related work in the literature and motivation of our work. Section 3 describes the key issues in the context of cooperative query answering. In Section 4, we discuss our proposed scheme. In Section 5, we show how our proposal is able to address the key issues. Finally, we draw our conclusions in Section 6.

2 Related Work and Motivation

Several techniques have been proposed in the literature based on logic model, semantic distance, fuzzy set theory, abstraction, and so on. The logic-based models [1,8] use first-order predicate logic to represent the database, the knowledge-base, and the user's queries. Content of the knowledge base helps in guiding query reformulation into more flexible and generalized query that provides relaxed, intelligent cooperative answer. However, these approaches have limitations in guiding the query relaxation process and the less intuitive query answering process due to lack of its expressiveness.

The semantic distance-based approaches [14,17] use the notion of semantic distance to represent the degree of similarity between data values, and provide ranked cooperative results sorted according to their semantic distances. For categorical data, distances between data values are stored in a table. However, since every pair is supposed to have semantic distances, in realistic application domain these approaches are inefficient as the table size gets extremely larger and becomes harder to maintain the consistency of the distance measures.

In [10], the initial queries are transformed into flexible form based on knowledge base and fuzzy set theory. Finally, these queries are rewritten into boolean queries and evaluated to the traditional database.

In abstraction-based models [4,6], the data are organized into multilevel abstraction hierarchies where nodes at higher level are the abstract representation of the nodes at lower level. The cooperative query answering is accomplished by generating a set of refined concrete queries by performing query abstraction and query refinement process by moving upward or downward through the hierarchy. Finally, the refined queries are issued to the database that provide additional useful information. These approaches suffer from high overhead when the degree of relaxation for a query is large, as the query abstraction-refinement process produces a large set of concrete queries to be issued to the database. To remedy this, fuzzy set theory or semantic distance approach is combined with abstraction hierarchy [5,13,15] to control the abstraction-refinement process and to provide a measure of nearness between exact and approximate answers.

However, all the above mentioned schemes do not provide any formal framework to cooperative query answering system. In addition, none of these schemes enlightens the key issues: soundness, relevancy and optimality in the context of cooperativeness of the query answers. Most of the existing techniques [4,5,6,13,15] suffer from the problem of soundness when query contains negation operation MINUS. In case of conceptual or approximate queries where approximate results are desirable, none of the schemes focuses on the way to compute aggregate functions when appearing in a query so as to preserve the soundness.

3 Key Issues: Soundness, Relevancy, Optimality

Any cooperative query answering scheme should respect three key issues: soundness, relevancy, and optimality. Intuitively, a cooperative query answer is *sound* if it is equal to or it is a superset of the corresponding extensional query answer. The *relevancy* of the answers *w.r.t.* the context concerns with avoiding the tuples that have no value to the user: all the information in the cooperative answer should have relevancy to the user. The third criteria *optimality* implies that the system should return as much information as possible, while satisfying the first two properties. Since there exist many cooperative answers corresponding to a given query under given context, the optimality describes the “preferability” of the query answers to the user.

Given a cooperative query processing system $\mathfrak{B}$, a database dB, a context C, and a query Q, the result obtained by the cooperative system is $R = \mathfrak{B}(dB, C, Q)$.

Definition 1 (Soundness). *Given a database dB, a query Q, and a context C. Let R be the extensional query answer obtained by processing Q on dB. The cooperative answer $R' = \mathfrak{B}(dB, C, Q)$ is sound if $R' \supseteq R$.*

The relevancy of the information to a user depends on his interests. These interests can be expressed in terms of constraints represented by well-formed formulas in first order logic. If the information provided by a cooperative system satisfies the set of constraints representing user's interests, it is treated as relevant. For instance, "flight duration in the result set must be less than 3 hours" can be used as a constraint that determines the relevancy of the information in the cooperative answer.

Definition 2 (Relevancy). *Given a database dB, a query Q, and a context C. Let $S(Q)$ be the set of constraints represented by well-formed formulas in first order logic that make explicit the answers relevant to the user. The cooperative answer $R = \mathfrak{B}(dB, C, Q)$ respects the relevancy if $\forall x \in R : \; x \models S(Q)$.*

It is worthwhile to mention that the system relaxes users' queries to obtain neighboring and conceptually related answers in addition to the exact answer. However, the formulas appearing in the pre-condition ϕ [11] of the query Q is different from the set of constraints appearing in $S(Q)$ that determines the relevancy of the cooperative answers. The constraints in $S(Q)$, in contrast to ϕ, is strict in the sense that there is no question of relaxing them, and violation of any of these constraints by the information in the cooperative answer will be treated as irrelevant.

A cooperative system may return different cooperative answers to a user in a given context. However, it is sensible to define a measure that describes the "preferability" of each answer. A cooperative answer is called more optimal than another answer if it is more preferable to the user in the given context than the other.

Definition 3 (Optimality). *Given a database dB, a query Q, a context C, and a set of constraints $S(Q)$ expressing user's requirements. The cooperative answer $R = \mathfrak{B}(dB, C, Q)$ is more optimal than $R' = \mathfrak{B}'(dB, C, Q)$ if*

$$\{x \in R : \; x \models S(Q)\} \supseteq \{x \in R' : \; x \models S(Q)\} \; \textit{and} \; \{y \in R : \; y \not\models S(Q)\} \subseteq \{y \in R' : \; y \not\models S(Q)\}$$

In other words, a cooperative answer is called more optimal than another answer when it contains more relevant information and less irrelevant information *w.r.t.* $S(Q)$ than the other.

4 Proposed Scheme

Our proposal consists of three phases: (i) Transforming the databases and its query languages by using abstract representation of the data values, (ii) Cooperative query evaluation over the abstract domain, and finally, (iii) Concretization of the cooperative abstract result.

4.1 Transforming from Concrete to Abstract Domain

In this section, we discuss how to lift databases and its query languages from concrete to the abstract domain of interests. The level of approximation of the database information obtained by abstraction gives a measure of the "preferability" of the cooperative answers and depends on the context in which queries are issued. The best correct approximation [9] of the database information according to the context provides the optimal cooperative answers to the end-users.

Generally, traditional databases are *concrete databases* as they contain data from the concrete domain, whereas *abstract databases* are obtained by replacing concrete values by the elements from abstract domains representing specific properties of interests. We may distinguish partially abstract database in contrast to fully abstract one, as in the former case only a subset of the data in the database is abstracted. The values of the data cells belonging to an attribute x are abstracted by following the Galois Connection $(\wp(D_x^{con}), \alpha_x, \gamma_x, D_x^{abs})$ [9], where $\wp(D_x^{con})$ and D_x^{abs} represent the powerset of concrete domain of x and the abstract domain of x respectively, whereas α_x and γ_x represent the corresponding abstraction and concretization functions (denoted $\alpha_x : \wp(D_x^{con}) \rightarrow D_x^{abs}$ and $\gamma_x : D_x^{abs} \rightarrow \wp(D_x^{con})$) respectively. In particular, partially abstract databases are special case of fully abstract databases where for some attributes x the abstraction and concretization functions are identity functions id, and thus, follow the Galois Connection $(\wp(D_x^{con}), id, id, D_x^{abs})$.

Table 1. Concrete and corresponding Abstract Databases

(a) Database containing concrete table "*flight*"

flight no.	source	destination	cost ($)	start-time	reach-time	availability
F001	Fiumicino (FCO)	Orly (ORY)	210.57	8.05	10.45	N
F002	Marcopolo (VCE)	Orly (ORY)	410.30	18.30	21.00	Y
F003	Ciampino (CIA)	Roissy Charles de Gaulle (CDG)	300.00	6.30	8.30	Y
F004	Urbe (LIRU)	Lyon-Saint Exupéry (LYS)	128.28	22.05	23.40	N
F005	Treviso (TSF)	Granby-Grand County (GNB)	200.15	16.00	17.20	Y
F006	Viterbo (LIRV)	Beauvais (BVA)	310.30	7.20	9.30	Y

(b) Abstract database containing abstract table "$flight^\sharp$"

flight no.$^\sharp$	source$^\sharp$	destination$^\sharp$	cost$^\sharp$	start-time$^\sharp$	reach-time$^\sharp$	availability$^\sharp$
$\top$	Rome	Paris	[200.00-249.99]	morning	morning	$\top$
$\top$	Venice	Paris	[400.00-449.99]	evening	night	$\top$
$\top$	Rome	Paris	[300.00-349.99]	morning	morning	$\top$
$\top$	Rome	Lyon	[100.00-149.99]	night	night	$\top$
$\top$	Venice	Lyon	[200.00-249.99]	afternoon	evening	$\top$

Let us illustrate it by an example. The database in Table 1(a) consists of concrete table "*flight*" that provides available flight information to the end-users of a travel agent application. The corresponding abstract table "$flight^\sharp$" is shown in Table 1(b) where source and destination airports are abstracted by the provinces they belong, the numerical values of the cost attribute are abstracted by the elements from the domain of intervals, the values of the start-time/reach-time attributes are abstracted by the periods from the abstract domain $PERIOD = \{\bot, morning, afternoon, evening, night, \top\}$ where $\top$ represents "anytime" and $\bot$ represents "don't know", the flight no. and availability attributes are abstracted by the topmost element $\top$ of their corresponding abstract lattices. Observe that the number of abstract tuples in an abstract database may be less than that in the corresponding concrete database.

Definition 4 (Abstract Database). *Let dB be a database. The database $dB^\sharp = \alpha(dB)$ where α is the abstraction function, is said to be an abstract version of dB if there exist a representation function γ, called concretization function such that for all tuple $\langle x_1, x_2, \ldots, x_n\rangle \in dB$ there exist a tuple $\langle y_1, y_2, \ldots, y_n\rangle \in dB^\sharp$ such that $\forall i \in [1 \ldots n]\ (x_i \in id(y_i) \vee x_i \in \gamma(y_i))$.*

4.2 Cooperative Query Evaluation

In [11], we proposed denotational semantics of database query languages in both concrete and abstract level. We extend this to the context of cooperative query answering, where the SQL queries are lifted into an abstract setting and instead of working on concrete databases they are applied on the corresponding abstract databases. However, in this paper, we restrict our discussions to the SELECT statements only. Let us start with a simple example:

Example 1. Consider an online booking application interacting with the concrete database depicted in Table 1(a). Suppose a user wants to travel from *Rome Fiumicino* airport to *Paris Orly* airport by a flight such that flight cost is less than or equal to *300 USD*. So the following query satisfying the required criteria can be issued:

Q_1 =`SELECT * FROM flight WHERE source = "Fiumicino" AND destination = "Orly" AND`
`cost` $\leq$ `300.00 AND availability=Y;`

Observe that the result ξ_1 of the query Q_1 is empty *i.e.* $\xi_1 = \emptyset$, because seats are not available in the flight from *Rome Fiumicino* to *Paris Orly* airport.

To obtain cooperative answers, we lift the whole query system from concrete to the abstract domain of interests by abstracting the database information and the associated query languages. The abstract database corresponding to the concrete database (Table 1(a)) is depicted in Table 1(b), and the abstract query corresponding to the concrete query Q_1 is shown below:

$Q_1^\sharp$ =`SELECT`$^\sharp$ `* FROM flight`$^\sharp$ `WHERE source`$^\sharp$ $=^\sharp$ `"Rome" AND destination`$^\sharp$ $=^\sharp$ `"Paris" AND`
`cost`$^\sharp$ $\leq^\sharp$ `[300.00, 349.99] AND availability`$^\sharp$ $=^\sharp$ $\top$;

where the abstract operation $\leq^\sharp$ for intervals is defined as follows:

$$[l_i, h_i] \leq^\sharp [l_j, h_j] \triangleq \begin{cases} true & \text{if } h_i \leq l_j \\ false & \text{if } l_i > h_j \\ \top & otherwise \end{cases}$$

and the abstract equality $=^\sharp$ is defined as usual.

When the imprecise abstract query $Q_1^\sharp$ is executed over the abstract database (Table 1(b)), it returns the flight information depicted in Table 2. This way, the abstraction of the databases and its query languages helps in obtaining additional information in the result.

Table 2. $\xi_1^\sharp$: Abstract Result of $Q_1^\sharp$

flight no.$^\sharp$	source$^\sharp$	destination$^\sharp$	cost$^\sharp$	start-time$^\sharp$	reach-time$^\sharp$	availability$^\sharp$
$\top$	Rome	Paris	[200.00-249.99]	morning	morning	$\top$
$\top$	Rome	Paris	[300.00-349.99]	morning	morning	$\top$

Consider a SELECT statement $Q = \langle A_{sel}, \phi \rangle$ where A_{sel} is *action part* and ϕ is *pre-condition part* of Q [11,12]. In the concrete domain, the pre-condition ϕ evaluates to two-valued logic (*true* and $false$) and the active data set on which A_{sel} operates contains those tuples for which ϕ evaluates to *true* only. In contrast, when we lift the whole query system to an abstract setting, the evaluation of the abstract pre-condition $\phi^\sharp$ over the abstract database results into a three-valued logic (*true*, $false$, and $\top$). The logic value $\top$ indicates that the tuple may or may not satisfy the semantic structure of $\phi^\sharp$. Thus, in the abstract domain, the active data set on which abstract SELECT action $A_{sel}^\sharp$ operates consists of those abstract tuples for which $\phi^\sharp$ evaluates to *true* or $\top$. For instance, in the query result $\xi_1^\sharp$ of Table 2, $\phi^\sharp$ evaluates to *true* for the first tuple with *cost*$^\sharp$ equal to $[200.00, 249.99]$, whereas it evaluates to $\top$ for the last tuple with *cost*$^\sharp$ equal to $[300.00, 349.99]$.

Soundness is preserved as the concretization of the abstract queries always results into a sound approximation of the corresponding concrete queries.

Cooperative query evaluation with aggregate functions: In case of conceptual or approximate queries where approximate results are desirable, none of the existing techniques focuses on the way to compute aggregate functions when appearing in a query so as to preserve the soundness. In this section, we discuss how to compute the cooperative aggregate functions in an abstract setting to provide the users a sound approximation of the aggregate results.

Let $T_\phi^\sharp$ be the set of abstract tuples for which $\phi^\sharp$ evaluates to *true* or $\top$. The application of abstract GROUP BY function $g^\sharp$ on $T_\phi^\sharp$ yields to a set of abstract groups $\{G_1^\sharp, G_2^\sharp, \ldots, G_n^\sharp\}$. When no $g^\sharp$ appears in abstract SELECT statement $Q^\sharp$, we assume $T_\phi^\sharp$ as a single abstract group.

Given an abstract group $G^\sharp$, we can partition the tuples in $G^\sharp$ into two parts: $G_{yes}^\sharp$ for which $\phi^\sharp$ evaluates to true, and $G_{may}^\sharp$ for which $\phi^\sharp$ evaluates to $\top$. Thus we can write $G^\sharp = G_{yes}^\sharp \cup G_{may}^\sharp$, where $G_{yes}^\sharp \cap G_{may}^\sharp = \emptyset$.

Let $s^\sharp$ be an abstract aggregate function and $e^\sharp$ be an abstract expression. To ensure the soundness, the computation of $s^\sharp(e^\sharp)$ on $G^\sharp$ is defined as follows: $s^\sharp(e^\sharp)[G^\sharp] = [min^\sharp(a^\sharp),\ max^\sharp(b^\sharp)]$, where $a^\sharp = fn^\sharp(e^\sharp)[G_{yes}^\sharp]$ and $b^\sharp = fn^\sharp(e^\sharp)[G^\sharp]$.

By $fn^\sharp(e^\sharp)[G_{yes}^\sharp]$ and $fn^\sharp(e^\sharp)[G^\sharp]$ we mean that the function $fn^\sharp$ is applied on the set of abstract values obtained by evaluating $e^\sharp$ over the tuples in $G_{yes}^\sharp$ and

$G^\sharp$ respectively, yielding to an abstract aggregate value as result. The computation of $fn^\sharp$ is defined differently by considering two different situations that can arise: (i) when the primary key is abstracted, yielding two or more tuples mapped into a single abstract tuple, and (ii) when the primary key is not abstracted and the identity of each tuples are preserved in abstract domain. Both the functions $min^\sharp$ and $max^\sharp$ take single abstract value $a^\sharp$ and $b^\sharp$ respectively as parameters which are obtained from $fn^\sharp$, and returns a concrete numerical value as output. $min^\sharp(a^\sharp)$ returns minimum concrete value from $\gamma(a^\sharp)$ and $max^\sharp(b^\sharp)$ returns maximum concrete value from $\gamma(b^\sharp)$, where γ is the concretization function.

Lemma 1. *Let s and e be an aggregate function and an expression respectively. Suppose the corresponding abstract representation of s and e are $s^\sharp$ and $e^\sharp$ respectively. Let $s^\sharp(e^\sharp)[G^\sharp]$ be an abstract aggregate value obtained by performing $s^\sharp$ parameterized with $e^\sharp$ over an abstract group $G^\sharp$. The abstract aggregate function $s^\sharp$ is sound if*

$$\forall G \in \gamma(G^\sharp), \quad s(e)[G] \in \gamma(s^\sharp(e^\sharp)[G^\sharp])$$

Example 2. Consider the database of Table 1(a) and the following query that computes the average ticket price for all flights departing after *7 o'clock* in the morning from any airport in Rome to any airport in Paris region:

Q_2 =`SELECT AVG(cost), COUNT(*) FROM flight WHERE source IN (FCO, CIA, LIRU, LIRV)`
`and destination IN (ORY, CDG, BVA) and start-time` $\geq$ `7.00`

If we execute Q_2 on the database of Table 1(a), we get the result ξ_2 depicted in Table 3.

Table 3. ξ_2: Result of Q_2 (concrete)

$AVG(cost)$	$COUNT(*)$
260.44	2

The abstract version of Q_2 is defined as below:

$Q_2^\sharp$ =`SELECT`$^\sharp$ `AVG`$^\sharp$`(cost`$^\sharp$`), COUNT`$^\sharp$`(*) FROM flight`$^\sharp$ `WHERE source`$^\sharp$ `IN (Rome)`
`and destination`$^\sharp$ `IN (Paris) and start-time`$^\sharp$ $\geq^\sharp$ `morning`

The result of $Q_2^\sharp$ on the abstract database of Table 1(b) is shown in Table 4. The

Table 4. $\xi_2^\sharp$: Result of $Q_2^\sharp$

$AVG^\sharp(cost^\sharp)$	$COUNT^\sharp(*)$
[0, 349.99]	[0, $+\infty$]

evaluation of the abstract WHERE clause extracts two tuple with $cost^\sharp$ equal to $[200.00, 249.99]$ and $[300.00, 349.99]$ both belonging to $G^\sharp_{may}$. So, $G^\sharp_{yes} = \{\}$. Thus, we get $a^\sharp = fn^\sharp(\{\}) = average^\sharp(\{\}) = [0, 0]$ and $b^\sharp = fn^\sharp(\{[200.00, 249.99], [300.00, 349.99]\}) = average^\sharp(\{[200.00, 249.99], [300.00, 349.99]\}) = [200.00, 349.99]$. Hence, $AVG^\sharp(cost^\sharp) = [min^\sharp(a^\sharp), max^\sharp(b^\sharp)] = [0, 349.99]$. In our example the abstraction of two or more concrete tuples may result into a single abstract tuple since the primary key is abstracted, so in such case $COUNT^\sharp(*) = [min^\sharp(a^\sharp), +\infty]$. observe that the result is sound *i.e.* $\xi_2 \in \gamma(\xi_2^\sharp)$.

Cooperative query evaluation with negation operation: Most of the existing abstraction-based techniques [4,5,6,13,15] suffer from the problem of soundness when query contains negation operation $MINUS$, because abstraction of the query appearing on right side of $MINUS$ may remove more information from the result of the query on left side of $MINUS$, yielding to a result that does not satisfy the soundness. In this section, we discuss the way to treat negation operation so as to preserve the soundness.

Given any abstract query $Q^\sharp$ and an abstract database state $dB^\sharp$, the result of the query can be denoted by $\xi^\sharp = [\![Q^\sharp]\!](dB^\sharp) = (\xi^\sharp_{yes}, \xi^\sharp_{may})$ where $\xi^\sharp_{yes}$ is the part of the query result for which semantic structure of $\phi^\sharp$ evaluates to true and $\xi^\sharp_{may}$ represents the remaining part for which $\phi^\sharp$ evaluates to $\top$[1]. Observe that we assume $\xi^\sharp_{yes} \cap \xi^\sharp_{may} = \emptyset$.

Consider an abstract query of the form $Q^\sharp = Q^\sharp_l \ MINUS^\sharp \ Q^\sharp_r$. Let the result of $Q^\sharp_l$ and $Q^\sharp_r$ be $\xi^\sharp_l = (\xi^\sharp_{yes_l}, \xi^\sharp_{may_l})$ and $\xi^\sharp_r = (\xi^\sharp_{yes_r}, \xi^\sharp_{may_r})$ respectively. The difference operation $MINUS^\sharp$ over an abstract domain is defined as follows:

$$\begin{aligned}\xi^\sharp &= \xi^\sharp_l \ MINUS^\sharp \ \xi^\sharp_r = \langle \xi^\sharp_{yes_l}, \xi^\sharp_{may_l} \rangle \ MINUS^\sharp \ \langle \xi^\sharp_{yes_r}, \xi^\sharp_{may_r} \rangle \\ &= \langle \xi^\sharp_{yes_l} - (\xi^\sharp_{yes_r} \cup \xi^\sharp_{may_r}), (\xi^\sharp_{may_l} \cup \xi^\sharp_{may_r}) - \xi^\sharp_{yes_r} \rangle \end{aligned} \tag{1}$$

Observe that the first component $(\xi^\sharp_{yes_l} - (\xi^\sharp_{yes_r} \cup \xi^\sharp_{may_r}))$ contains those tuples for which the pre-condition $\phi^\sharp$ strictly evaluates to *true*, whereas for the second component $((\xi^\sharp_{may_l} \cup \xi^\sharp_{may_r}) - \xi^\sharp_{yes_r})$ it evaluates to $\top$.

Example 3. Consider the database of Table 1(a) and the following query that finds all flights with ticket price strictly less than *205 USD*:

$Q_3 = Q_l$ `MINUS` Q_r, where Q_l = `SELECT * FROM flight`; and Q_r = `SELECT * FROM flight WHERE cost` ≥ 205.00;

The execution of Q_3 on the database of Table 1(a) yields the result ξ_3 shown in Table 5.

Table 5. ξ_3: Result of Q_3 (concrete)

flight no.	source	destination	cost ($)	start-time	reach-time	availability
$F004$	Urbe (LIRU)	Lyon-Saint Exupéry (LYS)	128.28	22.05	23.40	N
$F005$	Treviso (TSF)	Granby-Grand County (GNB)	200.15	16.00	17.20	Y

The corresponding abstract version of Q_3 is as follows:

$Q^\sharp_3 = Q^\sharp_l$ `MINUS`$^\sharp$ $Q^\sharp_r$, where $Q^\sharp_l$ = `SELECT`$^\sharp$ `* FROM flight`$^\sharp$; and $Q^\sharp_r$ = `SELECT`$^\sharp$ `* FROM flight`$^\sharp$ `WHERE cost`$^\sharp$ $\geq^\sharp [200.00, 249.99]$;

By following the abstract computation of $MINUS^\sharp$ defined in Equation 1, we get the result of $Q^\sharp_3$ depicted in Table 6 which is sound *i.e.* $\xi_3 \in \gamma(\xi^\sharp_3)$.

[1] When abstract query $Q^\sharp$ uses aggregate functions $s^\sharp$, application of $s^\sharp$ over a group $G^\sharp$ yields to a single row in $\xi^\sharp$. This row belongs to $\xi^\sharp_{may}$ only if all rows of that group belong to $G^\sharp_{may}$, otherwise it belongs to $\xi^\sharp_{yes}$.

Table 6. $\xi_3^\sharp$: Result by performing abstract computation of $Q_3^\sharp$

flight no.$^\sharp$	source$^\sharp$	destination$^\sharp$	cost$^\sharp$	start-time$^\sharp$	reach-time$^\sharp$	availability$^\sharp$
$\top$	Rome	Paris	[200.00-249.99]	morning	morning	$\top$
$\top$	Rome	Lyon	[100.00-149.99]	night	night	$\top$
$\top$	Venice	Lyon	[200.00-249.99]	afternoon	evening	$\top$

4.3 Concretization of the Cooperative Abstract Result

Given a concrete and the corresponding abstract databases dB and $dB^\sharp$ respectively, let $\xi^\sharp$ be the abstract answer obtained by executing the abstract query $Q^\sharp$ on $dB^\sharp$. The cooperative answer $R = \mathfrak{B}(dB, C, Q)$ returned to the user is obtained by: $R = \gamma(\xi^\sharp) \cap dB$. That is, the cooperative answer is obtained by mapping the abstract result into its concrete counterpart. For instance, after mapping $\xi_1^\sharp$ (Table 2), the user gets its concrete counterpart R_1 shown in Table 7.

Table 7. R_1: Concrete Result obtained by concretizing the abstract result $\xi_1^\sharp$

flight no.	source	destination	cost (\$)	start-time	reach-time	availability
F001	Fiumicino (FCO)	Orly (ORY)	210.57	8.05	10.45	N
F003	Ciampino (CIA)	Roissy Charles de Gaulle (CDG)	300.00	6.30	8.30	Y
F006	Viterbo (LIRV)	Beauvais (BVA)	310.30	7.20	9.30	Y

5 Soundness, Relevancy, and Optimality

Suppose, dB and $dB^\sharp$ represent a concrete database and its abstract version respectively. If Q and $Q^\sharp$ are representing the queries on concrete and abstract domain respectively, let ξ and $\xi^\sharp$ be the results of applying Q and $Q^\sharp$ on the dB and $dB^\sharp$ respectively.

Definition 5. *Let $dB^\sharp$ be an abstract table and $Q^\sharp$ be an abstract query. $Q^\sharp$ is sound iff $\forall dB \in \gamma(dB^\sharp)$. $\forall Q \in \gamma(Q^\sharp) : Q(dB) \in \gamma(Q^\sharp(dB^\sharp))$.*

Let us denote by the notation $\mathfrak{B} \blacktriangleleft D^{abs}$ the fact that the cooperative system $\mathfrak{B}$ uses the abstract domain D^{abs}, and by $D_1^{abs} \sqsupseteq D_2^{abs}$ the fact that the abstract domain D_1^{abs} is an abstraction of D_2^{abs}.

Definition 6. *Given two abstract domain D_1^{abs} and D_2^{abs}. The domain D_1^{abs} is an abstraction of D_2^{abs} (denoted $D_1^{abs} \sqsupseteq D_2^{abs}$) if $\forall X^\sharp \in D_2^{abs}$, $\forall x \in \gamma_2(X^\sharp)$, $\exists!\ l \in D_1^{abs}$: $\alpha_1(x) = l$, where α_1 is the abstraction function corresponding to D_1^{abs}, and γ_2 is the concretization functions corresponding to D_2^{abs}.*

The extensional query answering system uses zero level abstraction and it returns only the exact answer if available. More relaxation of the query indicates more abstraction used by the cooperative system, returning more cooperative information to the users. Thus whenever we tune the level of abstraction from lower to higher, the system returns monotonically increasing answer set, *i.e.*

If $(\mathfrak{B} \blacktriangleleft D_1^{abs})$ and $(\mathfrak{B}' \blacktriangleleft D_2^{abs})$ and $(D_1^{abs} \sqsupseteq D_2^{abs})$, then $\mathfrak{B}(dB, C, Q) \supseteq \mathfrak{B}'(dB, C, Q)$

When the term "relevancy" comes into the context, the tuning of abstraction must end at a particular point, after which the system returns irrelevant additional information that does not satisfy the constraints in $S(Q)$, where $S(Q)$ is the set of constraints that make explicit the answers relevant to the user. We call the abstraction used at that point as the best correct approximation of the database information. Best correct approximation, thus, depends on the context that defines $S(Q)$. More abstraction beyond the best correct approximation level makes the answer partially relevant as it includes additional irrelevant information *w.r.t.* $S(Q)$.

Example 4. The cooperative answer R_1 of the query Q_1 in Example 1 is shown in Table 7. Let the constraint set be $S(Q)$={flights must be destined in the airport ORY/BVA/CDG/LYS, flight duration must be less than 3 hours}. Observe that the cooperative answer in Table 7 is completely relevant as all tuples in the answer satisfy $S(Q)$. If we use an higher level of abstraction, for instance, if the source and destination airports in Table 1(a) are abstracted by the nations they belong (in our example, Italy and France), the corresponding cooperative answer R'_1 of the query Q_1 will contain all tuples of the concrete Table 1(a) except the tuple with flight no. F002. The answer R'_1 is partially relevant because one tuple among them (flight no. equal to F005) does not satisfy $S(Q)$.

Our system can work together with a filtering system that can filter out those tuples from the partially relevant results that do not satisfy $S(Q)$, and ranks the results based on the satisfiability of tuples *w.r.t.* $S(Q)$. However, the level of abstraction determines the efficiency of the system with respect to the processing time.

5.1 Partial Order between Cooperative Answers

Given a database dB, a query Q, and a context C, the cooperative system may return different cooperative answers to the user under context C depending on the level of abstraction of the abstract domain which is used. We define a partial order between any two cooperative answers: a cooperative answer $R = \mathfrak{B}(dB, C, Q)$ is said to be better than another answer $R' = \mathfrak{B}'(dB, C, Q)$ (denoted $R \leq R'$) if R is more optimal than R' (see definition 3). The partial-ordered set of all cooperative answers for a given query under given context forms a lattice. The bottom most element R_0 determines worst cooperative answer which is completely irrelevant *w.r.t.* $S(Q)$, whereas the top most element R_n is the best cooperative answer which is completely relevant *w.r.t.* $S(Q)$.

Example 5. The cooperative answer R_1 of the query Q_1 in Example 1 is shown in Table 7. When we abstract the airports in Table 1(a) by the nations they belong, the corresponding cooperative answer R'_1 of the query Q_1 will contain all tuples of the concrete Table 1(a) except the tuple with flight no. F002. Since R'_1 contains one irrelevant tuple (tuple with flight no. F005) *w.r.t.* $S(Q)$ as depicted in Example 4, after filtering out the irrelevant tuple we get the result R_2 shown in Table 8. Observe that R_2 is better than the result R_1 (*i.e.* $R_2 < R_1$) since R_2 is more optimal than R_1 according to definition 3.

Table 8. R_2: Cooperative result of Q_1 while using more abstraction

flight no.	source	destination	cost ($)	start-time	reach-time	availability
F001	Fiumicino (FCO)	Orly (ORY)	210.57	8.05	10.45	N
F003	Ciampino (CIA)	Roissy Charles de Gaulle (CDG)	300.00	6.30	8.30	Y
F004	Urbe (LIRU)	Lyon-Saint Exupéry (LYS)	128.28	22.05	23.40	N
F006	Viterbo (LIRV)	Beauvais (BVA)	310.30	7.20	9.30	Y

There exists a wide variety of abstract domains with different expressiveness and complexity that focus on the linear relationship among program variables, such as Interval Polyhedra [3,2] to infer interval linear relationship, or Difference-Bound Matrices [16] representing the constraints of the form $x - y \leq c$ and $\pm x \leq c$ where x, y are program variables and c is constant. We can exploit such abstract domains by focusing on the set of constraint $S(Q)$ that make the answers relevant to the users.

6 Conclusions

Our new cooperative query answering scheme needs to be further refined. In particular, we are currently investigating its application to more sophisticated scenarios on different abstract domains, in order to properly address the tradeoff between accuracy and efficiency.

Acknowledgement

Work partially supported by RAS project "TESLA - Tecniche di enforcement per la sicurezza dei linguaggi e delle applicazioni".

References

1. Braga, J.L., Laender, A.H.F., Ramos, C.V.: Cooperative relational database querying using multiple knowledge bases. In: Proceedings of the 12th International Florida Artificial Intelligence Research Society Conference, May 1999, pp. 95–99. AAAI Press, Orlando (May 1999)
2. Chen, L., Miné, A., Cousot, P.: A sound floating-point polyhedra abstract domain. In: Ramalingam, G. (ed.) APLAS 2008. LNCS, vol. 5356, pp. 3–18. Springer, Heidelberg (2008)
3. Chen, L., Miné, A., Wang, J., Cousot, P.: Interval polyhedra: An abstract domain to infer interval linear relationships. In: Palsberg, J., Su, Z. (eds.) SAS 2009. LNCS, vol. 5673, pp. 309–325. Springer, Heidelberg (2009)
4. Chu, W.W., Chen, Q.: A structured approach for cooperative query answering. IEEE Transactions on Knowledge and Data Engineering 6(5), 738–749 (1994)
5. Chu, W.W., Chen, Q.: Neighborhood and associative query answering. Journal of Intelligent Information Systems 1(3-4), 355–382 (1992)

6. Chu, W.W., Yang, H., Chiang, K., Minock, M., Chow, G., Larson, C.: Cobase: a scalable and extensible cooperative information system. Journal of Intelligent Information Systems 6(2-3), 223–259 (1996)
7. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 238–252. ACM Press, New York (1977)
8. Gaasterland, T.: Cooperative answering through controlled query relaxation. IEEE Expert: Intelligent Systems and Their Applications 12(5), 48–59 (1997)
9. Giacobazzi, R., Ranzato, F., Scozzari, F.: Making abstract interpretations complete. Journal of the ACM (JACM) 47(2), 361–416 (2000)
10. Hachani, N., Ounelli, H.: A knowledge-based approach for database flexible querying. In: DEXA 2006, pp. 420–424. IEEE CS, Krakow (2006)
11. Halder, R., Cortesi, A.: Abstract interpretation for sound approximation of database query languages. In: Proceedings of the IEEE 7th International Conference on INFOrmatics and Systems (INFOS 2010), Advances in Data Engineering and Management Track, March 28–30, pp. 53–59. IEEE Catalog Number: IEEE CFP1006J-CDR, Cairo (2010)
12. Halder, R., Cortesi, A.: Observation-based fine grained access control for relational databases. In: Proceedings of the 5th International Conference on Software and Data Technologies (ICSOFT 2010), July 22–24, pp. 254–265. INSTICC Press, Athens (2010)
13. Huh, S.-Y., Moon, K.-H.: Approximate query answering approach based on data abstraction and fuzzy relation. In: Proceedings of INFORMS-KORMS, Seoul, Korea, pp. 2057–2065 (June 2000)
14. Ichikawa, T., Hirakawa, M.: Ares: a relational database with the capability of performing flexible interpretation of queries. IEEE Transactions on Software Engineering 12(5), 624–634 (1986)
15. Keun Shin, M., Huh, S.Y., Lee, W.: Providing ranked cooperative query answers using the metricized knowledge abstraction hierarchy. Expert Systems with Applications 32(2), 469–484 (2007)
16. Miné, A.: A new numerical abstract domain based on difference-bound matrices. In: Danvy, O., Filinski, A. (eds.) PADO 2001. LNCS, vol. 2053, pp. 155–172. Springer, Heidelberg (2001)
17. Palpanas, T., Koudas, N.: Entropy based approximate querying and exploration of datacubes. In: Proceedings of the 13th International Conference on Scientific and Statistical Database Management (SSDBM 2001), pp. 81–90. IEEE Computer Society Press, Fairfax (2001)

An Improved B+ Tree for Flash File Systems

Ferenc Havasi

Department of Software Engineering, University of Szeged
Árpád tér 2. 6722, Szeged, Hungary
havasi@inf.u-szeged.hu

Abstract. Nowadays mobile devices such as mobile phones, mp3 players and PDAs are becoming evermore common. Most of them use flash chips as storage. To store data efficiently on flash, it is necessary to adapt ordinary file systems because they are designed for use on hard disks. Most of the file systems use some kind of search tree to store index information, which is very important from a performance aspect. Here we improved the B+ search tree algorithm so as to make flash devices more efficient. Our implementation of this solution saves 98%-99% of the flash operations, and is now the part of the Linux kernel.

1 Introduction

These days mobile devices such as mobile phones, mp3 players, PDAs, GPS receivers are becoming more and more common and indispensable, and this trend is expected to continue in the future. New results in this area can be very useful for the information society and the economy as well.

Most of the above-mentioned devices handle data files and store them on their own storage device. In most cases this storage device is flash memory [3]. On smart devices an operating system helps to run programs that use some kind of file system to store data. The response time (how much time is needed to load a program) and the boot time (how much time is needed to load all the necessary code and data after power up) of the device both depend on the properties of the file system. Both of these parameters are important from the viewpoint of device usability.

Here we introduce a new data structure and algorithm designed for flash file systems that work more efficiently than the previous technical solutions. We begin by introducing the workings of flash chips, then describe the previously most prevalent Linux flash file system, namely JFFS2. Its biggest weakness is its indexing method: it stores the index in the RAM memory, not in the flash memory. This causes unnecessary memory consumption and performance penalties. Next, we introduce the new method in a step-by-step fashion, with an efficient combination of storing the index in memory and flash after taking factors like data security and performance into account.

This new solution has been implemented in the UBIFS file system by our department in cooperation with Nokia, and it is now an official part of the Linux kernel.

2 How the Flash Memory Works

Flash memory is a non-volatile computer memory that can be electrically erased and reprogrammed. One of the biggest limitations of flash memory is that although it can be read or programmed one byte or word at a time in a random-access fashion, it must be erased one "block" at a time. The typical size of a block is 8-256K.

The two common types of flashes are NOR and NAND flash memories, whose basic properties are summarized in the following table [8].

	NOR	NAND
Read/write size	Can read/write bytes individually	Can read/write only pages (page size can be 512 or 2048 bytes)
I/O speed	Slow write, fast read	Fast write, fast read
Erase	Very slow	Fast
XIP (Execute in Place)	Yes	No
Fault tolerance, detection	No	Yes
Price/size	Relatively expensive	Relatively cheap

One of the drawbacks of the flash system is that its erase block can be erased about 100,000 times, and afterwards the chip will be unstable. This is why most of the ordinary file systems (FAT, ext2/3, NTFS, etc.) are unusable on flash directly, because all of them have areas which are rarely rewritten (FAT, super block,), and this area would soon be corrupted.

One of the most common solutions to balance the burden of the erase blocks is FTL (Flash Translation Layer) [6], which hides the physical erase blocks behind a layer. This layer uses a map to store data about what the corresponding physical erase block is for each logical number. Initially this map is identical, so for example logical block 5 is mapped to physical block 5. This layer also contains an erase counter for each block (how many times it was erased). If this counter reaches a high number (relative to the average), the system will exchange two erase blocks (using the map), selecting an erase block which has a relatively low strain. This method is used in most pen drives to keep the burden low. It works quite well in practice, but does not provide the optimal solution to performance problems. For instance, to overwrite just a few bytes (such as a pointer in a search tree), an entire erase block ($\sim$128K) has to been erased and reprogrammed.

Accordingly, especially in the case of root file systems, it is worthwhile using flash file systems which are designed specifically for flash devices. Now we will discuss Linux flash files systems (JFFS and JFFS2), which are freely available to everyone.

3 JFFS, JFFS2: Flash File System without Flash Index

The basic idea behind JFFS [2] is quite simple: the file system is just a concentric journal. In essence, all of the modifications on the file system are stored as a journal entry (node). When mounting, the system scans this journal and then replays the events in the memory, creating an index to register which file is where. If the journal entries are such that the device is nearly full, the system performs the following steps:

- From the beginning of the used area copy all of the journal entries which are still valid, so the corresponding file is not deleted or overwritten.
- Erase the emptied area (erase block), so the there will be new space to store the new journal entries.

This very simple approach eases the burden on the erase blocks, but it also has its drawbacks:

1. If a dirty (deleted or overwritten) data area is in the middle of the current journal area, to free it, it is necessary to copy half of the entire journal.
2. When mounting, it is necessary to scan the entire medium.

Problem 1 is solved by the new version of this file system called JFFS2. It utilizes lists to register the dirty ratio of erase blocks, and if free space is required, it frees the dirty erase blocks (after moving the valid nodes to another place). There is a slight chance that it will free clean erase blocks as well, just to ease the burden, but this will rarely occur.

Problem 2 is only partially solved by JFFS2. It collects information via erase blocks needed when mounting, and it stores them at the end of the erase blocks, so only this needs to be scanned. Afterwards it attempts to minimize the size of the index in the RAM. However, the memory consumption and the mounting time are still linearly proportional to the size of the flash, and in practice over 512M may be unusable, especially in the case of large files e.g. video films.

Because the root of this problem lies in the base data structures and operating method of the JFFS2, we should construct a new file system to eliminate the linear dependency. To achieve this, it is necessary to store index information on the flash so as to avoid always having to rebuild it when mounting.

4 B+ Tree

Most file systems employ search-trees to index the stored data, and the B+ tree [4] is a special search-tree with the following features:

- It stores records: $r = (k, d)$; k = key, d = data. The key is unique.
- Data is stored only in leaves, inner-nodes are only index-nodes.
- In an index-node there are x keys, and also $x + 1$ pointers, each pointing to the corresponding subtree.
- The B+ tree has one main parameter, namely its order. If the order of a B+ is d, then for each index node there is a minimum of d keys, and a maximum of $2d$ keys, so there are minimum of $d + 1$ pointers, and maximum of $2d + 1$ pointers in the node.
- From the above, if a B+ tree stores n nodes, its height must not be greater than $log_d(n) + 1$. The total cost of insertion and deletion is $O(log_d(n))$.

This data structure is used by some database systems like PostgreSQL and MySQL, and file systems like ReiserFS, XFS, JF2 and NTFS. These file systems are based on the behaviour of real hard disks; that is, each block can be overwritten an unlimited number of times. Thus if there is a node insertion, only an update of the corresponding index node is needed at that location. Unfortunately it does not work well with flash storage devices, so it was found necessary to improve the flash-optimized version of the B+ tree.

5 Wandering Tree

A modified version of the B+ tree can be found in the LogFS file system [7], which is a flash file system for Linux. It is still in the development stage, and probably will be never finished, because UBIFS offers a much better alternative. This B+ variant is called a wandering tree. The general workings of this tree can be seen in Figure 1.

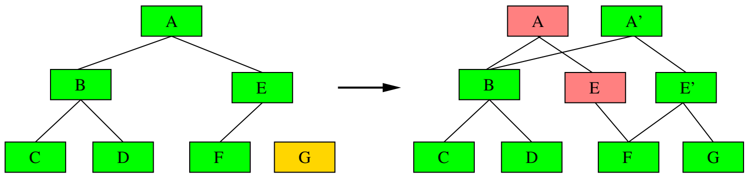


Fig. 1. Wandering tree before and after insertion

Like the ordinary B+ tree algorithm, during a node insertion it is normally necessary to modify a pointer at just one index node. In the case of flash memory the modification is costly, so this wandering algorithm writes out a new node instead of modifying the old one. If there is a new node, it is necessary to modify its parent as well, up to the root of the tree. It means that one node insertion (not counting the miscellaneous balancing) requires h new nodes, where h is the height of the tree. It also generates h dirty (obsolete) nodes, as well. Because h is $O(log_d(n))$, where n is the tree node number, the cost of this operand is still $O(log_d(n))$.

6 TNC: An Improved Wandering Tree

The above wandering tree algorithm still has performance issues, because its insert method is inefficient: it requires $log_d(n)$ new nodes, and it also generates a number of garbage nodes.

To solve these problems we decided to improve this algorithm, which now works in the UBIFS file system [1]. Due to its efficient caching technique it is able to collect node insertions and deletions in the memory, so fewer flash operations are required. At the same time, the method can ensure that if there

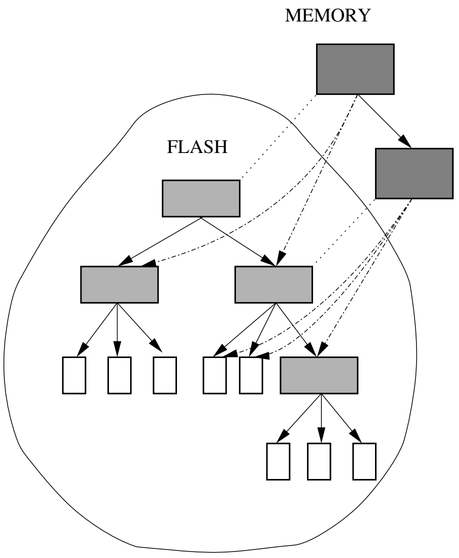


Fig. 2. The TNC data structure

is a sudden power loss (in the case of an embedded system this could happen at any time) there will be no data loss.

This new data structure and the algorithm are both called the TNC (Tree Node Cache). It is a B+ tree, which is partly in the flash memory, and partly in the memory (see Figure 2). Its operators are improved versions of those used in the wandering tree algorithm.

6.1 Data Structure of TNC

When TNC is not in use (e.g. the file system is not mounted) all the data is stored in the flash memory, in the ordinary B+ tree format.

When in use, some index nodes of the tree are loaded into the memory. The caching works in such a way that the following statement is always true : if an index node is in the RAM memory, its children may also be in the memory, or in the flash memory. But if the index node is not in the memory, all of its children are in the flash memory.

If an index node is in the memory, the following items are stored in it:

- **Flag clean:** it tells us whether it has been modified or not.
- **The address of the flash area where the node was read from.** (In the case of being eliminated from the memory, and it was not modified, this address will be registered in its parent in the memory. If it was modified, a new node will be written out, and the old location will be marked as garbage.)
- **Pointers to its children.** Each pointer stores information about whether the child is on the flash or has been read into memory.

6.2 TNC Operations

Using the data structures above, the following operators can be defined:
Search (read):

1. Read the root node of the tree into the memory, then point to it using the pointer p.
2. If p is the desired node, return with the value of p.
3. Find at node p the corresponding child (sub tree), where the desired node is.
4. If the child obtained is in the memory, set pointer p to it, and jump to point 2.
5. The child is in the flash memory, so read this into memory. Mark this child in p as a memory node.
6. Set pointer p to this child, and jump to point 2.

Clean-cache clean-up (e.g. in the case of low memory):

1. Look for an index-node in the memory which has not yet been modified, and for which all of its children are in the flash memory. If there is no such index-node, exit.
2. Set the pointers in the identified node's parent to the original flash address of the node, and free it in the memory.
3. Jump to point 1, if more memory clean-up is needed.

Insert (write):

1. Write out the data as a leaf node immediately. UBIFS writes them out to the BUD area[1], which is specially reserved for leaf nodes, just to make it easier to recover when necessary.
2. Read (search) all of the nodes into memory that need to be modified using the B+ algorithm. (In most cases it is just one index node)
3. Apply the B+ tree modifications in the memory.
4. Mark all modified nodes as dirty.

In the method described above node insertions can be collected, and we can apply them together with significantly lower flash overheads.

Commit (Dirty-cache clean-up):

1. Look for a dirty index node that has no dirty child. If found, call it node n.
2. Write out a new node n onto the flash, including its children's flash addresses.
3. Mark the place dirty where the node n was previously located, and update the flash pointer in the memory representation of the node to the new flash address.

[1] There are two kinds of data earase block in UBIFS: BUD erase block, and non-BUD erase block. UBIFS stores only the leaf nodes in the BUD erase blocks, all other type of data nodes are stored in non-BUD erase blocks.

4. Mark the parent of node n as dirty (if it is not the root node), and mark node n as clean.
5. Jump to point 1 until there is a dirty node.

Deletion:

1. Read (search) all of the nodes into memory that need to be modified using the B+ algorithm. (In most cases it is just one index node)
2. Apply the B+ tree modifications in the memory.
3. Mark all modified nodes as dirty.

7 Power Loss Handling in TNC

In the case of power-loss, the information stored in the memory is lost. To prevent this from happening, UBIFS combines TNC with a journal, where the following information is stored:

– A journal entry with a pointer to new BUD erase blocks. BUD erase blocks in UBIFS are reserved areas for leaf nodes. If the BUD area is full, a new free erase block will be reserved for this purpose.
– Delete an entry after each node deletion.
– A journal entry after each commit with a list of still active BUD areas.

In the event of power loss, the correct TNC tree can be recovered by performing the following steps:

1. Start with the tree stored on flash.
2. Look for the last commit entry in the journal. All of the events that occurred from that point have to be scanned.
3. All of the node insertions stored in the BUD areas marked in the journal, and all of the deletion nodes stored in the journal have to be replayed in the memory.

8 Experiments

Measuring file system performance objectively is not a simple task, because it depends on many factors like the architecture behaviour and caching of the operation system. To avoid theese strong dependencies, we decided to measure just the most important factor of the flash file system performance, namely the size of flash I/O operands to evaluate different TNC configurations and examine their properties using the UBIFS implementation.

The method applied was the following: we unpacked the source of Linux kernel version 2.6.31.4 onto a clean 512M file system, and deleted data using the commands below. During the test, the system counted how many flash operands (in terms of node size) were made with and without TNC.

```
mount /mnt/flash
mkdir /mnt/flash/linux1
tar xfz linux-2.6.31.4.tar.gz /mnt/flash/linux1
rm -rf /mnt/flash/linux1
umount /mnt/flash
```

We measured the performance using different TNC configuration. A TNC configuration has the following parameters:

TNC buffer size: The maximal size of the memory buffer that TNC uses to cache. If it is full, it calls commit and shrink operands. (In our test case, the maximal size of TNC was 23158 (*sizeof(node)), so if the TNC buffer was larger than 23158, shrink was not called.)

Shrink ratio: In the case of shrink, the shrink operand will be called until this percentage of the TNC nodes is freed.

Fanout: B+ tree fanout number: the maximum number of children of a tree node. ($2d$, where d is the order of the B+ tree.)

Table 1. TNC flash operations (measured in terms of node size)

Max. TNC size	Without TNC	With TNC	Shrink Ratio	With TNC / without TNC
5000	2161091	38298	25 %	1.77 %
10000	2211627	31623	25 %	1.43 %
15000	2191395	24632	25 %	1.12 %
20000	2244013	20010	25 %	0.89 %
25000	2192044	12492	25 %	0.57 %
5000	2163769	36273	50 %	1.68 %
10000	2250872	31570	50 %	1.40 %
15000	2225334	22583	50 %	1.01 %
20000	2225334	20002	50 %	0.92 %
25000	2183596	12457	50 %	0.57 %
5000	2215993	36759	75 %	1.66 %
10000	2290769	32578	75 %	1.42 %
15000	2244385	29956	75 %	1.33 %
20000	2238633	20002	75 %	0.89 %
25000	2205709	12958	75 %	0.59 %

Table 1 and Figure 3 show the results of measuring flash performance when the TNC *buffer size* and *shrink ratio* were varied. As can be seen, TNC saves 98.23-99.41% of the flash operands. Increasing the TNC size, more of the flash operations are saved, but varying the shrink ratio has no noticeable effect here.

Table 2 shows what happens if we change the fanout value of the tree. The number of TNC nodes decreases, but the size of a TNC node increases, because a TNC node contains more pointers and keys. The size of the flash operations is the product of these two factors, and it has a minimum fanout value of 32.

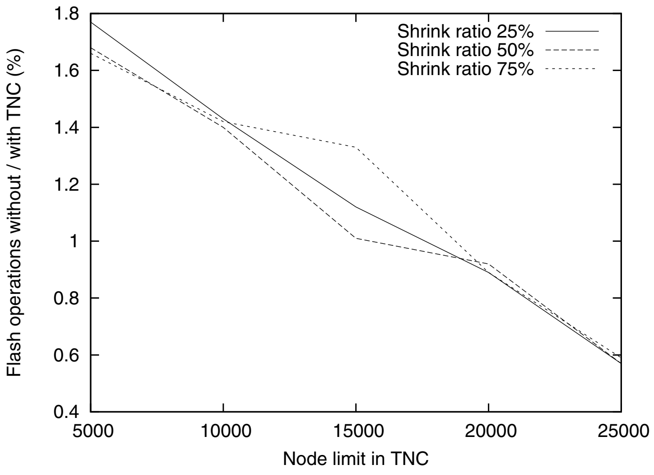


Fig. 3. Performance of TNC flash operations compared to the simple wandering algorithm

Table 2. Effect of varying the TNC fanout

Fanout	Without TNC in nodes	With TNC in nodes	Max TNC in nodes	TNC node size	Max TNC in MB	Flash ops in MB
4	1134784	48392	64801	176	10.88	8.12
8	2168308	12405	23189	304	6.72	3.6
16	1304212	3577	9662	560	5.16	1.91
32	1024363	1317	4669	1072	4.77	1.35
64	1140118	3420	3671	2096	7.34	2.35
128	767005	1245	1586	4144	6.27	3.35
256	930236	1641	980	8240	7.7	4.35

Table 3. TNC size depending on the tree fanout

I/O Size (MB) \ Fanout	8	16	32	64
50	3302	1456	703	351
100	6364	2818	1355	671
200	12925	4620	2224	1106
400	23518	8978	4282	2861
600	43320	18426	8846	5840
800	44948	22070	12273	8527

In the remaining tests we took different samples from the source code of Linux kernel version 2.6.31.4. Table 3 and Figure 4 tell us the maximal TNC size (setting no limit) when the fanout is varied, and the size of the I/O operands (size of the "file-set" above) as well.

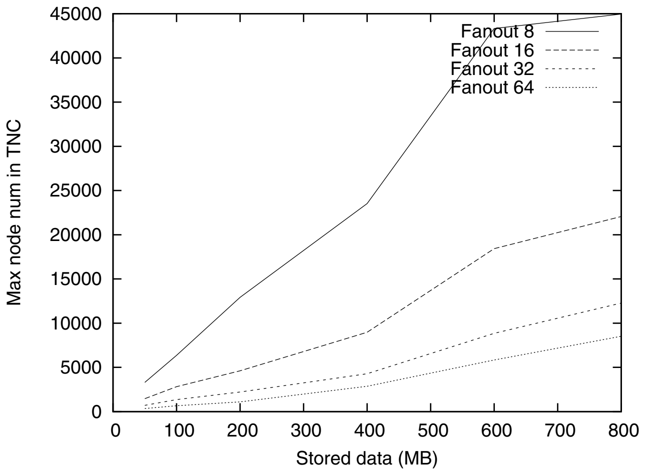


Fig. 4. TNC size depending on the tree fanout

It is very helpful to know its behaviour, especially if we want to use this technique in an embedded system where the system performance and the maximal memory usage of the file system are both of crucial importance.

9 Related Work

The authors of [9] outlined a method that has a similar goal to ours, namely to optimize the B+ tree update on a flash drive. The method collects all the changes in the memory (in LUP = lazy-update-pool), and after it has filled up, data nodes are written out in groups. It also saves flash operations, but unlike our method, using LUP means a lower read speed because, before searching in the tree, it always has to scan the LUP. In the case of TNC, there is usually a higher read speed because the nodes (at least the modified ones) are in the memory. Our method is power-loss safe, but the authors of [1] do not discuss what happens when the information is stored in the LUP. The advantage of their method is the following: the node modifications can be grouped more freely (not just sequentially), so it may be easier (and require less memory) to close the tree operations intersecting the same tree-area.

The goal outlined in [10] is also a B+ tree optimization on a flash memory. It collects as well any changes made in the memory. It calls this area the Reservation Buffer. It is filled up and these changes are written out and grouped by Commit Policy into flash as an Index Unit. It makes use of another data structure called the Node translation table to describe which node has to be transformed by which Index Unit. To search in the tree it is necessary to scan both the Node Transaction Table and the Index Units.

The method described in [5] is essentially an improved version of that described in [10]. Instead of the simple Reservation buffer it utilizes the Index

Buffer, which monitors the tree modifications and if any intersect the same node, it closes them or, where possible, deletes them. In the case of commit, it collects data about the units belonging to the same nodes, and writes them out to one page.

10 Summary

Here the author sought to improve the wandering tree algorithm used by flash file systems, so as to make it more efficient and save over 98% of the flash operands. It has a power-loss safe variant, and it has a much better performance than a simple wandering tree.

The new generation of the Linux flash file system (UBIFS) uses this algorithm and data structure, enabling one to use a flash file system efficiently on 512M or larger flash chips. This implementation is now part of the Linux kernel mainline.

Acknowledgement. This research was supported by the TÁMOP-4.2.1/B-09/1/KONV-2010-0005 program of the Hungarian National Development Agency.

References

1. Ubifs website, http://www.linux-mtd.infradead.org/doc/ubifs.html
2. JFFS: The journalling flash file system. In: Proceedings of the Ottawa Linux Symposium (2001)
3. Bez, R., Camerlenghi, E., Modelli, A., Visconti, A.: Introduction to flash memory. Proceedings of the IEEE 91(4), 489–502 (2003)
4. Comer, D.: Ubiquitous b-tree. ACM Comput. Surv. 11(2), 121–137 (1979)
5. Lee, H.-S., Sangwon Park, H.J.S., Lee, D.H.: An efficient buffer management scheme for implementing a b-tree on nand flash memory. Embedded Software and Systems, 181–192 (2007)
6. Intel: Understanding the flash translation layer (ftl) specification. Tech. rep., Intel Corporation (1998)
7. Jrn Engel, R.M.: Logfs - finally a scalable flash file system
8. M-Systems: Two technologies compared: Nor vs. nand - white paper. Tech. rep., M-Systems Corporation (2003)
9. On, S.T., Hu, H., Li, Y., Xu, J.: Lazy-update b+-tree for flash devices. In: Mobile Data Management. In: IEEE International Conference on Mobile Data Management, pp. 323–328 (2009)
10. Wu, C.-H., Kuo, T.-W., Chang, L.P.: An efficient b-tree layer implementation for flash-memory storage systems. ACM Trans. Embed. Comput. Syst. 6(3), 19 (2007)

Comparing GPU and CPU in OLAP Cubes Creation

Krzysztof Kaczmarski

Faculty of Mathematics and Information Science,
Warsaw University of Technology,
pl. Politechniki 1, 00-661 Warsaw, Poland
k.kaczmarski@mini.pw.edu.pl

Abstract. GPGPU (General Purpose Graphical Processing Unit) programming is receiving more attention recently because of enormous computations speed up offered by this technology. GPGPU is applied in many branches of science and industry not excluding databases, even if this is not the primary field of expected benefits.

In this paper a typical time consuming database algorithm, i.e. OLAP cube creation, implemented on GPU is compared to its CPU counterpart by analysis of performance, scalability, programming and optimisation ease. Results are discussed formulating roadmap for future GPGPU applications in databases.

Keywords: OLAP cube, GPGPU, CUDA, GPGPU optimization.

1 Introduction

Recently we may observe an increasing interest in GPGPU (General Purpose Graphical Processing Unit) systems and algorithms. This new paradigm in parallel programming, placed somewhere between PCs and supercomputers, promises enormous performance with low cost machines. For many algorithms a typical stock graphical device may generate speed up of tens or in some cases of even hundreds times. Numerical problems, computation centric algorithms with good abilities for parallel execution are the most effective. Even the additional programming effort and in many cases a need of reimplementation of whole applications is worth being done. Increasing volumes of data to be processed forces all branches of science and industry to look for new computational capabilities. This paper analyses a GPGPU application in databases on a typical algorithm, i.e. OLAP cube creation including performance, scalability, evolution, programming effort and optimisation ease. Further more, we shall compare it to a similar serialized CPU solution.

Section 2 presents the implementation of the experiment on CPU and GPU. Then, section 3 provide measured results and analysis of both solutions. Section 4 concludes.

1.1 GPGPU Parallel Processing Capabilities

General Purpose Graphic Processing Unit (GPGPU) is a new parallel processing technology developed by manufacturers traditionally colligated with graphics. For example, NVIDIA company introduced CUDA (Compute Unified Device Architecture) [1] technology that can be utilized on many popular graphics cards to perform pure numerical computations. Similar technology is developed by ATI (ATI Stream) [2]. Established consortia work on a standard GPGPU programming language called OpenCL [3].

According to the Flynn's taxonomy [4] vector-based GPU parallel processing can be classified as single instruction, multiple data (SIMD). In other words, the very same operation is performed simultaneously on multiple data by multiple processing units. Since a GPU may have hundreds cores, and each of the cores is capable of running many threads simultaneously, the potential acceleration of processing is huge. Current GPGPU devices go one step further and allow single processors to choose different execution paths – single instruction multiple threads (SIMT) architecture.

Execution of a GPGPU program usually consists of two parts: host code running on CPU and device code (also called a kernel) running on GPU. CPU code may run many classical threads and many kernels asynchronously in the same time (also on multiple GPU devices). This capabilities resulted in variety of interesting applications such as fast sorting, face recognition, FFT implementation, real-time image analysis, robust simulations and many others ([5]). As shown in section 1.2, some attempts to harness the computational power of GPUs into databases, business intelligence and particularly in OLAP applications have also taken place.

1.2 GPGPU and Databases

Among many possible usages of GPGPU programming there were already many successful applications in databases. Due to limitations of this very specific hardware they were mostly focused on relational systems. In 2004, one of the first significant papers presented several algorithms performing fast database operations on GPUs [6]: relational query with predicates, range query, k-th largest number, etc. There is also a SQLite query engine ported to GPU with average speed up at x20 for various queries but with many other limitations [7]. Other authors build a relational join composed of several GPGPU primitives [8]. This approach is very flexible, open for further extensions and low level modifications. Authors observe x2-7 speed-up when comparing to a highly optimized CPU counterpart. Yet another work focuses on external sorting mechanisms which are important for large databases. [9].

Parallel processing of the data for OLAP cubes has been discussed by Raymond in [10], who used PC clusters. Approximately 50% speed-up was obtained and the algorithm was nearly linear where the number of records per processor exceeded 500 000. The other paper by Dehne [11] showed that the cube creation can be paralleled by using multiple CPUs, but the scalability was not linear. These approaches focus on finding an optimal and equal load for all processors in a cluster. GPU parallelism assures automatically optimum load of all cores if only proper conditions when executing a parallel code are fulfilled. Our implementation uses all available cores at 100% of instruction throughput. An analysis of different execution configurations showed that this optimal load of a multiprocessor is done well at the hardware level.

GPU usage for accelerating performance of In-Memory OLAP servers has been proposed and developed by Jedox company [12]. The system is able of huge acceleration of OLAP analyses. This approach is similar to ours but no evidence on detailed algorithm and used data volume are available.

1.3 OLAP Cube Creation Problem

In order to evaluate performance of GPU and CPU we use a typical business intelligence problem: OLAP (On-Line Analytical Processing) cube creation, which is a decision making process important support. These multidimensional data structures contain aggregated data at different levels of aggregation (i.e. sales per years, months or days). Thanks to preprocessing, building reports and ad-hoc analyses can be very quick (”on-line”). However, creation of an OLAP cube may be time-consuming and therefore OLAP cubes often have to be restricted or limited in their size (i.e. scope, meaning that granularity of the cube has to be lower).

Typical OLAP cubes used in professional projects contain millions of aggregations. For example, an OLAP cube designed and deployed at a well-known Polish supermarket ALMA S.A. contains dimensions of hundreds of levels (e.g. SKU dimension). Although the underlying (detailed) data is not of exceptionally huge volume, the cube creation would have lasted 10 hours even on a powerful server ([13]). After optimization of the input data and reducing the number of intersections within the cube this time has been reduced to 5 minutes, but still it gives the idea of the nature of the problem.

The time of OLAP cube creation can be an issue on a heavily used configuration or when the cube has to be frequently updated (Multidimensional OLAP requires rebuilding to update). Even if this is not the case, the possibility of fast OLAP cubes generating can be useful for ad-hoc analyses and reports performed on local machines (even on laptops).

2 Implementation of OLAP Cube Creation

The data in the OLAP cubes is organized into dimensions and measures. Dimensions are categorical variables like year, month, product, region etc. The measures are numerical values e.g. sales amount, transactions count, customers count etc. The process of generating an OLAP cube involves:

1. finding all intersections of the dimensions
2. defining navigation paths according to the specified hierarchies
3. calculating the aggregates

The measures which the aggregates are calculated for may be any additive functions. Typically these are sum, average, maximum, minimum, median, count. OLAP cubes are usually stored as multidimensional tables in the file system. The actual file data structure contains the values of the aggregates for all selected intersections of dimensions. If an NWAY generation was chosen, then all possible combinations of dimensions values will be computed. The input data for an OLAP cube is typically stored as: a star schema; a snowflake schema; a constellation schema or single flat, denormalized data table (*detailed table*).

Since points 1 and 2 from the generation process may be precomputed and derived from database meta data and indexes information we focus on the most changing and time consuming point 3. To simplify memory storage we use only the simplest, yet efficient way to store data: denormalized data table.

2.1 Sequential Approach

The sequential algorithm processes the data table records one by one updating the values of aggregates at the dimensions' intersections. As said before, this process can be time consuming. In real-life situations generation times of several hours can pose some difficulties, especially when there is a need to recreate the cube frequently.

```
procedure calcCubeCPU(cube, data, data_size)
  for k:=1 to data_size do
     intersectionIdx := calcIntersectionIdx(data[k])
     cube[intersectionIdx] := calcAggregate(cube[intersectionIdx], data[k])
```

Fig. 1. A sequential algorithm. k denotes indexes of input data set while `calcIntersectionIdx` is responsible for calculating appropriate cube intersection upon input data. `calcAggregate` calculates a new aggregate value from the previous value and new data (min, max, sum, etc.).

Speed evaluation of the above very simple sequential solution for flat denormalized table shows that it is very fast if all data is stored in RAM. Here, we do not discuss how data was transferred into RAM memory. For comparison with GPU, which is anyway dependent on RAM, it is not important what techniques, if any, were used, and what is the physical representation of the database. Since RAM is now cheap and RAM-only databases are more and more popular, for this experiment we can assume that we only deal with an in-memory database. Memory caching, internal processors' optimizations and optimal memory reads and writes make CPU implementation really hard to be beaten by a parallel procedure, which in most cases has to perform additional tasks. Consumed time measurements for the sequential algorithm compared to parallel ones are presented in fig. 5.

2.2 A Naive GPGPU Solution

A typical way of creating a GPGPU algorithm is to start from a naive solution and than optimize data structures, implementation or the algorithm itself.

A naive parallel implementation of a sequential procedure leads to a very simple, yet very inefficient algorithm. If we consider a single database entry to be processed by a single SIMD processor, all the data is computed (ideally) in the same time. In the first step each processor reads its input data, in the second step computes a cube intersection to be updated, and in the third stores appropriate value (see fig. 2). This algorithm behaves poorly (about x20 times slower than CPU) because of many conflicts between threads trying to update the same cube intersection in the same time. Collisions may be solved by atomic addition function (available in NVIDIA CUDA). However, atomic operations degrade memory bandwidth by serialization of parallel reads and writes. It is also impossible to calculate all kinds of aggregate functions by currently available hardware level atomic functions. One can try to implement software level exclusive writes [14] but this again limits parallelism.

```
procedure calcCubeNaiveGPU(cube, data, data_size)
1   forall 1<=k<=data_size in parallel do
2      intersectionIdx := calcIntersectionIdx(data[k])
3      cube[intersectionIdx] := calcAggregate(cube[intersectionIdx], data[k])
```

Fig. 2. A naive parallel algorithm with many write conflicts. k denotes indexes of input data set, processed concurrently, while `calcIntersectionIdx` is responsible for calculating appropriate cube point upon input data. `calcAggregate` calculates a new aggregate value from the previous value and new data.

2.3 Improved GPGPU Solution

Much better results can be obtained if only conflicts in accessing cube intersections could be removed. They appear, because many threads read the data influencing the same point in the cube. But if we could assure that a single thread gets all the data for the single point, and only for this point, then there will be no writing delays. All threads could do their jobs independently. At this point we should notice that if the data is sorted, then all inputs needed for a single cube intersection's calculation lay together. Then the only difficulty is to find these sequences (clusters) in the input data. This can be hard for a sequential process but it is easy for the parallel SIMD algorithm. A single thread reads two data records, say number n and $n-1$. Then it calculates cube intersections for them (c_i, c_j). If $c_i = c_j$ then it means that index n is in the middle of a sequence for given cube intersection, but if $c_i <> c_j$ then there is a border of sequences between $n-1$ and n. The rest of the computations is as follows. Having the beginning and end points of all the sequences for a given data set, we run another parallel computations with one thread responsible for processing the single cube intersection's aggregation (fig. 3).

Algorithm Optimization. This algorithm is GPGPU efficient in a sense that it uses the hardware according to its limitations. It is also theoretically efficient since its parallel computational complexity is obviously asymptotically not higher than the sequential one.

```
procedure calcCubeImprovedGPU(cube, data, data_size, cube_size)
1   forall 1<k<=data_size in parallel do
2      intersectionIdx1 := calcIntersectionIdx(data[k-1])
3      intersectionIdx2 := calcIntersectionIdx(data[k])
4      if (intersectionIdx1 <> intersectionIdx2)
5         begins[intersectionIdx2] = k
6         ends[intersectionIdx1] = k-1
7   begins[calcIntersectionIdx(data[0])] := 0
8   ends[calcIntersectionIdx(data[data_size]) := data_size-1
9   acc := 0
10  forall 1<=c<=cube_size in parallel do
11     for begins[c]<=i<=ends[c] do
12        acc := calcAggregate(acc, data[i])
13     cube[c] : = acc
```

Fig. 3. An improved parallel algorithm without write conflicts. Assuming sorted input data set. `k` denotes input data records. `c` denotes available cube intersections.

However, a closer look at the first part of the algorithm in fig 3 shows that each thread reads two subsequent input data records. When multiplied by all n threads we get all data records read twice, which is a waste of the global memory bandwidth. This can be avoided only by using a thread block's shared memory as a short-term cache for the global memory data reads. On-chip shared memory is about 100 times faster then GPU global memory. Although this technique required programming manual transfers between global and shared memory, a significant speed-up was observed. We should note there that limitations of fast on-chip memory forced major changes in the algorithm itself. Its architecture changed from straightforward data oriented to a cluttered memory-caching.

An important optimization for SIMD processing is coalesced memory reading and writing (non-coalesced read may degrade memory access even by x16 times). For NVIDIA devices, coalesced memory access is possible if threads within the so-called half warp (16 threads for current devices) read from the same memory segment (length from 32 to 256 bytes). If these circumstances are fulfilled, all data read or write for all 16 threads is done in single instruction, which is a great improvement for parallel algorithm, when compared to sequential one and CPU. In case of our GPGPU algorithm we perform a parallel access of global memory in lines 2, 3, 5, 6, 11, 12 and 13. In lines 2, 3 and 12 reading input data may be properly organized to assure coalescing. Also reading sequence indexes in line 11 may be coalesced. Accessing cube cells for writing is rather random or we could say data dependent. Coalescing for lines 5, 6 and 13 cannot not achieved due to random character of cube intersections writing. The most important problem with this optimization is that it is highly data-size dependent. If structure of records changes (even by 1 byte) all the potential of coalescing may be lost. It could be very hard, or even impossible to achieve a general and generic coalesced memory access for varying record sizes.

Another possible place for optimization of the algorithm is calculation of aggregation value for given intersection, which is done in lines 11 and 12. This simple *for* loop construct performed by one thread during optimization was exchanged with a parallel efficient reduction [15,16] on a set of processors with logarithmic time complexity (observed a 2x speed-up).

All the mentioned optimizations of the parallel algorithm (especially shared memory usage) make it much longer, less readable and too complicated to be presented in a pseudo code version here. Its performance, which is overall about 10 times faster than CPU one is described in the section 3.

At this point we should have a look at the CPU algorithm again. Comparisons of CPU and GPU algorithms are often not fair for CPU using not optimized code and not efficient solutions [17]. We can notice that our CPU procedure is really not efficient from the current double core CPU's point of view. If we use all the cores in GPU running parallel threads we should do the same with CPU, especially if we consider that all currently available CPUs contain at least two, four or more cores. This conclusion heads us toward a parallel CPU procedure.

```
procedure calcCubeParallel(cube, cube_size, data, data_size, no_threads)
    ExecutionPlan plan[no_threads]
    defParallelPlan(plan, data, data_size, cube, cube_size, no_threads)
    for 1<=i<=no_threads in parallel do
      calcCubeCPU(plan[i].data, plan[i].cube, plan[i].data_size)
```

Fig. 4. A CPU parallel algorithm. `defParallelPlan` procedure takes input data and an array of `execPlan` and defines data ranges for independent threads. Then all threads with different execution plans are started.

2.4 Parallel CPU Algorithm

A CPU parallel algorithm with many threads is almost as simple as sequential one. If we have sorted input data then we can, in almost constant time (dependent only on number of threads which is constant), find independent partitions to assure that concurrent threads will not access the same cube intersection. This task is done by procedure `defParallelPlan` in the fig. 4. It returns execution plan for each CPU thread containing input data partition, output cube section, data ranges, etc.

According to our expectations the multiple threads CPU algorithm behaves properly when run on a dual core processor. Two threads increase the speed by about 40%. Also as it was expected, four threads do not change anything when run on thwo cores since two of four threads must wait for the processor's time all the time.

2.5 Parallel GPU Devices Algorithm

If we have successfully executed a parallel CPU procedure on double core processor, can we do the same with two graphical devices? Modern motherboards and devices with double GPU's (like GeForce 295GTX) allows up to eight GPUs in single PC box. This would give us a tremendous speed up.

CUDA documentation says [1] that for each graphical device one must execute single CPU thread which keeps the context and host code communicating with the dedicated device. So, for two devices it seems fine to have a dual core CPU. For four devices, a quad core system seems to be more sensible. A multiple GPU test was created exactly in the same way as parallel CPU code except for the line 4, where we call a GPU calculation algorithm. Limited space does not allow us to present this straightforward procedure here.

Surprisingly, the results are exactly opposite to what we could expect. The overall execution time of a cube calculation on two GPUs was longer than with one GPU! This problem is widely discussed by CUDA programmers and seems to be a limitation of current popular motherboards and CPUs. There are some anticipated changes in Intel's Nehalem i7 architecture but not yet evaluated widely by the community. Also cost of this most powerful chip together with appropriate GPU devices places all the system rather in advanced business market not an average customer which is against the assumption of bringing supercomputer power to the masses.

Why multiple GPU performance is so bad? The reason is in the specific task we execute. As many other database algorithms it is highly data intensive. Processing power is not so crucial. The most of the procedure is just reading or writing data. Also important part of the code is only copying data from RAM to the device's global memory. This is where our program meets the PCI Express (Peripheral Component Interconnect Express) standard bottleneck. Although version 2.0 of the standard works with x16 speed that is about 3GB/s. If we plug-in a double graphic card in a single slot it degrades to x8. Therefore, the resulting times must be much worse than for a single GPU. But database applications are almost always data intensive and scalability of GPGPU database solutions for multiple devices is right now questionable.

3 Results

3.1 Experiment Set Up

The test environment constituted a machine with Intell's 3Ghz CPU Core2 Duo, 4GB of RAM and single NVIDIA GTX 295 card. This GPU device is build of two multiprocessors (1.242 GHz, 240 cores and 896MB of DDR3 memory each). Sequential algorithms was executed on CPU while parallel algorithm on one or both GPU devices.

The data input used for testing was a flat data table (detailed table) with 25 millions records. Each record contained fields: *Year*, *Month*, *Day*, *Group*, *Product* and *Amount*. The data has thus 2 dimensions: time and product with 3 levels (year, month, day) in the time dimension and 2 levels in the product dimension (product group, product). The measure used was a sum of sales amount. The resulting OLAP cube with the highest granularity with all possible intersections contained 1 424 016 aggregations.

In the experiment, for CPU code pure C as the only programming language, while for GPU CUDA for C from SDK version 3.0 was used.

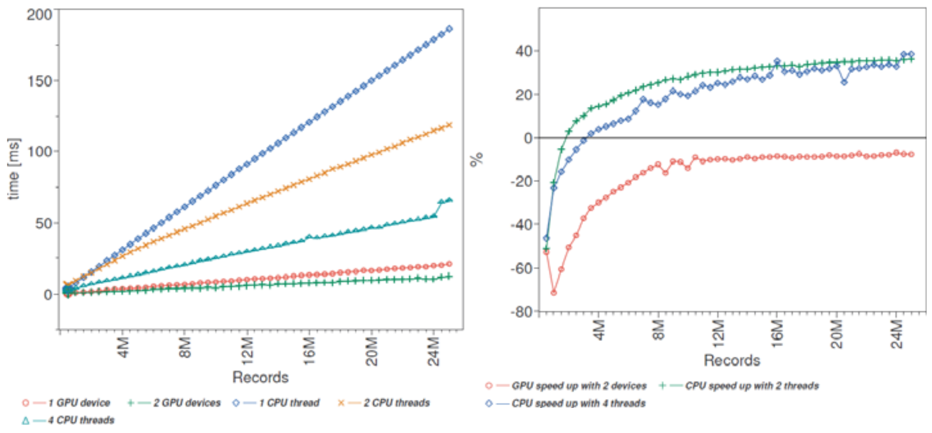


Fig. 5. (left) Running time against data sample size. **(right)** Performance improvement when adding more threads/GPU devices.

3.2 CPU and GPU Performance

The experiment was run on subsequent samples of the input data from 1 to 25 millions records. We divided performance comparison into two parts. The first one focuses only on pure algorithm running times, the second compares scalability of both algorithms and executing hardware.

The graph of running times, including only pure algorithm execution, without any additional set up operations, are given in the fig. 5 (left). We measured only the time spent by each thread inside the cube calculation procedure. Therefore we performed the experiment with five different execution set-ups: one, two and four CPU threads, one and two GPU devices. Since the algorithm is linear and can be well divided into independent partitions the running times as expected present a speed up of about 50% when run in multiple threads and multiple devices configurations, as expected.

The GPU implementation of the algorithm is about 10 times faster than the CPU one. Single GPU peek performance processes more than 1 million records per millisecond, while single CPU thread achieved only about one hundred thousands records per millisecond.

However, we must point out here that the GPU algorithm requires time consuming transfer of data between RAM and graphical device's global memory. This very time consuming operation performed over PCIE bus interface consumes much more time than the algorithm itself and for 25M records took about 340ms (17 times longer than the algorithm itself), which make the overall GPU execution pointless if the cube generation, or any other operation, is the only task to be done. This problem is mostly not mentioned by many authors who often show only "computation time". Utilization of GPGPU in databases may be sensible if data may be stored in the device's memory for longer time. Only device memory based data storage and execution engine working both on CPU and GPU sides could properly handle this mission.

The second analysis presents results of the algorithm and execution engine (CPU or GPU) scalability when switching from single to multiple processors environment. The fig. 5 (right) presents percentage improvement of overall procedure execution time (including set up operations, memory allocation, cleaning, data copying forth and back). As expected we can see good improvement around 40% for CPU, while for GPU we can observe degradation of overall speed. As it was said earlier this problem is caused by PCI-Express interface which slows down for concurrent data transfers. The final results for two graphical devices running in parallel are about 5% slower than one graphical device. The only benefit be could observe of using two devices is the doubled memory capacity, which would enable us to execute tasks of more than 64M records.

4 Conclusions

In this paper we analysed CPU and GPU implementation of a typical database computational algorithm. Both CPU and GPU code was highly optimised to assure peek performance.

When comparing just database records processing time, the GPU version proved to be significantly faster than its sequential CPU counterpart. In case of NVIDIA CUDA capable devices number of threads physically executed in the same clock cycle varies from 8 for the simplest mobile GPUs to hundreds for the most advanced ones (like GPUs from Fermi or Tesla lines). In this case, since most of the algorithm is concurrent, Amdahl's law promises great performance and it can be observed.

Our experiments proved that the practice of GPGPU programming is far from theoretical capabilities if special care on implementation is desisted. Procedures need to optimized at a very low level. The best results can be achieved only for full memory bandwidth and instruction throughput.

The CPU version was much easier to be codded. It took only about 2 hours to get it working with two threads, while GPU code took about 2 weeks of debugging, profiling and optimizing. Although, there are API interfaces for many programming languages still GPGPU programming is highly limited by hardware capabilities. An inexperienced programmer may face an unbreakable barrier of shared memory banks conflicts, memory bandwidth, number of registers per processor, templates meta programming, limitations in task synchronization or inter task communication.
Predicted advantages of utilization of GPU devices in databases:

- Enormous instruction throughput and data bandwidth. If there is a possibility of storing all the database in a GPU device memory (the most advanced cards may offer together up to 16GB) one may expect really spectacular results by minimizing time costly transfers between CPU RAM and device. However, there is no clear idea how a generic database storage can be organized to optimize GPU performance.
- Possible speed up of algorithm building blocks when using many well documented APIs with primitives like sorting, prefix-sums, array packing, etc.

- A GPU device may be often exchanged or multiplied up to 8 devices at low cost. Even an average PC can be equipped with a high performance GPU device achieving a low cost excellent improvement in performance.

Among properties of GPU devices we should enumerate sources of potential problems for database management systems:

- GPGPU hardware and its programming languages are still at the beginning of the evolution process. Programming is difficult and very low-level, highly bound to hardware capabilities and internal construction of a particular device.
- Implementations of memory intensive algorithms must be optimized for given, fixed input data. This is very unpleasant for database applications, which must cope with very different and changing data. Moreover, relational databases with well defined, fixed columns are not the only ones on the market. Unfortunately, due to the highly vector-like processing nature, GPUs are not yet ready for unstructured data. So, there is no well known API which could help to organize a general database storage and assure peek GPU performance by for example memory coalesced reads and writes.
- There is no clear way how stored procedures can be implemented at the GPU side. The same problem arises for object databases when objects' behaviour is to act concurrently for many objects in the same time.
- There are so far no evidences that indexes as structures, which must be often randomly accessed and modified by many threads in the same time, can be currently efficiently implemented at GPGPU side.
- Although a single thread in NVIDIA CUDA is executed independently from other threads, a decrease of efficiency may be observed for highly branching algorithms: when a single thread computes its branch, other threads at given streaming processor must wait.
- Scalability of multiple devices systems suffer from PCI-E interface limitations and may slow down the overall application.
- ACID properties need independent host threads running on single or many devices in the same tine. To be evaluated if possible with current GPU devices at proper level.

Last but not least, we believe that GPGPU processing will be important for future DBMSs and we expect effort of both GPU and database communities to achieve a significant cooperation of both technologies.

References

1. NVIDIA Corporation, CUDA programming guide (2009), www.nvidia.com/cuda
2. ATI Corporation, ATI stream sdk v2.2 documentation, http://developer.amd.com/gpu/ATIStreamSDK
3. Khronos Group, OpenCL - the open standard for parallel programming of heterogeneous systems, http://www.khronos.org/opencl/

4. Flynn, M.J.: Some computer organizations and their effectiveness. IEEE Transactions on Computers C-21, 948–960 (1972)
5. NVIDIA Corp., CUDA C posters, www.nvidia.com/object/SC09posters.html
6. Govindaraju, N.K., Lloyd, B., Wang, W., Lin, M.C., Manocha, D.: Fast computation of database operations using graphics processors. In: SIGMOD Conference, pp. 215–226. ACM, New York (2004)
7. Bakkum, P., Skadron, K.: Accelerating sql database operations on a gpu with cuda. In: Kaeli, D.R., Leeser, M. (eds.) GPGPU. ACM International Conference Proceeding Series, vol. 425, pp. 94–103. ACM, New York (2010)
8. He, B., Yang, K., Fang, R., Lu, M., Govindaraju, N.K., Luo, Q., Sander, P.V.: Relational Joins on Graphics Processors. In: Wang, J.T.-L. (ed.) SIGMOD Conference, pp. 511–524. ACM, New York (2008)
9. Govindaraju, N.K., Gray, J., Kumar, R., Manocha, D.: GPUTeraSort: high performance graphics coprocessor sorting for large database management. In: SIGMOD, pp. 325–336 (2006)
10. Raymond, T., Wagner, A., Yin, Y.: Iceberg-cube computation with pc clusters. In: Proc. ACM SIGMOD Conf. (2001)
11. Dehne, S.H.F., Eavis, T., Chaplin, A.: Parallelizing the data cube. In: Proc. Eighth Int'l Conf. Database Theory (January 2001)
12. Lauer, T., Datta, A., Khadikov, Z.: A CUDA-powered in memory OLAP server. NVIDIA Research Summit (2009)
13. Koral, K.: Benefits from BI at ALMA. In: SAS Business Forum, Poland (2007)
14. Shams, R., Kennedy, R.A.: Efficient histogram algorithms for NVIDIA CUDA compatible devices. In: Proc. Int. Conf. on Signal Processing and Communications Systems (ICSPCS), Gold Coast, Australia, pp. 418–422 (December 2007)
15. Blelloch, G.E.: Prefix sums and their applications. In: Sythesis of parallel algorithms, pp. 35–60. Morgan Kaufmann, San Francisco (1990)
16. Harris, M.: Optimizing parallel reduction in CUDA (2008)
17. Lee, V.W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A.D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R., Dubey, P.: Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu. SIGARCH Comput. Archit. News 38(3), 451–460 (2010)

A Power Consumption Analysis Technique Using UML-Based Design Models in Embedded Software Development*

Doo-Hwan Kim[1], Jong-Phil Kim[1], and Jang-Eui Hong[2]

[1,2] Chungbuk National University, Dept. of Computer Science,
410 Sungbongro, Heungdukgu, Cheongju, 361-763, Rep. of Korea
{dhkim,kimjp}@selab.cbnu.ac.kr, jehong@chungbuk.ac.kr

Abstract. Although the power consumption of embedded system depends on the operation of hardware devices, software behaviors give great effect to the power consumption because of its functionality and complexity growth. This paper proposes a power consumption estimation technique using design models of software to support energy-efficient embedded software development. Even though code-based power analysis techniques have been proposed, these techniques have demerits that the analysis time is long and feedback is not easy. Our proposed technique makes use of UML behavior models for the power consumption analysis in order to overcome the demerits of code-based analysis. When comparing with the existing code-based analysis, our technique can provide the power analysis result at earlier phase than implementation. Therefore, software engineer can apply our technique to select energy-efficient design decisions in embedded software development process.

Keywords: Power consumption analysis, UML models, Embedded software.

1 Introduction

Embedded systems have a limited power supply because they make use of limited hardware devices and used in wireless environment, frequently. Accordingly, the importance of power consumption management has been emphasized in the practical application of embedded systems. So, there have been hardware-driven studies that develop long-lasting battery and low power device [1]. However, since software is recently forming a greater part of embedded system and becomes complicated in its functionality, several researches became interested in the power-related works of embedded software [2]-[3].

The existing studies on power consumption analysis of embedded software have been conducted in the level of instructions or source codes. However, because of a large number of instruction cycles and its fine-grained analysis, these approaches

* This work was supported by the Korea Research Foundation Grant (KRF-2008-313-D00936) and also partially supported by the MKE, Korea, under the ITRC support program supervised by the NIPA (NIPA-2010-(C1090-1031-0001)).

have shortcomings that the analysis time is long and the feedback of the analysis results is not easy, even though the analysis result is appropriate[3]-[9]. To complement these weaknesses, some studies have been conducted on model-based power analysis that provides the advantages of high power saving and fast analysis time, as shown in Fig. 1.

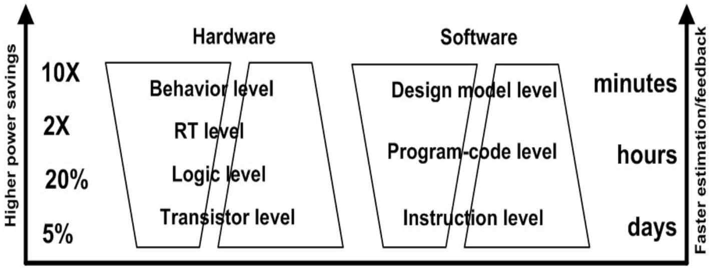


Fig. 1. Power analysis efficiency by abstraction levels [10]

Most studies on model-based power analysis are mainly performed using specific models which represent the power properties of embedded software [10]-[12]. However, these studies have hassles that not only require additional efforts to develop the model for power analysis only, but also difficult to reflect the analysis result to software design models. Therefore, this paper proposes a power consumption analysis technique using design models based on UML 2.0 [13] without developing any other specific model in the process of embedded software development. Our proposed technique can help designers to select efficient design decisions in the early phase of a development. Also, it provides the advantage of time reduction for power consumption analysis in comparison with source-code based techniques.

The rest of the paper is organized as follows. In Section 2 we survey the related work. Section 3 explains the process of a model-based power consumption analysis which we are proposed. In Section 4, we illustrate the experiments of our technique and the results we obtained. Finally, in Section 5 we conclude this paper and present future work.

2 Software Power Analysis

The techniques of software power analysis are categorized into 3 classes: instruction-based analysis, code-based analysis and model-based analysis. There are many distinguishable researches in each category, but we survey on model-based analysis techniques which are related with our research approach.

The power analysis techniques using software model are proposed by Tan [10], Jun [12], Dhouib [14] and Kim [15]. Tan analyzed the energy consumption of embedded software considering with the communication between tasks, which compose the architecture of embedded software. The architecture is represented by SAG (Software Architecture Graph), which nodes indicate executable tasks and the edges indicate the communication links. After building the SAG, Tan showed that energy consumption can be reduced by graph reduction technique which uses to decrease the number of tasks and communication loads. However, even if Tan made use of the SAG as a software model, the nodes (tasks) in the graph are implemented with C language to estimate the power consumption.

Jun [12] estimated the energy consumption of embedded software using the energy interface automata which models the states of a system. After defining the internal behavior of software component with the energy interface automata, they assigned energy consumption rates to the nodes and edges of the automata and then analyzed the energy consumption through automata simulation. However this technique has a limit of that the values of the nodes and edges, i.e., energy vector values were randomly assigned.

The technique developed by Dhouib [14] is to analyze the energy consumption of software system using an extension of AADL (Architectural Analysis and Design Language). After describing software and hardware architecture models, the software model is deployed to hardware model to analyze energy consumption of a system. Although this paper claimed that the power consumption is analyzed with the data transfers that handled with IPC mechanisms, the technique is focused on the software model deployment onto the hardware model without the suggestion of an energy analysis technique.

Although above techniques are model-based power consumption analysis approaches, additional effort is required to create the analysis model in the process of embedded software development, and also did not provide the seamless approach to support UML-based software modeling in model-driven development framework [16]. Kim [15] proposed UML model based power analysis technique. This proposition is our prior research and it just focused on the establishment of energy library and its usability.

3 Proposed Estimation Process

The power consumption analysis technique of embedded software proposed in this paper uses the behavioral models of UML 2.0[13]. The sequence diagram of UML can represent the details of software behaviors depending on the flow of time, and can describe the internal actions of a specific function with an action language [17]. Therefore we design the behaviors of embedded software using UML such as class diagram (CD), sequence diagram (SD), interaction overview diagram (IOD), and action language (AL), and then analyze power consumption using these models. Such behavioral models can intuitively depict the functional behaviors of embedded software, can occur in real operation.

Our estimation process of model-based power consumption is shown in Fig. 2. The estimation process is started with UML design models, and the models transformed to CFG (Control Flow Graph) [18]. Through the traversal of the CFG, we find an EBU (Energy Behavioral Unit), which is defined as the basic unit of power consumption analysis. The sum of the power consumptions of every EBU is our estimation result.

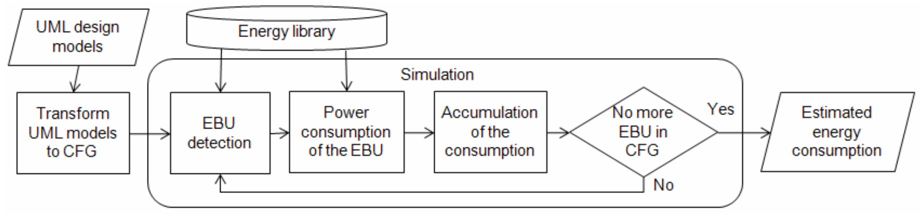

UML design models
Energy library
Simulation
Transform UML models to CFG
EBU detection
Power consumption of the EBU
Accumulation of the consumption
No more EBU in CFG
Yes
No
Estimated energy consumption

Fig. 2. Model-based power consumption estimation process

3.1 Transform UML Model to CFG

The design model represented with UML is transformed to CFG which is a connected graph and has strong expressiveness for behavioral sequence of software function. This transformation is to unfold the hierarchical structure of UML diagrams, or to integrate diagrams drawn in several pages into one. Thus the CFG reveals synchronous action, asynchronous action, parallel action, branch action, fork and join actions, and etc which are inherent in the UML diagrams.

Using the CFG is easier to identify power consumption factors than UML models. The CFG is used as the input of simulation that calculates the power consumption of software design model according to the behavior of software. The algorithm for creating CFG from UML models is presented in detail by the study of Garousi [18]. We define here the node structure of our CFG as follows:

```
  Struct CFG_NODE {
    Node_id := node identifier;
    Predecessor_Node := node identifiers;
    Successor_Node := node identifiers;
    BelongTo := diagram_name.object_name;
    Guard_Cond := condition clause to control the behaviors alt, loop,
etc, appeared in SD.

    Node_Type := UML element node type;
    Node_Type_Start := boolean;
    Node_Type_End : = boolean;

    // to represent method call or action language appeared in SD.
    Included_Category := {FunCall, ActionLang}
    Included_Token_Set : = a linked list of terms, appeared in FunCall
or ActionLang;
   } End_Struct
```

3.2 Building Energy Library

The energy library as shown in Fig. 2 is a component of infrastructure that should be constructed in advance for our power consumption analysis. Our energy library is based on EBU, and defined as follows;

[Definition] Energy library EL is defined with a tuple,

$$EL = \langle E, V, \alpha \rangle \tag{1}$$

where, E is a set of EBUs, V is a set of virtual instructions, and α is a set of mapping actions such that $e \in E \mapsto \{ v \mid v \in V \}$.

The EBU is the basic unit derived from the behavior analysis of UML model elements to calculate power consumption. To elicit the EBU, we investigate UML meta-models which define all structural components of UML diagrams [13]. EBUs are corresponding to class identifiers from the meta-models, and the identifier has its behavioral operations. This means that EBU can consume electric power by the operations. The identified EBUs are stored into our energy library, and the list of EBUs is shown in Table 1.

Table 1. List of EBUs

Diagrams	EBUs
SD	Message (send, receive), MessageSort (SynchCall, AsynchCall, Signal), InteractionOperator (alt, opt, loop, par)
IOD	Fork, Join (wait), Invocation (Create), InterruptableActivityRegion
AL	Bitwise Operator (+, -, *, /, %, <<, |, !, ==, &&, >=, etc.), Function Operator (read, write, malloc, etc.)

Virtual instruction, V in Definition (1) is a set of generalized instructions, which is defined to estimate power consumption caused by a program execution. The notion of virtual instruction is introduced by Bommie [19]. We extend the notion to discriminate instruction types. Our virtual instructions are divided into Virtual Primitive Instruction (VPI) and Virtual System Function (VSF). The VPI is also a set of generalized assembly instructions created from compiling AL on an ARM processor and VSF is a set of generalized system functions that are provided in Embedded Linux Operating System. The list of virtual instructions is shown in Table 2. Even though more instructions can be identified than those of Table 2, those instructions can be implemented with the composition of the instructions in Table 2.

Table 2. List of virtual instructions

Types	Instructions	
VPI	load, store, add. sub, call divide, call module, mult, compare, convertType, branch, bitNOT, bitAND, bitOR, bitXOR, bitShiftLeft, bitShiftRight	
	Process Manager	fork(), waitpid(), wait(), signal()
VSF	IPC	msgsnd(), msgrcv(), msgget(), msgctl(), semget(), semctl(), semop(), pipe(), pipe open(), pipe write()
	File System	fileopen(), fileclose(), fileread(), filewrite()
	Memory Manager	shmget(), shmat(), shmdt(), shmctl()

The power consumption of these virtual instructions is profiled by EMSIM 2.0 simulator [20]. EMSIM simulator generates the values of power consumption in nano Joule (nJ) which reflects the platform specific features. To obtain the power consumption for every virtual instruction, we develop test codes for each VPI and VSF by operand type and by parameter size, respectively. The simulation is performed 200 iterations for each operand types of char, short, int, float, and double and for each parameter size of 1, 2, 4, 8, 16, ..., 2048 bytes.

Table 3 shows the sample of energy profiling results for virtual instructions. The whole profiling results can find at [21].

Table 3. Estimated power consumption for VPI

VPI	Type	Energy(nJ)	**VSF**	Parameters	Macro-model (nJ)
load	char	14.1	fork()	-	132202.6
	short	28.1	waitpid()	-	25077.3
	int	13.0	wait()	-	24986.6
	float	13.0	signal()	-	9460.6
	double	13.7	File open()	-	17886.6
store	char	13.2	File close()	-	8262.2
	short	26.4	File read()	χ bytes	$5.1\chi + 49308.5$
	int	12.5	File write()	χ bytes	$5.6\chi + 32022.3$
	float	12.8	shmget()	-	126358.8
	double	13.2	shmat()	-	11599.9

Each EBU is mapped with the combination of one or more virtual instructions at least. This mapping is performed by refining the EBU in the same instruction level as VPI and VSF. For example, an EBU "alt" is composed of the detail operations which are loading the condition variables, comparing the variables, and branching to result point. Therefore, the EBU "alt" is mapped with three VPI instructions of load, compare and branch. By using this energy library, we can acquire the power consumption of each EBU.

3.3 EBU Detection

To detect an EBU from the CFG, we traverse every nodes of the CFG sequentially. Using the definition of CFG node structure, described in Section 3.1, the EBU detection algorithm is shown in Fig. 3.

```
Algorithm : EBU Detection
Input : CFG consisted with m nodes;
Output : detected EBU list EL = {$e_1$, ..., $e_l$};

1.  Let S is selected node in CFG, initially S = first node;
2.  Let E is a set of EBUs from energy library;
3.  Let m is a number of nodes in CFG;
4.  Let n is a number of elements of E;
5.  for i = 1 to m do
6.     if the Node_Type_Start of S is true then
7.        for j = 1 to n do
8.            compare the Node_Type of S with $e_j \in E$;
9.            if the Node_Type correspond to $e_j$ then
10.              add $e_j$ into EL;
11.           end if
12.       end for
13.       S = Successor_Node of S;
14.    else
15.       S = Successor_Node of S;
16.    end if
17. end for
18. return EL;
```

Fig. 3. EBU detection algorithm

Like the algorithm depicted in Fig. 3, to be used as a criterion for EBU detection, the selected node of CFG is defined as S, which is a node of CFG. The term Node_Type_Start indicates that a node is the start node of CFG transformed from a UML element. That is used to select a criterion node for EBU detection. And the comparison between the node type of the S and EBUs is performed to find the corresponding EBU form energy library. Here we assume that an EBU in accordance with the node type always exists. The detected EBU is added into the EBU list, and the selected node is defined as a successor node of S. Finally, the EBU list will be returned as the result of the algorithm.

3.4 Power Consumption of EBU

If an EBU is detected from the CFG, the power consumption of the EBU is calculated by getting the energy values of the virtual instructions which mapped with the EBU within energy library. By repeating EBU detection until the traversal reaches at the final node of the CFG, we can calculate the total power consumption, TEC of embedded software as follows.

$$TEC = \sum_{i=1}^{l} EBU_i (\sum_{j=1}^{m} Evpi_j + \sum_{k=1}^{n} Evsf_k)$$

The total amount of power consumption, TEC is sum of energy values of all EBUs which can be calculated with the energy values *Evpi* and *Evsf* of VPI and VSF instructions for an EBU, respectively. In this calculation, the power consumption for the *Evpi* and the *Evsf* by loop iteration and parameter size is calculated prior to the final sum.

4 Experiments

We have experiments for our power analysis technique with some applications to show its applicability. In this section, we first explain our experiment environments, and then we show the experiment results to answer to following three questions; How about the accuracy of our analysis results with comparison of source-code based analysis results? How about the speed-up in the analysis time of our techniques? And how to apply our technique in embedded software design process?

4.1 Experiment Environments

We develop a tool to analyze power consumption of embedded application, which is called ESUML-EA (Embedded Software modeling with UML – Energy Analyzer). The ESUML-EA tool [21][22] is developed with Java-Eclipse 3.5 in Windows 2000, and supports software modeling with UML 2.0 and estimates the power consumption of the software. The code-based power analysis tool which we use to compare the analysis results is EMSIM 2.0 [20]. EMSIM 2.0 simulates C code execution based on strong-ARM processor architecture and embedded Linux kernel 2.4.x. There are four target applications in our experiment, which are popular functions in mobile products as follows: (App 1) data retrieval form phone book, (App 2) shortest path selection in road-navigator, (App 3) translate image [23] in digital camera, and (App 4) image encoding [24] using Huffman coding.

4.2 Accuracy of Model-Based Analysis

At first, we examine the accuracy of our proposed technique using four mobile applications to verify the applicability for practical embedded software development. In order to perform the experiment, we develop UML design models for those applications using ESUML-EA, and then analyze the power consumption of these applications with the design models (UML diagrams). Fig. 4 and Fig. 5 show one of the diagrams and analysis result for App 4 in ESUML-EA screen.

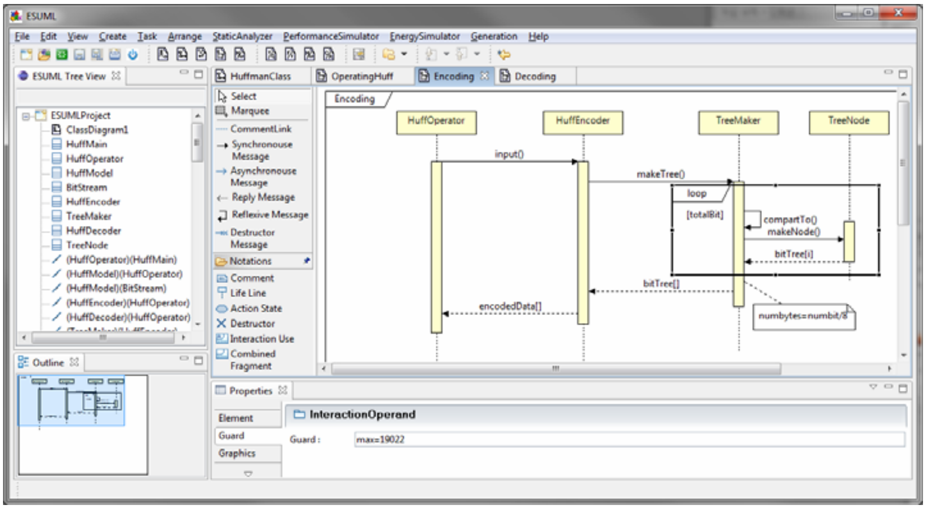
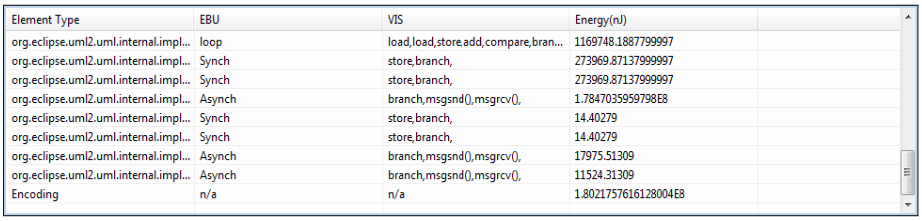


Fig. 4. A diagram of App 4 using ESUML-EA

Fig. 4 shows a sequence diagram of the encoding behaviors. In the diagram, the object TreeNode has their detail action which represents with action language, appeared in annotated notation. Also the loop construct can iterate until the given max number appeared in below property window. The power consumption for every EBU which consists of the encoding behaviors is shown Fig. 5.



Element Type	EBU	VIS	Energy(nJ)
org.eclipse.uml2.uml.internal.impl...	loop	load,load,store,add,compare,bran...	1169748.1887799997
org.eclipse.uml2.uml.internal.impl...	Synch	store,branch,	273969.87137999997
org.eclipse.uml2.uml.internal.impl...	Synch	store,branch,	273969.87137999997
org.eclipse.uml2.uml.internal.impl...	Asynch	branch,msgsnd(),msgrcv(),	1.7847035959798E8
org.eclipse.uml2.uml.internal.impl...	Synch	store,branch,	14.40279
org.eclipse.uml2.uml.internal.impl...	Synch	store,branch,	14.40279
org.eclipse.uml2.uml.internal.impl...	Asynch	branch,msgsnd(),msgrcv(),	17975.51309
org.eclipse.uml2.uml.internal.impl...	Asynch	branch,msgsnd(),msgrcv(),	11524.31309
Encoding	n/a	n/a	1.8021757616128004E8

Fig. 5. The analysis result of App 4 using ESUML-EA

Also, we analyze the power consumptions of those four applications, implemented with C language to compare with our model-based analysis results. The code-based analysis is performed with EMSIM 2.0. The summary of the power consumption analysis results for those four applications is listed in Table 4.

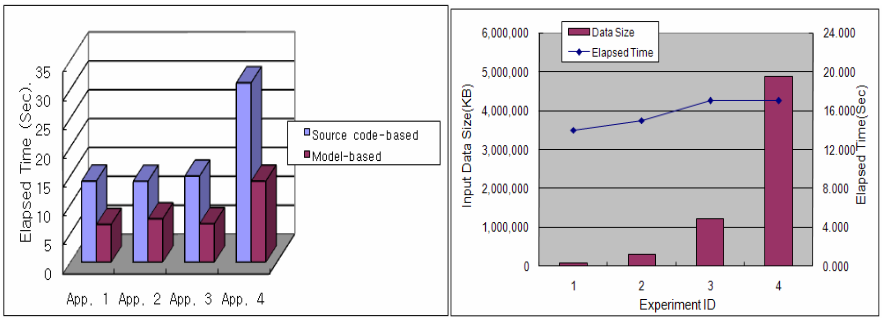
Table 4. Summary of the power consumption analysis results

Target Module	Input Data (Bytes)	Estimated Energy Consumption (nJ)		Deviation (%)
		Code-based	Model-based	
App. 1	80	113203.6	103936.8	8.91
App. 2	288	136982.1	135419.3	1.15
App. 3	262144	11143678.6	10168948.7	9.58
App. 4	31323	190584749.8	183144624.3	3.91

As shown in Table 4, the deviation of estimated energy consumption between two techniques is lower than 10%. From the results, we believe that our proposed model-based power analysis technique has a capable of the applicability to embedded software development with supporting the change of design model and identifying the bottleneck of power consumption to gain the energy-efficient application.

4.3 Elapsed Time for Analysis

A model-based power consumption analysis technique is distinguishable in analysis time compared with code-based analysis technique because model is more abstract than code. To compare the analysis time, we measure the elapsed time of power analysis from EMSIM 2.0 and ESUML-EA which run on the same processor. Fig. 6(a) shows the elapsed time for the target four applications.



(a) (b)

Fig. 6. (a) Comparison of the elapsed time for 4 applications, (b) The elapsed time comparison by data size in App 4

As shown in Fig. 6(a), the model-based estimation time is reduced to a half of source code-based one. This time reduction is obvious, but on the other hand, this means that our model-based analysis technique is very useful in practical embedded software development. We also measure the difference of elapsed time by changing the size of input data in App 4 using ESUML-EA. The graph of the measurement is shown in Fig. 6(b). From the figure, the elapsed time for the analysis using ESUML-EA did not appear to be a big change. This causes that the elapsed time in ESUML-EA is calculated by just library retrieval time to get EBU energy values and multiply operation time for macro-models and loop iterations.

4.4 Application of Model-Based Analysis Results

ESUML-EA, a model-based power consumption analysis tool can report the analysis results by unit of diagrams or objects which are composed of design model. Therefore, we can identify which object or which function consumes a lot of power from the analysis report. This provides useful information to check whether the power requirement is satisfied in design phase of embedded software. Fig. 7 shows the results of power consumption analysis by the objects for App 4.

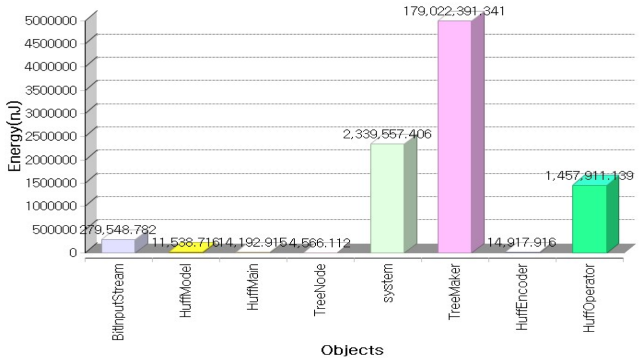


Fig. 7. Power consumption by the objects in App 4

From the Fig. 7, the object TreeMaker, which works to make the encoding tree, consumes a lot of power rather than other objects. Thus, when it is required to reduce the power consumption of the encoding application, we can investigate the object TreeMaker at first to gain high payoff.

5 Conclusions and Future Work

This paper describes a model-based power consumption analysis technique to develop energy-efficient embedded software. To analyze energy consumption of embedded software, our technique provides systematic process which consists of the functions of CFG generation from UML models, EBU detection from the CFG, and energy consumption calculation. Also we provide energy library which contains the energy value for every EBU.

The existing techniques of model-based power consumption analysis require exclusive models for just energy consumption analysis. However, our technique can analyze the power consumption without the development of specific analysis models in software development process because of using UML model. To support the automation of our technique, we developed ESUML-EA tool for UML-based power consumption analysis. We obtained the experimental results that our proposed technique can provide the accuracy of power consumption analysis within 10% deviation and the reduction of analysis time to a half with comparing of code-based analysis technique.

We believe that our proposed technique gives the benefits of early consideration on the power consumption, fast analysis time, and seamless process support in energy-efficient embedded software development. Future work following the current results is on the technique of power consumption analysis for MPSoC (Multi-Processor System on Chip) architecture.

References

1. Spies, P., Babel, P.: Wireless Energy Transmission System for Low-Power Device. In: 17th IEEE Sensors Conference, pp. 33–36. IEEE Press, Italy (2008)
2. Fei, Y., Ravi, S., Raghunathan, A., Jha, N.K.: Energy-Optimizing Source Code Transformations for Operating System-Driven Embedded Software. ACM Transactions on Embedded Computing Systems 7(1), 1–26 (2007)
3. Tiwari, V., Malik, S., Wolfe, A.: Power analysis of embedded software: A first step towards software power minimization. IEEE Transactions on VLSI systems 2(4), 437–445 (1994)
4. Lee, M.T., Tiwari, V., Malik, S., Fujita, M.: Power Analysis and Minimization Techniques for Embedded DSP Software. IEEE Transactions on VLSI Systems 5(1), 123–135 (1997)
5. Klass, B., Thomas, D.E., Schmit, H., Nagle, D.F.: Modeling Inter-instruction energy effects in a digital signal processor. In: 25th International Symposium on Computer Architecture, Spain (1998)
6. Chang, N., Kim, K.H., Lee, H.G.: Cycle-Accurate Energy Consumption Measurement and Analysis: Case Study of ARM7TDMI. In: International Symposium on Low Power Electronics and Design, pp. 185–190. ACM Press, Italy (2000)
7. Sinha, A., Chandrakasan, A.P.: JouleTrack – A Web based Tool for Software Energy Profiling. In: 38th IEEE Conference on Design Automation, pp. 220–225. ACM Press, Las Vegas (2001)
8. Qu, G., Kawabe, N., Usami, K., Potkonjak, M.: Code Coverage-Based Power Estimation Techniques for Microprocessors. Journal of Circuits, Systems, and Computers 11(5), 1–18 (2002)
9. Tan, T.K., Raghunathan, A., Lakshminarayana, G., Jha, N.K.: High-Level Energy Macromodeling of Embedded Software. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 21(9), 605–610 (2003)
10. Tan, T.K., Raghunathan, A., Jha, N.K.: Software Architectural Transformation: A New Approach to Low Energy Embedded Software. In: The Design, Automation and Test in Europe Conference and Exhibition, pp. 1046–1051. IEEE Press, Los Alamitos (2003)
11. Yue, X., Xuehai, Z., Xi, L., Yuchang, G.: OOEM: Object-Oriented Energy Model for Embedded Software Reuse. In: IEEE International Conference on Information Reuse and Integration, pp. 551–558. IEEE Computer Society Press, Los Alamitos (2003)

12. Jun, H., Xuandong, L., Guoliang, Z., Chenghua, W.: Modeling and Analysis of Power Consumption for Component-Based Embedded Software. In: Zhou, X., Sokolsky, O., Yan, L., Jung, E.-S., Shao, Z., Mu, Y., Lee, D.C., Kim, D.Y., Jeong, Y.-S., Xu, C.-Z. (eds.) EUC Workshops 2006. LNCS, vol. 4097, pp. 795–804. Springer, Heidelberg (2006)
13. OMG, Unified Modeling Language: Superstructure. V2.1.2 (formal 2007-11-02), Object Management Group (2007)
14. Dhouib, S., Senn, E., Diguet, J.P., Laurent, J., Blouin, D.: Model Driven High-level Power Estimation of Embedded Operating Systems Communication Service. In: International Conference on Embedded Software and Systems, pp. 475–481. IEEE Computer Society Press, Los Alamitos (2009)
15. Kim, J.P., Kim, D.H., Hong, J.E.: Estimating Power Consumption of Mobile Embedded Software based on Behavioral Model. In: IEEE International Conference on Consumer Electronics, pp. 106–107. IEEE Computer Society Press, Los Alamitos (2010)
16. Selic, B.: The Pragmatics of Model-Driven Development. IEEE Software, 19–25 IEEE Press (2003)
17. PTI, Object Action Language Manual. Doc. Version 1.4, Project Technology Inc. (2004)
18. Garousi, V., Briand, L.C., Labiche, Y.: Control Flow Analysis of UML 2.0 Sequence Diagrams. In: Hartman, A., Kreische, D. (eds.) ECMDA-FA 2005. LNCS, vol. 3748, pp. 160–174. Springer, Heidelberg (2005)
19. Bammie, J.R.: Software Performance Estimation Strategies in a System-level Design Tool. In: International Conference on Hardware Software Codesign, pp. 82–86. ACM Press, New York (2000)
20. Tan, T.K., Raghunathan, A., Jha, N.K.: EMSIM: An Energy Simulation Framework for an Embedded Operating System. In: International Symposium of Circuits and Systems, pp. 464–467. IEEE Press, Los Alamitos (2002)
21. ESUML Project (September 2010), `http://selab.cbnu.ac.kr/projects/esuml/index.html`
22. Kim, D.H., Hong, J.E.: Energy Component Library for Power Consumption Analysis of Embedded Software. Journal of KIPS 16-D(6), 871–880 (2009)
23. Michek, D.E.: Developing Image Processing Algorithm with System Generator. In: Broadcast Solution Guide, pp. 4–6 (2005)
24. Huang, H.C., Wu, J.L.: Novel real-time software-based video coding algorithms. IEEE Transactions on Consumer Electronics 39(3), 570–580 (1993)

Advice Complexity and Barely Random Algorithms[⋆]

Dennis Komm and Richard Královič

Department of Computer Science, ETH Zurich, Switzerland
{dennis.komm,richard.kralovic}@inf.ethz.ch

Abstract. Recently, a new measurement – the *advice complexity* – was introduced for measuring the information content of online problems. The aim is to measure the bitwise information that online algorithms lack, causing them to perform worse than offline algorithms. Among a large number of problems, a well-known scheduling problem, *job shop scheduling with unit length tasks*, and the *paging* problem were analyzed within this model. We observe some connections between advice complexity and randomization. Our special focus goes to barely random algorithms, i. e., randomized algorithms that use only a constant number of random bits, regardless of the input size. We adapt the results on advice complexity to obtain efficient barely random algorithms for both the job shop scheduling and the paging problem.

Furthermore, so far, it has not been investigated for job shop scheduling how good an online algorithm may perform when only using a very small (e. g., constant) number of advice bits. In this paper, we answer this question by giving both lower and upper bounds, and also improve the best known upper bound for optimal algorithms.

1 Introduction

In classical algorithmics, one is interested in designing fast algorithms that create high-quality solutions for a large set of instances of specific problems. Moreover, in many practical applications, another challenge arises for the algorithm designer: Often, not the whole input is known at once, but it arrives piecewise in consecutive time steps. After every such time step, a piece of output has to be created which must not be changed afterwards, i. e., the algorithm has to compute the output without knowing the whole input. We call such situations *online scenarios* and the according strategies to cope with them *online algorithms*. We do not give a detailed introduction, but point the reader to the standard literature, e. g., [4,9].

Classically, the output quality of an online algorithm is measured by the *competitive ratio* [4,9], i. e., the quotient of the cost of the solution the online algorithm computes for a particular problem instance and the cost of an optimal (offline) solution for this instance.

[⋆] This work was partially supported by ETH grant TH 18 07-3 and SNF grant 200020-120073.

Here, we are dealing with online algorithms that have access to an additional advice tape thought of as being written by an oracle $\mathtt{O}$ that sees the whole input in advance and has unlimited computational power. The motivation behind this setup is that $\mathtt{O}$ can give some information about the future parts of the input to the algorithm. This allows us to measure the amount of such information that is necessary/sufficient to obtain an online algorithm with a certain competitive ratio, i. e., we can measure how much information about future the online algorithm is really missing. In some sense, we can see this setup as a generalization of randomized online algorithms where the algorithm has access to another tape with random bits written on it. The concept of *online algorithms with advice* was introduced in [6] and since then revised and applied to several online problems in [2,7]. In the following, we use the same notation as in [2].

Definition 1. *An* online algorithm $\mathtt{A}$ with advice *computes the output sequence* $\mathcal{A}^{\phi} = \mathtt{A}^{\phi}(I) = (y_1, \ldots, y_n)$ *such that* y_i *is computed from* $\phi, x_1, \ldots, x_i$*, where* ϕ *is the content of the advice tape, i. e., an infinite binary sequence.* $\mathtt{A}$ *is* c-competitive with advice complexity $s(n)$ *if there exists a constant* α *such that, for every* n *and for each input sequence* I *of length at most* n*, there exists some* ϕ *such that* $\text{cost}(\mathtt{A}^{\phi}(I)) \leq c \cdot \text{cost}(\mathtt{Opt}(I)) + \alpha$ *and at most the first* $s(n)$ *bits of* ϕ *have been accessed during the computation of* $\mathtt{A}^{\phi}(I)$*.*

Although ϕ is infinitely long, $\mathtt{A}$ only uses a finite prefix during its computation. However, the length of this prefix is determined by the actual run of $\mathtt{A}$.

Moreover, in this paper we are dealing also with randomized online algorithms, i. e., online algorithms that are allowed to base some of their calculations on random decisions.

Definition 2. *A randomized online algorithm* $\mathtt{R}$ *computes the output sequence* $\mathcal{A}^{\phi} = \mathtt{R}^{\phi}(I) = (y_1, \ldots, y_n)$ *such that* y_i *is computed from* $\phi, x_1, \ldots, x_i$*, where* ϕ *is the content of the random tape, i. e., an infinite binary sequence where every bit is chosen uniformly at random and independently of all the others. By* $\text{cost}(\mathtt{R}(I))$ *we denote the random variable expressing the cost of the solution computed by* $\mathtt{R}$ *on* I*. Algorithm* $\mathtt{R}$ *is* c*-competitive if there exists a constant* α *such that, for every input sequence* I*,* $\mathrm{E}[\text{cost}(\mathtt{R}(I))] \leq c \cdot \text{cost}(\mathtt{Opt}(I)) + \alpha$*.*

Generating random numbers might be expensive. Hence, we are interested in designing good randomized algorithms that use as few random bits as possible. It is possible to measure the amount of random bits needed by a randomized algorithm as a function of the input length, in a similar way as the time complexity, space complexity, or advice complexity is measured. Randomized algorithms that use only a constant number of random bits, regardless of the input size, are called *barely random algorithms* [4], introduced in [12]. The number of random bits used by these algorithms is asymptotically minimal, hence they can be considered the best algorithms with respect to the amount of randomness used.

It is very simple to observe that, if there is a c-competitive randomized algorithm $\mathtt{R}$ solving some online problem P using $r(n)$ random bits, where n is the length of the input instance, there also exists a c-competitive algorithm with advice A solving P with advice complexity $s(n) = r(n)$. Indeed, it is sufficient to provide, for every input, the best possible choice of random bits as an advice for A, which then simulates R in a straightforward way. This result can be used for propagating the lower bounds on advice complexity to lower bounds on randomized algorithms using a restricted number of random bits:

Observation 1. *Assume that there is no c-competitive algorithm with advice that solves an online problem P with advice complexity $s(n)$. Then there is no c-competitive randomized algorithm that solves P with $r(n) = s(n)$ random bits.*

The opposite direction does not necessarily hold, i. e., it is not known if it is always possible to transform an efficient algorithm with advice into an efficient randomized algorithm. Nevertheless, the proofs used to construct efficient algorithms with advice can sometimes be adapted to the randomized settings as well. In this way, we obtain some interesting results about barely random algorithms.

1.1 Contribution and Organization of the Paper

In this paper, we deal with two online problems introduced below, the *job shop scheduling* (Jss) problem and the *paging* (Paging) problem. For both, we present efficient barely random algorithms in Section 2, which are the first barely random algorithms known for these problems. The algorithm for Paging, obtained by adapting the results on the advice complexity of Paging, has a competitive ratio asymptotically equal to the best possible competitive ratio of any randomized algorithm. The presented algorithm for Jss can reach a competitive ratio of $1 + \varepsilon$ for any fixed constant $\varepsilon > 0$. This is, however, worse than the best known unrestricted randomized algorithm, whose competitive ratio converges to 1 with growing input size. In Section 3, we discuss the advice complexity of Jss. Due to Observation 1, our results presented here imply that our barely random algorithm for Jss is asymptotically optimal.

1.2 Job Shop Scheduling

First, we are dealing with the following problem called *job shop scheduling* or Jss for short (see [2,5,8,9,11] for a more detailed introduction and description). Let there be two so-called *jobs* A and B, each of which consists of m *tasks*. Each task needs to be processed on a specific *machine*. These are identified by their indices $1, 2, \ldots, m$. Processing one task takes exactly 1 time unit and, since both jobs need every machine exactly once, we may represent them as permutations $P_A = (p_1, p_2, \ldots, p_m)$ and $P_B = (q_1, q_2, \ldots, q_m)$, where $p_i, q_j \in \{1, 2, \ldots, m\}$ for every $i, j \in \{1, 2, \ldots, m\}$. The meaning of such a permutation is that the tasks must be performed in the order specified by it and that the k-th job must be finished before one may start with job $k + 1$. If, at one time step, both jobs A and B ask for the same machine, one of them has to be delayed. The costs of a solution are measured as the total time needed by both machines to finish all jobs. The goal is to minimize this time, which we also call the makespan.

In an online scenario, the permutations P_A and P_B arrive successively, i. e., only p_1 and q_1 are known at the beginning and p_{i+1} $[q_{j+1}]$ is revealed after p_i $[q_j]$ has been processed.

We use the following graphical representation, which was introduced in [5]. Consider an $(m \times m)$-grid where we label the x-axis with P_A and the y-axis with P_B. The cell (p_i, q_j) models that, in the corresponding time step, A processes a task on machine p_i while B processes a task on q_j. A feasible schedule for the induced instance of JSS is a path that starts at the upper left vertex of the grid and leads to the bottom right vertex. It may use diagonal edges whenever $p_i \neq q_j$. However, if $p_i = q_j$, both A and B ask for the same machine at the same time and therefore, one of them has to be delayed. In this case, we say that A and B collide and call the corresponding cells in the grid *obstacles* (see Fig. 1(a)). If the algorithm has to delay a job, we say that it *hits an obstacle* and may therefore not make a diagonal move, but either a horizontal or a vertical one. In the first case, B gets delayed, in the second case, A gets delayed.

Observation 2. *The following facts are immediate [2,9].*

(i) Since P_A and P_B are permutations, there is exactly one obstacle per row and exactly one obstacle per column for every instance.
(ii) There are exactly m obstacles overall for any instance.
(iii) Every optimal solution has cost of at least m and therefore every online algorithm is 2-competitive or better.
(iv) Every feasible solution makes exactly as many horizontal moves as it makes vertical ones. We call the number of horizontal [vertical] moves the delay *of the solution.*
(v) The cost of a solution is equal to m plus the delay of the solution.
(vi) Hitting an obstacle causes additional costs of at most 1 (in certain situations even none) since one diagonal move can be simulated by exactly one vertical and one horizontal move.

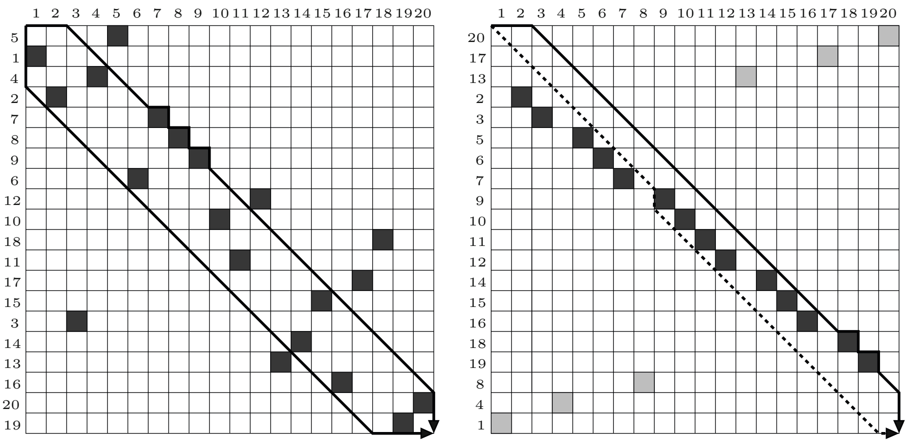
Furthermore, in [8] it was shown that, for every instance, there always exists a solution with costs of at most $m + \lceil \sqrt{m} \rceil$.

Let diag_0 be the main diagonal (from $(1, 1)$ to (m, m)) in the grid. The diagonal that has a distance of i from diag_0 and lies below [above] it, is called diag_{-i} $[\text{diag}_i]$. Similar to [8], for any odd d, we consider a certain set of strategies

$$\mathcal{D}_d = \left\{ D_i \;\middle|\; i \in \left\{ -\frac{d-1}{2}, \ldots, \frac{d-1}{2} \right\} \right\}$$

where D_j is the strategy to move to the starting point of diag_j with j steps, to follow it when possible, and to avoid any obstacle by making a horizontal step directly followed by a vertical one (thus returning to diag_j).

Please note that it is crucial for our analysis that the algorithm *returns* to the diagonal even though there might be situations where it is an advantage not to take the vertical step after the horizontal one.



(a) The strategies D_{-3} and D_2
(b) A hard instance for $\mathcal{D}_5$

Fig. 1. An example with two jobs each of size 20. Obstacles are marked by filled cells.

1.3 Paging

The second problem we focus on is among the most-studied online problems with great practical relevance. The *paging problem*, Paging for short, is motivated by the following circumstance: the performance of today's computers is limited by the fact that the physical memory is a lot slower than the CPU (this fact is known as the von Neumann bottleneck). Hence, the concept of a very fast (and therefore more expensive and consequently smaller) *cache* is used to store as much of the content of the physical memory as possible. We aim at maximizing the communication between the CPU and the cache and thereby minimizing the more costly communication between the CPU and the physical memory. A similar situation occurs between the physical memory and the much slower hard disc. Formally, we deal with the following problem.

Definition 3 (Paging Problem). *The input is a sequence of integers representing requests to logical pages $I = (x_1, \ldots, x_n)$, $x_i > 0$. An online algorithm* $\mathtt{A}$ *maintains a buffer (content of the cache) $B = \{b_1, \ldots, b_K\}$ of K integers, where K is a fixed constant known to* $\mathtt{A}$*. Before processing the first request, the buffer gets initialized as $B = \{1, \ldots, K\}$. Upon receiving a request x_i, if $x_i \in B$, then* $\mathtt{A}$ *creates the partial output $y_i = 0$. If $x_i \notin B$, then a* page fault *occurs, and the algorithm has to find some* victim *b_j, i. e., $B := (B \setminus \{b_j\}) \cup \{x_i\}$, and $y_i = b_j$. The cost of the solution $\mathcal{A} = \mathtt{A}(I)$ is the number of page faults, i. e.,* $\mathrm{cost}(\mathcal{A}) = |\{y_i \mid y_i > 0\}|$.

A more complete description of Paging can be found in [4]. Furthermore, in [2,6], the problem was examined within the scope of advice complexity.

Throughout this paper, by $\log x$ we denote the logarithm of x with base 2.

2 Barely Random Algorithms

In what follows, we construct barely random algorithms for both problems.

2.1 Job Shop Scheduling

We consider the class $\mathcal{D}_d$ of diagonal strategies as introduced in Section 1 for some odd constant $d > 1$. Consider a barely random algorithm $\mathtt{R}_d$ that chooses a strategy from this class uniformly at random, using at most $\log d$ random bits to do so.

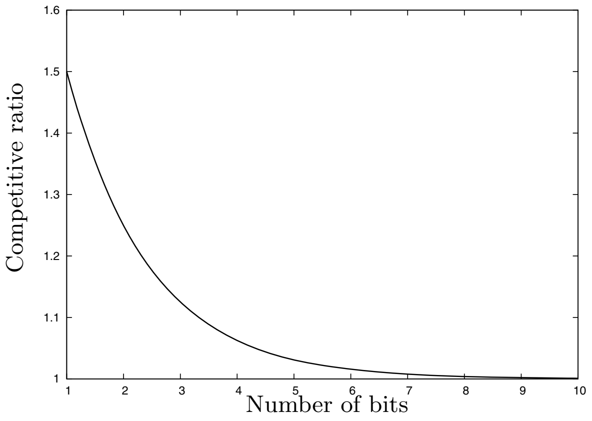


Fig. 2. The competitive ratio of $\mathtt{R}_d$ depending on $\log d$ for m tending to infinity

Theorem 1. *The algorithm* $\mathtt{R}_d$ *achieves an expected competitive ratio that tends to* $1 + 1/d$ *for a growing number of tasks.*

Proof. For every odd d, consider the following random variables $X_1, X_2, X, Y\colon \mathcal{D}_d \to \mathbb{R}$, where $X_1(D_i)$ is the delay caused by the initial horizontal [vertical] steps made by the strategy D_i, $X_2(D_i)$ is the delay caused by D_i hitting obstacles, $X(D_i) = X_1(D_i) + X_2(D_i)$ is D_i's overall delay, and $Y(D_i) = m + X(D_i)$ is D_i's overall cost. Recall that D_{-j} and D_j make the same amount of vertical [horizontal] moves at the beginning. Since there are exactly m obstacles in total for every instance, we immediately get

$$\mathrm{E}[X_2] = \frac{1}{d}\left(X_2(D_0) + 2\sum_{i=1}^{(d-1)/2} X_2(D_i)\right) \le \frac{m}{d}$$

and since $X_1(D_0) = 0$, we get

$$\mathrm{E}[X_1] = \frac{1}{d}\left(X_1(D_0) + 2\sum_{i=1}^{(d-1)/2} X_1(D_i)\right) = \frac{2}{d}\sum_{i=1}^{(d-1)/2} i = \frac{d^2-1}{4d}.$$

Due to the linearity of expectation, it follows that

$$\mathrm{E}[Y] = m + \mathrm{E}[X] = m + \mathrm{E}[X_2] + \mathrm{E}[X_1] \leq m + \frac{m}{d} + \frac{d^2-1}{4d}.$$

Therefore, the expected competitive ratio of $\mathtt{R}_d$ is at most

$$\frac{\frac{(d+1)m}{d} + \frac{d^2-1}{4d}}{m} = 1 + \frac{1}{d} + \frac{d^2-1}{4dm}$$

which, for increasing m, tends to $1 + 1/d$ as claimed. □

In Section 3 (see Theorem 5), we will prove a lower bound of $1+1/2^{b+3}-\varepsilon$ for any online algorithm with advice that reads b bits. Together with Observation 1, we obtain that, for any $\varepsilon > 0$, no randomized algorithm that uses at most b random bits can obtain a competitive ratio of $1 + 1/2^{b+3} - \varepsilon$. Hence, barely random algorithms for Jss cannot have a competitive ratio that tends to 1 with growing m. This means they perform worse than the randomized algorithms from [8] (using an unrestricted number of random bits), which can reach a competitive ratio that tends to 1 with growing m.

2.2 Paging

Next, we look at Paging and show the existence of a barely random algorithm that achieves a low competitive ratio. It is well known that no deterministic algorithm for Paging can be better than K-competitive, where K is the size of the cache, and that there exists an $\mathcal{O}(\log K)$-competitive randomized algorithm [4]. More precisely, there is a H_k-competitive randomized algorithm for Paging, where $H_k = \sum_{i=1}^{k} 1/i$ is the k-th harmonic number, and this bound is tight [1]. To the best of our knowledge, however, all randomized algorithms for Paging known so far that reach a competitive ratio of $\mathcal{O}(\log K)$ use $\Omega(n)$ random bits for inputs of length n, and no efficient barely random algorithm for Paging is known up to now.

In [2,3], it was shown that there exists an online algorithm with advice $\mathtt{A}$ that reads $\log b$ bits of advice and has a competitive ratio of at most $3\log b + \frac{2(K+1)}{b} + 1$, where K is the buffer size of $\mathtt{A}$. This result can be easily adapted for the randomized case:

Theorem 2. *Consider* Paging *with buffer size K, and let $b < K$ be a power of 2. There exists a barely random algorithm for* Paging *that uses* $\log b$ *random bits, regardless of the input size, and achieves a competitive ratio of*

$$r \leq 3\log b + \frac{2(K+1)}{b} + 1.$$

Proof. The proof is almost identical to the proof of Theorem 5 in [3]. The core idea of this proof is to construct b deterministic algorithms $\mathtt{A}_1, \ldots, \mathtt{A}_b$ such that, for any input instance, the total number of page faults generated by all algorithms together is limited. In particular, the proof of [3, Theorem 5] describes a set of b algorithms such that, for any input instance I, the total number of page faults is bounded by

$$m\frac{b}{2}(3\log b + 1) + m(K+1),$$

where m is a certain parameter depending on I. Furthermore, any algorithm makes at least $m/2$ page faults on I. Hence, selecting one of the b algorithms uniformly at random and running it yields a randomized algorithm with an expected number of page faults $m\left(\frac{K+1}{b} + \frac{3}{2}\log b + \frac{1}{2}\right)$. Thus, the competitive ratio of such a randomized algorithm is at most

$$\frac{m\left(\frac{K+1}{b} + \frac{3}{2}\log b + \frac{1}{2}\right)}{\frac{m}{2}} = \frac{2(K+1)}{b} + 3\log b + 1.$$

Obviously, selecting the algorithm can be done with $\log b$ bits. □

The previous theorem shows that there exists a barely random algorithm for Paging that uses only $\log K$ bits and reaches a competitive ratio of $\mathcal{O}(\log K)$ which is asymptotically equivalent to the best possible randomized algorithm.

3 Advice Complexity of Job Shop Scheduling

Here, we consider the advice complexity of Jss, that is, we give lower and upper bounds on the number of advice bits needed to achieve a certain output quality. Doing so, we improve and generalize some of the results obtained in [2].

3.1 Optimality

We quickly discuss the amount of information needed for an online algorithm to produce an optimal output for Jss. Due to space limitations, we do not prove the following theorem, but point the reader to [10].

Theorem 3. *There exists an optimal online algorithm* A *with advice complexity* $s(m) = 2\lceil\sqrt{m}\,\rceil - \frac{1}{4}\log m$ *for any instance of* Jss.

3.2 Competitive Ratio

Consider the proof of Theorem 1: actually, this is a probabilistic proof for the competitive ratio of an algorithm with advice $\mathtt{A}_d$ that uses $\log d$ bits of advice to choose a strategy from $\mathcal{D}_d$, instead of random bits. In fact, as shown in Observation 1, the best random string for the input can be provided as the advice. This immediately implies the following theorem.

Theorem 4. *There exists an online algorithm with advice* $\mathtt{A}_d$ *that reads* $\log d$ *advice bits and has a competitive ratio tending to* $1 + 1/d$. □

In [10], we showed a slightly better bound which is left out here due to space restrictions. It is not difficult to see that the above bound is almost tight for every d. To show this, we give a construction that blocks all diagonals the algorithm chooses from. Following any of the blocked diagonals causes the algorithm to have costs of at least $m + m/d$, whereas an optimal solution has costs of exactly $m + 1$.

Lemma 1. *For any d, the competitive ratio of the algorithm $\mathtt{A}_d$ is at least $1+1/d$.*

Proof. Let m be divisible by d and let m/d be even. We now describe how to sufficiently delay every possible diagonal strategy. Suppose we want to make sure that every strategy has a delay of at least l (where l is divisible by 2). At first, we place l obstacles in the center of the main diagonal, i. e., in the cells $(m/2 - l/2 + 1, m/2 - l/2 + 1)$ to $(m/2 + l/2, m/2 + l/2)$. For now, let us focus on the cells which are in the bottom-right quadrant of the $(m \times m)$-grid. For each $i \in \{1, 2, \dots, (d-1)/2\}$, we create one block of obstacles. The block corresponding to i consists of $l - i$ obstacles. All of these obstacles are put on the i-th diagonal above the main one, in consecutive rows, just below the rows used by the block $i - 1$. In particular, the obstacles of block 1 are located on

$$\left(\frac{m+l}{2}+1, \frac{m+l}{2}+2\right), \dots, \left(\frac{m+l}{2}+l-1, \frac{m+l}{2}+l\right),$$

the obstacles of block 2 are located on

$$\left(\frac{m+l}{2}+l, \frac{m+l}{2}+l+2\right), \dots, \left(\frac{m+l}{2}+2l-3, \frac{m+l}{2}+2l-1\right),$$

etc. Hence, we need to use $l - i$ rows and $l - i + 1$ columns to build the block i (the first column of the block is empty, since block i is on a different diagonal than block $i - 1$). To be able to successfully build all of the blocks, we need at least

$$\frac{l}{2} + 1 + (l-1) + 1 + (l-2) + 1 + \dots + \left(l - \frac{d-1}{2}\right)$$

columns (clearly, if there are enough columns available, there are enough rows as well). Since we have exactly $m/2$ columns, we have to ensure that

$$\frac{l}{2} + \sum_{i=1}^{(d-1)/2} 1 + l - i \le \frac{m}{2}$$

$$\iff \quad \frac{l}{2} + \frac{d-1}{2}(1+l) - \frac{d^2-1}{8} \le \frac{m}{2} \quad \iff \quad l \le \frac{m + d^2 - d}{d}.$$

The same construction can be performed in the top-left quadrant in a symmetric way. In every block, there is one free column. It remains to use the rows not used by any block (nor the obstacles in the main diagonal) to put a single obstacle to every such free column. To do so, we use the top-right and bottom-left quadrant. It is straightforward to observe that this is always possible, even without using any diagonal neighboring the main one.

An example of this construction for $m = 20$ and $\mathcal{D}_5$ shown in Fig. 1(b). It is immediately clear that any optimal solution has costs of exactly $m + 1$. An optimal solution follows the main diagonal until the first obstacle is hit. Afterwards, the solution makes one vertical step and follows the first diagonal below the main one (i. e., diag_{-1}).

We can therefore guarantee that the competitive ratio of $\mathtt{A}_d$ on this instance is at least

$$\frac{m+\frac{m+d^2-d}{d}}{m+1}=\frac{(m+1)(1+\frac{1}{d})-1-\frac{1}{d}+d-1}{m+1}=1+\frac{1}{d}+\frac{d^2-2d-1}{d(m+1)}\geq 1+\frac{1}{d}.$$

□

Up to this point, we have shown that, with a small constant number of advice bits, it is possible to perform very well. In [2], it was shown that at least

$$\left\lfloor\frac{\sqrt{16m+9}-11}{8}\right\rfloor$$

bits are needed for any online algorithm with advice to be optimal.

A naturally arising question is whether we can be $(1+\Theta(1/m))$-competitive with reading a constant number of advice bits, i. e., if it suffices to use a constant number of bits to get arbitrarily close to the optimal solution. In the following, we disprove this.

Theorem 5. *For any $\varepsilon > 0$, any online algorithm with advice that reads b bits of advice cannot be better than $(1+1/2^{b+3}-\varepsilon)$-competitive.*

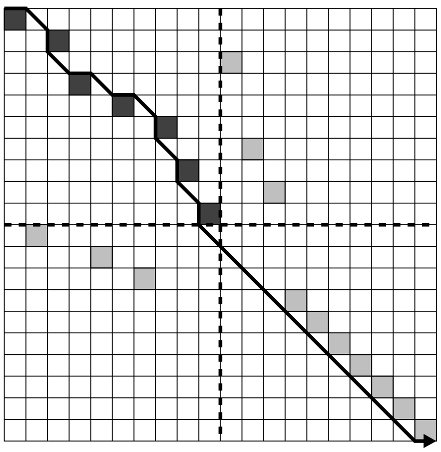
Proof. In [9], it was shown that any deterministic online algorithm $\mathtt{A}$ for Jss has a competitive ratio of at least 9/8. There always exists an adversary that can make sure that every second move $\mathtt{A}$ makes in the upper left quadrant is not a diagonal move: The intuitive idea is that, after every diagonal move of $\mathtt{A}$, the algorithm reaches a column and a row in which the adversary has not yet placed an obstacle. It follows that, if $\mathtt{A}$ leaves this quadrant, it already has made at least $m/4$ non-diagonal moves and therefore has a delay of at least $m/8$. This idea is shown in Fig. 3(a). Recall that we already know that there always exists an optimal solution with costs of at most $m+\lceil\sqrt{m}\,\rceil$ (see [8]).

Let m be a multiple of 2^{b+3}. Suppose we are now dealing with any algorithm $\mathtt{A}_b$ that reads b bits of advice while processing an input of size m. We impose another virtual grid on the $(m\times m)$-grid, where each virtual cell consists of $m/2^b$ original cells. Let us now consider the 2^b virtual cells on the main diagonal and label them $S_1, S_2, \ldots, S_{2^b}$.

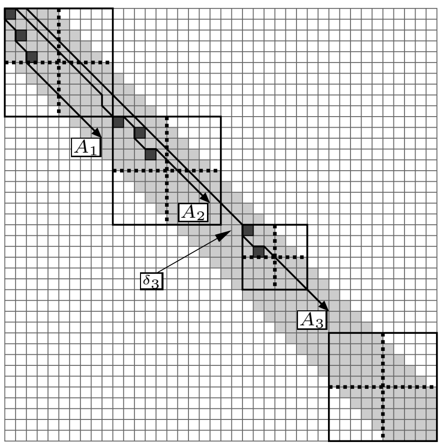
We call all original cells that have a deviation of less than $m/2^{b+1}$ from the main diagonal the *active zone* (marked grey in Fig. 3(b)). Any strategy that leaves this zone at any point makes at least $m/2^{b+1}$ horizontal [vertical] moves. We conclude

$$\frac{m+\frac{m}{2^{b+1}}}{m+\lceil\sqrt{m}\,\rceil}\geq\left(1+\frac{1}{2^{b+1}}\right)\frac{1}{1+\frac{\lceil\sqrt{m}\,\rceil}{m}}\geq 1+\frac{1}{2^{b+1}}-\varepsilon$$

for sufficiently large m and we may therefore assume that no strategy leaves the active zone. Observe that we may think of $\mathtt{A}_b$ as 2^b deterministic algorithms we have to deal with. We may label them $A_1, A_2, \ldots, A_{2^b}$ by sorting the corresponding advice in canonical order and assign each deterministic algorithm



(a) A hard instance as used in [9]



(b) An instance as used in the proof of Theorem 5

Fig. 3. An example of how to place the obstacles in such a way that any deterministic algorithm cannot make two consecutive diagonal moves in the first quadrant as presented in [9] and a hard instance for $\mathtt{A}_b$ that uses this construction 2^b times

A_i to exactly one virtual cell S_i. Algorithm A_1 enters S_1 right after receiving the first request. We observe that, since S_1 has size $(m/2^b \times m/2^b)$, S_1's first quadrant has size $(m/2^{b+1} \times m/2^{b+1})$. Thus, by applying the scheme used in [9], we can guarantee that A_1 makes at least 2^{b+2} non-diagonal moves within this first quadrant, and this causes a delay of at least $m/2^{b+3}$. Afterwards, we do not consider A_1 anymore. Since no algorithm leaves the active zone, A_i always enters S_i's first quadrant at some point. If A_i enters S_i at its upper left corner, we can apply the same argumentation as for S_1. Therefore, let δ_i denote the deviation of A_i from the main diagonal where $0 < \delta_i \leq m/2^b$. Without loss of generality, we assume that A_i is above D_0 when it enters S_i's first quadrant. It is then clear that A_i has already made at least δ_i horizontal moves (Fig. 3(b) depicts this situation for $i = 3$ and $\delta_3 = 2$). We now shrink S_i to S_i' such that A_i enters S_i' at its left corner and that S_i' has a size of $(m/2^b - \delta_i, m/2^b - \delta_i)$. We can assure that A_i makes at least

$$\left\lfloor \left\lfloor \frac{\frac{m}{2^b} - \delta_i}{2} \right\rfloor /2 \right\rfloor \geq \left\lfloor \frac{\frac{m}{2^b} - \delta_i - 1}{4} \right\rfloor \geq \frac{\frac{m}{2^b} - \delta_i - 5}{4}$$

non-diagonal moves in the first quadrant of S_i and the overall delay is therefore at least

$$\frac{\frac{m}{2^b} - \delta_i - 5}{8} + \delta_i = \frac{m}{2^{b+3}} + \frac{7\delta_i - 5}{8} \geq \frac{m}{2^{b+3}}.$$

It is straightforward to check that a similar strategy is possible if A_i is below D_0. Thus, we can conclude that $\mathtt{A}_b$ cannot achieve a competitive ratio better than

$$\frac{m + \frac{m}{2^{b+3}}}{m + \lceil \sqrt{m}\, \rceil}$$

which is greater than $1 + \frac{1}{2^{b+3}} - \varepsilon$ for sufficiently large m, finishing the proof. □

Using this result, an easy calculation shows that the bound from Theorem 1 is tight up to a small multiplicative constant tending to 1 for an increasing b. See [10] for the details omitted here due to space restrictions.

References

1. Achlioptas, D., Chrobak, M., Noga, J.: Competitive analysis of randomized paging algorithms. Theoretical Computer Science 234(1–2), 203–218 (2000)
2. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: On the advice complexity of online problems. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 331–340. Springer, Heidelberg (2009)
3. Böckenhauer, H.-J., Komm, D., Královič, R., Královič, R., Mömke, T.: Online Algorithms with Advice. Technical Report 614. ETH Zurich, Department of Computer Science (2009)
4. Borodin, A., El-Yaniv, R.: Online computation and competitive analysis. Cambridge University Press, New York (1998)
5. Brucker, P.: An efficient algorithm for the job-shop problem with two jobs. Computing 40(4), 353–359 (1988)
6. Dobrev, S., Královič, R., Pardubská, D.: How much information about the future is needed? In: 34th Conf. on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008), pp. 247–258 (2008)
7. Emek, Y., Fraigniaud, P., Korman, A., Rosén, A.: Online computation with advice. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 427–438. Springer, Heidelberg (2009)
8. Hromkovič, J., Mömke, T., Steinhöfel, K., Widmayer, P.: Job shop scheduling with unit length tasks: bounds and algorithms. Algorithmic Operations Research 2(1), 1–14 (2007)
9. Hromkovič, J.: Design and Analysis of Randomized Algorithms: Introduction to Design Paradigms. Springer, New York (2006)
10. Komm, D., Královič, R.: Advice Complexity and Barely Random Algorithms. Technical Report 684. ETH Zurich, Department of Computer Science (2010)
11. Mömke, T.: On the power of randomization for job shop scheduling with k-units length tasks. RAIRO Theoretical Informatics and Applications 43, 189–207 (2009)
12. Reingold, N., Westbrook, J., Sleator, D.: Randomized competitive algorithms for the list update problem. *Algorithmica* 11(1), 15–32 (1994)

Alternative Parameterizations for Cluster Editing[⋆]

Christian Komusiewicz and Johannes Uhlmann

Institut für Informatik, Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{c.komus,johannes.uhlmann}@uni-jena.de

Abstract. Given an undirected graph G and a nonnegative integer k, the NP-hard Cluster Editing problem asks whether G can be transformed into a disjoint union of cliques by applying at most k edge modifications. In the field of parameterized algorithmics, Cluster Editing has almost exclusively been studied parameterized by the solution size k. Contrastingly, in many real-world instances it can be observed that the parameter k is not really small. This observation motivates our investigation of parameterizations of Cluster Editing different from the solution size k. Our results are as follows. Cluster Editing is fixed-parameter tractable with respect to the parameter "size of a minimum cluster vertex deletion set of G", a typically much smaller parameter than k. Cluster Editing remains NP-hard on graphs with maximum degree six. A restricted but practically relevant version of Cluster Editing is fixed-parameter tractable with respect to the combined parameter "number of clusters in the target graph" and "maximum number of modified edges incident to any vertex in G". Many of our results also transfer to the NP-hard Cluster Deletion problem, where only edge deletions are allowed.

1 Introduction

The NP-hard Cluster Editing problem is among the best-studied parameterized problems. It is usually defined as follows:

Input: An undirected graph $G = (V, E)$ and an integer $k \geq 0$.
Question: Can G be transformed into a cluster graph by applying at most k edge modifications?

Herein, an *edge modification* is either the deletion or insertion of an edge and a *cluster graph* is a graph where every connected component is a clique. The cliques of a cluster graph are referred to as *clusters*. Cluster Deletion is defined analogously except that only edge deletions are allowed.

So far, the proposed fixed-parameter algorithms for Cluster Editing almost exclusively examine the parameter solution size k. While several algorithmic

⋆ Supported by the DFG, research project PABI NI369/7.

improvements have led to impressive theoretical results, it has been observed that the parameter k is often not really small for real-world instances [3]. Still, the fixed-parameter algorithms can solve many of these instances [3], which raises the question whether there are "hidden parameters" that are implicitly exploited by these algorithms. In the spirit of multivariate algorithmics (cf. [15]), this work aims at identifying promising new parameterizations for Cluster Editing that help to separate easy from hard instances.

Related Work. The NP-hardness of Cluster Editing has been shown several times [14, 17, 1]. The problem remains NP-hard even when the solution may contain at most two clusters [17]. The parameterized complexity of Cluster Editing with respect to the parameter k has been extensively studied. After a series of improvements [10, 9, 16, 11, 2, 4], the currently fastest fixed-parameter algorithm for this parameter has running time $O(1.82^k + n^3)$ [2] and the currently smallest problem kernel contains at most $2k$ vertices [4]. Several experimental studies demonstrate that fixed-parameter algorithms can be applied to solve real-world instances of Cluster Editing [6, 3]. Further theoretical studies have dealt with the parameterized complexity of different generalizations of Cluster Editing [5, 12, 8] for example by replacing the clique requirement in the cluster graph with other models for dense graphs. The problem to transform a graph into a cluster graph by a minimum number of vertex deletions is called Cluster Vertex Deletion(CVD). Hüffner et al. [13] presented an $O(2^k k^9 + nm)$-time iterative compression algorithm for CVD.

Our Results. Motivated by the observation that the parameter k is often very large in practice and subsequent calls for "better parameterizations" [7], we consider new parameters for Cluster Editing. Answering an open question by Dehne [7], we show that Cluster Editing is fixed-parameter tractable with respect to the parameter "cluster vertex deletion number" c of the input graph G. Moreover, we consider the parameter t denoting the "maximal number of modified edges incident to any vertex". First, we show that Cluster Editing is NP-hard for maximum degree-six graphs. Since in an optimal solution the number of incident edge modifications of a vertex is bounded by its degree, there is no hope for fixed-parameter tractability with respect to t. However, we can show that a restricted version of Cluster Editing is fixed-parameter tractable when combining t with the parameter d denoting the number of cliques in the final cluster graph. Our methods include enumerative approaches, matching techniques, and problem kernelization.

Due to lack of space, several proofs are deferred to a full version of the paper.

Preliminaries. Given a graph $G = (V, E)$, we use $V(G)$ to denote the vertex set of G and $E(G)$ to denote the edge set of G. Let $n := |V|$ and $m := |E|$. The (open) neighborhood $N(v)$ of a vertex v is the set of vertices adjacent to v, and the closed neighborhood is $N[v] := N(v) \cup \{v\}$. For a vertex set W let $E_W := \{\{v, w\} \mid \{v, w\} \subseteq W\}$ denote the set of all size-two subsets of W. We use $G[V']$ to denote the subgraph of G induced by $V' \subseteq V$, that is, $G[V'] := (V', E_{V'} \cap E)$.

Moreover, $G - v := G[V \setminus \{v\}]$ for a vertex $v \in V$ and $G - e := (V, E \setminus \{e\})$. Let $E \Delta F := (E \setminus F) \cup (F \setminus E)$ denote the symmetric difference of two sets E and F. The *edit distance* between two graphs G_1 and G_2 on the same vertex set is $|E(G_1) \Delta E(G_2)|$. Given three graphs G_1, G_2, and G_3, we say that G_2 is *closer* to G_1 than G_3 if the edit distance between G_1 and G_2 is strictly smaller than the edit distance between G_1 and G_3. For a graph $G = (V, E)$ and a set $S \subseteq E_V$ let $G \Delta S := (V, E \Delta S)$ denote the graph that results by modifying G according to S. A set of pairwise adjacent vertices is called *clique*.

2 Cluster Vertex Deletion Number

In this section, we present fixed-parameter algorithms for Cluster Editing (CE) and Cluster Deletion (CD) parameterized by the size of a minimum-cardinality vertex set Y such that removing Y from the input graph G results in a cluster graph. In the following, we will refer to this parameter as cluster vertex deletion number c of G. Note that c is bounded from above by the size k of a minimum-cardinality edge-modification set: deleting for each edge modification one of the two vertices (arbitrarily chosen) clearly results in a cluster graph. Our algorithms solve the optimization version of CE and CD, that is, they find a minimum-cardinality edge modification or edge deletion set, respectively. Both algorithms make use of the following observation for cliques that are large in comparison to the size of their neighborhood. These cliques are preserved to large extent by any optimal solution for CE or CD.

Lemma 1. *Let K denote a clique in G of size at least $2 \cdot |N_G(K)|$. Then, for every optimal edge modification set (optimal edge deletion set) S, the graph $G \Delta S$, contains a cluster K' with*

$$|K \cap K'| \geq |K| - 2|N_G(K)|.$$

Proof. We show the lemma for the case of an optimal edge modification set S. Let $K'_1, \ldots, K'_\ell$ denote the clusters in $G \Delta S$ with $K'_i \cap K \neq \emptyset$. Furthermore, define $B_i := K'_i \cap K$ and observe that $K = \bigcup_{i=1}^{\ell} B_i$. Note that in the case that $|K| \leq 2|N_G(K)| + 1$ the lemma trivially holds since $|B_1| \geq 1$ and $|K| - 2|N_G(K)| \leq 1$.

Assume towards a contradiction that all B_i's contain less than $|K| - 2|N_G(K)|$ vertices. This implies that—in order to separate the B_i's from each other—the solution contains at least

$$1/2 \sum_{i=1}^{\ell} |B_i|(|K| - |B_i|) > 1/2 \sum_{i=1}^{\ell} (|B_i| \cdot 2|N_G(K)|) = |N_G(K)| \cdot |K|$$

edge deletions. Hence, one obtains a cluster graph that is closer to G by deleting in $G \Delta S$ all edges between K and $N(K)$ (at most $|N(K)| \cdot |K|$) and undoing all edge deletions between vertices in K (at least $|N(K)| \cdot |K| + 1$); a contradiction to the fact that S is optimal. It is easy to verify that all steps of the proof hold for an optimal edge deletion set, too. □

Next, we present our fixed-parameter algorithm for Cluster Editing.

2.1 Cluster Editing

Given an input graph G and a size-c cluster vertex deletion set Y of G, a key observation used by our algorithm is that clusters in $G-Y$ that are much larger than Y will not be split by any optimal edge modification set. Recall that an isolated clique of a cluster graph is called a *cluster*.

Lemma 2. *Let Y denote a size-c cluster vertex deletion set and let K denote a cluster in $G-Y$ of size at least $3c+1$. Then, for every optimal edge modification set S, the graph $G \Delta S$ contains a cluster K' with*

1. *$K \subseteq K'$ and*
2. *$K' \subseteq K \cup Y$.*

Proof. Let $K_1, \ldots, K_\ell$ ($\ell \geq 1$) denote the clusters in $G \Delta S$ with $K_i \cap K \neq \emptyset$. Let $B_i := K \cap K_i$ and note that $K = \bigcup B_i$. Without loss of generality, assume that B_1 has maximum size of all B_i's. Since $N_G(K) \subseteq Y$ and by Lemma 1, $|B_1| \geq c+1$. First, we show that $K_1 \setminus B_1 \subseteq Y$. Let $X := K_1 \setminus (B_1 \cup Y)$ and assume towards a contradiction that $X \neq \emptyset$. Since $|B_1| \geq c+1$ one obtains a cluster graph that is closer to G than $G \Delta S$ by making X an isolated clique, which requires at most $|X| \cdot c$ additional edge deletions, however, allows one to undo the edge insertions between B_1 and X which amount to at least $|X| \cdot (c+1)$.

Next, we prove that $\ell = 1$, directly implying the lemma. Assume towards a contradiction that $\ell > 1$. Since $|B_1| > c$ one obtains a cluster graph that is closer to G than $G \Delta S$ by deleting all edges between B_2 and $K_2 \setminus B_2$ (at most $|K_2 \cap Y| \cdot |B_2|$ additional edge deletions), inserting all edges between B_2 and $K_1 \setminus B_1$ (at most $|B_2| \cdot |K_1 \cap Y|$ edge insertions since $K_1 \setminus B_1 \subseteq Y$), and undoing the edge deletions between B_2 and B_1 (at least $|B_2| \cdot |B_1|$). Since $|B_2| \cdot |K_2 \cap Y| + |B_2| \cdot |K_1 \cap Y| \leq |B_2| \cdot c < |B_2| \cdot (c+1) \leq |B_2| \cdot |B_1|$, this is a contradiction to the fact that S is optimal. □

According to Lemma 2, a cluster in $G-Y$ of size at least $3c+1$ will not be split by any optimal edge modification set and also not "merged" with any other clusters of $G-Y$. We refer to these clusters of $G-Y$ as *large clusters*. The basic idea to establish fixed-parameter tractability is as follows (see Alg. 1 for an outline). Given a cluster vertex deletion set Y, in a first step, we guess the partition of Y "induced" by the clusters generated by an optimal edge modification set (Line 5 of Alg. 1). We refer to the sets of such a partition as *fixed subclusters* in the following. Then, in a second step, we guess which of these fixed subclusters will end up in a cluster together with a large cluster (Line 10 of Alg. 1). From Lemma 2, we know that the large clusters cannot be split, and since the subclusters in Y are fixed, the large clusters end up in a final cluster with at most one fixed subcluster. Hence, the "mapping" of the large clusters to the respective fixed subclusters can efficiently be done by computing a maximum weight matching. To this end, Alg. 1 employs the subroutine `SolveLarge` (see Line 13). The remaining instance is solved by the subroutine `SolveSmall`. This subroutine uses data reduction to bound the number of small clusters in $G-Y$ by a function

Algorithm 1. An algorithm to find an optimal solution for Cluster Editing

```
Function CEbyCVD(G)
Input: A graph $G = (V, E)$
Output: Size of an optimal solution $S$ for Cluster Editing
1  $Y$ = SolveCVD (G) ;
2  Let $A_1, \ldots, A_p$ denote the clusters in $G - Y$ of size at most $3d$;
3  Let $B_1, \ldots, B_q$ denote the clusters in $G - Y$ of size at least $3d + 1$;
4  $m_1 = +\infty$;
5  forall partitions $Q_1, \ldots, Q_t$ of $Y$ ($1 \leq t \leq |Y|$) do
6      Add all edges between vertices in $Q_i$, $1 \leq i \leq t$;
7      Delete all edges between $Q_i$ and $Q_j$, $1 \leq i < j \leq t$;
8      Let $c_1$ denote the number of these edge modifications;
9      $m_2 = +\infty$;
10     forall subsets $I \subseteq \{1, \ldots, t\}$ do
11         Delete all edges between $Q_i$ and $A_j$, $i \in I$ and $1 \leq j \leq p$;
12         Let $c_2$ denote the number of these edge deletions;
13         $c_l$ =SolveLarge ($\{Q_i \mid i \in I\}$, $B_1, \ldots, B_p$) ;
14         $c_s$ =SolveSmall ($\{Q_i \mid i \in \{1, \ldots, t\} \setminus I\}$, $A_1, \ldots, A_q$ );
15         $m_2 = \min(m_2, c_2 + c_l + c_s)$;
16     end
17     $m_1 = \min(m_1, c_1 + m_2)$;
18 end
19 return $m_1$;
```

only depending on c, thus yielding a problem kernel for this subproblem. This directly implies fixed-parameter tractability.

Next, we focus on the computation of an optimal solution for the subproblem that has to be solved by `SolveLarge`. Formally, we have to find a solution to the following problem Fixed Clique Cluster Editing: Let $G = (V, E)$ be a graph and let $B \uplus Q$ be a two partition of V such that $G[B]$ and $G[Q]$ are cluster graphs. Furthermore, let $\mathcal{B} = \{B_1, \ldots, B_q\}$ be the set of clusters in $G[B]$ and let $\mathcal{Q} = \{Q_1, \ldots, Q_s\}$ be the set of clusters in $G[Q]$. The task is to find a cluster graph G_c on V such that the following *cluster constraints* are fulfilled:

1. each set in $\mathcal{B} \cup \mathcal{Q}$ is a subset of a cluster of G_c,
2. for every $B_i \in \mathcal{B}$ ($Q_i \in \mathcal{Q}$) the cluster containing B_i (Q_i) does not contain any other set from $\mathcal{B}$ ($\mathcal{Q}$) and at most one set from $\mathcal{Q}$ ($\mathcal{B}$), and
3. among all such cluster graphs G_c has minimum edit distance to G.

This problem can be formulated as a bipartite maximum weight matching problem and, hence, can be solved in polynomial time.

Lemma 3. Fixed Clique Cluster Editing *can be solved in polynomial time.*

Next, we focus on the problem that has to be solved by `SolveSmall`, where all remaining clusters in $G - Y$ have size at most $3c$.

Lemma 4. *Let Y denote a cluster vertex deletion set for G of size c. If all clusters in $G - Y$ have size at most $3c$, then* CE *is fixed-parameter tractable.*

Proof. The basic idea to show fixed-parameter tractability is as follows. We group the clusters of $G-Y$ into different "types", where two clusters Q_i and Q_j of $G-Y$ have the same type if there is a graph-isomorphism ϕ between $G[Y \cup Q_i]$ and $G[Y \cup Q_j]$ such that $\forall v \in Y : \phi(v) = v$. We show that the number of types of the clusters in $G-Y$ is bounded by $2^{O(c^2)}$ and that if there is a type of which there are more than $O(c^2)$ clusters, then one of these clusters will be a cluster in the final cluster graph. Hence, we can delete all edges outgoing from this cluster and remove it from G. Afterwards, since each cluster contains at most $3c$ vertices the total number of vertices is bounded by a function only depending on c, directly implying fixed-parameter tractability.

To bound the number of cluster types, we first classify the vertices in $V \setminus Y$ in 2^c types; two vertices in $u, w \in V \setminus Y$ have the same type if $N_G(u) \cap Y = N_G(w) \cap Y$. Then, a cluster K can be described by a vector of length 2^c with at most $3c$ non-zero entries, where the ith entry contains the number of type-i vertices in K (note that the sum of entries does not exceed $3c$). Further, two clusters have the same type if these corresponding vectors are identical. Finally, observe that there are at most $\sum_{i=1}^{3c} \binom{3c \cdot 2^c}{i} \leq 3c \cdot (3c \cdot 2^c)^{3c} = 2^{O(c^2)}$ cluster types.

We now show that for each cluster type we can delete all but c^2 clusters. First, note that there are at most c clusters in a closest cluster graph that contain vertices from Y. Second, we can assume that each cluster of a closest cluster graph intersecting with Y contains vertices from at most c clusters of $G-Y$; it is straightforward to verify that otherwise separating the vertices of one cluster results in a cluster graph with the same or smaller edit distance to G. As a consequence, each of the at most c clusters intersecting with Y can contain vertices from at most c clusters of each type. Finally, note that if there is a cluster K of $G-Y$ such that in a closest cluster graph no vertex of K is in a cluster intersecting with Y, then K is a cluster of this closest cluster graph. Hence, if there are c^2+i, $1 \leq i$, clusters of the same type in $G-Y$, then at least i of these clusters are clusters in a closest cluster graph, and, hence, can be deleted (together with the edges between these clusters and Y). After deleting these clusters there are at most $2^{O(c^2)}c^2 + c = 2^{O(c^2)}$ vertices in G. □

Using these two results on the running times of `SolveLarge` and `SolveSmall`, we can show the fixed-parameter tractability with respect to c.

Theorem 1. Cluster Editing *is fixed-parameter tractable with respect to the cluster vertex deletion number c of G.*

Proof. We use Alg. 1 to compute an optimal solution for CE. First, we show that Alg. 1 is correct. To this end, let G' denote a cluster graph with closest edit distance to G. Since Alg. 1 enumerates all partitions of Y, the sets $Q_1, \ldots, Q_t$ (Line 5) once one-to-one correspond to the clusters in $G'[Y]$. By Lemma 2, all clusters of size at least $3c+1$ ("large cluster") in $G-Y$ either form a cluster in G' or are contained in a cluster of G' together with vertices from Y. Since the algorithm has already guessed the partition of Y, every such large cluster is contained in a cluster of G' with the vertices of at most one Q_i. By trying all two-partitions of $\{1, \ldots, t\}$, Alg. 1 guesses the Q_i's that are clusters in G'

or that are contained in a cluster together with one large cluster. Note that the corresponding subproblem exactly corresponds to FIXED CLIQUE CLUSTER EDITING since each clique Q_i is in a cluster with at most one large cluster B_j and vice versa. The remaining clusters in $G - Y$ all have size at most $3c$. Hence, by Lemma 4 the remaining instance can be solved in fpt-time.

For the running time note that there are $O(c^c)$ partitions of Y. For each such partition we try all two-partitions. Hence, Alg. 1 enters the body of the inner for loop $O(2^{c\log(c)+c})$ times. Since by Lemma 3 the subroutine `SolveLarge` can be applied in polynomial time and since subroutine `SolveSmall` runs in fpt-time (by Lemma 4), Alg. 1 runs in fpt-time. A naive estimation gives a bound of $2^{2^{O(c^2)}}$ for the combinatorial explosion in the running time. □

2.2 Cluster Deletion

For CLUSTER DELETION parameterized by the cluster vertex deletion number, we can achieve an algorithm with a better running time than the algorithm for CLUSTER EDITING (that in the subprocedure `SolveSmall` basically resorts to brute-force). The main feature of CLUSTER DELETION that helps in achieving this better algorithm is that none of the clusters in $G-Y$, where Y is the cluster vertex deletion set, can be merged since only edge deletions are allowed.

Theorem 2. CLUSTER DELETION *can be solved in* $c^{7c}|V|\cdot poly(n)$ *time, where* c *is the cluster vertex deletion number of* G.

3 Maximum Degree

We show that CLUSTER EDITING is NP-hard even when restricted to graphs with maximum degree six. To the best of our knowledge all previous NP-hardness proofs require an unbounded degree. As an immediate consequence, CLUSTER EDITING is NP-hard even if each vertex is only incident to a constant number of modified edges.

For the NP-hardness proof we present a reduction from 3-SAT. The basic idea of the construction is as follows. For each variable x_i of a given 3-CNF formula we construct a so-called variable cycle of length $4m$. It is easy to verify that only deleting every second edge gives an optimal solution for an even-length cycle. Thus, the two corresponding possibilities are used to represent the two choices for the value of x_i. Moreover, for each clause C_j containing the variables x_p, x_q, and x_r, we connect the three corresponding variable cycles by a clause gadget. In doing so, the goal is to ensure that if the solutions for the variable gadgets correspond to an assignment that satisfies C_j, then all P_3s of the clause gadget can be destroyed by four edge modifications and otherwise by at least five edge modifications. This guarantees that there is a satisfying assignment for the 3-CNF formula if and only if the constructed graph can be transformed into a cluster graph by exactly $2mn + 4m$ edge modifications. The details follow.

Given a 3-CNF formula ϕ consisting of the clauses $C_0, \ldots, C_{m-1}$ over the variables $\{x_0, \ldots, x_{n-1}\}$, construct a CE-instance consisting of a graph $G = (V, E)$ and an integer k as follows.

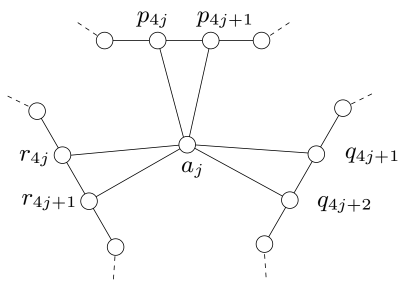


Fig. 1. Illustration of the clause gadget for a clause $C_j = (x_p \vee \overline{x_q} \vee x_r)$

For each variable x_i, $0 \leq i < n$, G contains a *variable cycle* consisting of the vertices $V_i^v := \{i_0, \ldots, i_{4m-1}\}$ and the edges $E_i^v := \{\{i_k, i_{k+1}\} \mid 0 \leq k < m\}$ (for ease of presentation let $i_{4m} = i_0$). So far, the constructed graph consists of a disjoint union of cycles, each of length $4m$. We use the following notation. The edges $\{i_0, i_1\}, \{i_2, i_3\}, \ldots, \{i_{4m-2}, i_{4m-1}\}$ of the variable cycle of x_i are called *even* and all others are called *odd*.

Moreover, for each clause C_j containing the variables x_p, x_q, and x_r (either negated or non-negated), we construct a clause gadget connecting the variable gadgets of x_p, x_q, and x_r. More specifically, let a_j be a new vertex and let E_j^c contain for each $i \in \{p, q, r\}$ the edges $\{a_j, i_{4j}\}$ and $\{a_j, i_{4j+1}\}$ if x_i occurs non-negated in C_j or the edges $\{a_j, i_{4j+1}\}$ and $\{a_j, i_{4j+2}\}$, otherwise. See Fig. 3 for an illustration. Finally, let $V := \bigcup_{i=0}^{n-1} V_i^v \cup \bigcup_{j=0}^{m-1} \{a_j\}$ and $E := \bigcup_{i=0}^{n-1} E_i^v \cup \bigcup_{j=0}^{m-1} E_j^c$. This completes the construction of $G = (V, E)$.

Theorem 3. Cluster Editing *and* Cluster Deletion *are NP-complete even when restricted to graphs with maximum vertex degree six.*

4 Number of Clusters and Maximum Number of Edge Modifications per Vertex

We show that a constrained version of Cluster Editing is fixed-parameter tractable with respect to the combined parameter "number d of clusters in the target graph" and "maximum number t of modifications per vertex". The problem under consideration is a generalization of Cluster Editing:

(d, t)-Constrained-Cluster Editing ((d, t)-CCE)
Input: An undirected graph $G = (V, E)$, a function $\tau : V \to \{0, \ldots, t\}$, and nonnegative integers d and k.
Question: Can G be transformed into a cluster graph G' by applying at most k edge modifications such that G' has at most d clusters and each vertex $v \in V$ is incident to at most $\tau(v)$ modified edges?

We use τ during our algorithm to keep track of the number of modifications that each vertex has been incident to. We can initially set $\tau(v) := t$ for each $v \in V$ and model directly the constraints described above. If only edge deletions are allowed, the corresponding problem is called (d, t)-Constrained-Cluster Deletion. Clearly, CE is exactly (n, n)-CCE. We investigate the parameterized complexity of (d, t)-CCE with respect to the combined parameter (d, t). Before presenting an algorithm for this problem, we discuss several aspects of both the problem formulation and parameterization.

Concerning the problem formulation, in many application scenarios a reasonable upper bound for the number of clusters d is given in advance. Furthermore, constraining the maximum number t of modifications per vertex yields another measure of closeness of the cluster graph to the input graph. In comparison to Cluster Editing, (d, t)-Constrained-Cluster Editing allows to further constrain the solution by adjusting the values of d and t. In certain application scenarios this may help to obtain more reasonable clusterings.

Concerning the parameterization, we consider the combined parameter (d, t) since Cluster Editing is NP-hard for $t = 6$ (which follows from Theorem 3). Moreover, when comparing the parameterizations k and (d, t) one can observe that for some instances, k is not bounded by a function in d and t. Consider for example a graph $G = (V, E)$ that consists of two cliques K_1 and K_2, each of order $|V|/2$. Furthermore, let each $v \in K_1$ have exactly one neighbor in K_2 and vice versa. An optimal solution for this graph is to delete all $|V|/2$ edges between K_1 and K_2. Hence, the parameter k is very large for this instance, whereas $d = 2$ and $t = 1$. In general, we can always assume $t \leq k$. The general relation between d and k is a bit more tricky. For example, in case G is connected, we can assume $d \leq k+1$ since we can create at most $k+1$ connected components by applying k edge modifications to G. Furthermore, in case G does not contain isolated cliques, we can assume $d \leq 2k$, since to each clique in the final cluster graph at least one edge modification is incident. In summary, the parameters d and t can be arbitrarily small compared to k and are bounded from above by a linear function of k when G does not contain isolated cliques.

We now show that (d, t)-Constrained-Cluster Editing is fixed-parameter tractable with respect to (d, t). More precisely, we present four data reduction rules for (d, t)-Constrained-Cluster Editing that produce a kernel consisting of at most $4dt$ vertices. The first two rules identify edge modifications that have to be performed by every solution, since otherwise there would be vertices to which more than t edge modifications are incident.

Reduction Rule 1. *If G contains two adjacent vertices $u, v \in V$ such that $|N(u) \setminus N[v]| > 2t$, then remove $\{u, v\}$ from E and set $\tau(v) := \tau(v) - 1$, $\tau(u) := \tau(u) - 1$, and $k := k - 1$.*

Reduction Rule 2. *If G contains two non-adjacent vertices $u, v \in V$ such that $|N(u) \cap N(v)| > 2t$, then add $\{u, v\}$ to E and set $\tau(v) := \tau(v) - 1$, $\tau(u) := \tau(u) - 1$, $k := k - 1$.*

Lemma 5. *Rules 1 and 2 are correct and can be exhaustively performed in $O(n^3)$ time.*

Proof. Let $(G = (V, E), d, t, k)$ be an input instance of (d, t)-Constrained-Cluster Editing. We show the correctness of each rule and then bound the running time of exhaustively applying both rules.

Let u and v be as described in Rule 1. We show that every solution deletes the edge $\{u, v\}$. Suppose that there is some solution S that does not delete $\{u, v\}$, let $G' := G \Delta S$ be the resulting cluster graph, and let K be the cluster of G' such that $u, v \in K$. Clearly, $|K \cap N(u) \setminus N[v]| \leq t$ since at most t inserted edges are incident to v. Then, however, more than t deleted edges are incident to u. This contradicts that S is a solution.

Let u and v be as described in Rule 2. We show that every solution adds the edge $\{u, v\}$. Suppose that there is some solution S that does not add $\{u, v\}$, let $G' := G \Delta S$ be the resulting cluster graph, and let K be the cluster of G' such that $u \in K$ and $v \notin K$. Since at most t deleted edges are incident to u, we have $|N(u) \cap N(v) \cap K| > t$. Then, however more than t deleted edges are incident to v. This contradicts that S is a solution.

To achieve a running time of $O(n^3)$ we proceed as follows. First, we initialize for each pair of vertices $u, v \in V$ three counters, one counter that counts $|N(u) \cap N(v)|$, one counter that counts $|N(u) \setminus N[v]|$, and one counter that counts $|N(v) \setminus N[u]|$. For each such pair, this is doable in $O(n)$ time when an adjacency matrix has been constructed in advance. Hence, the overall time for initializing the counters for all possible vertex pairs is $O(n^3)$. All counters that warrant an application of either Rule 1 or Rule 2 are stored in a list. We call these counters *active*. Next, we apply the reduction rules. Overall, since $k \leq n^2$ the rules can be applied at most n^2 times. As long as the list of active counters is non-empty, we perform the appropriate rule for the first active counter of the list. It remains to update all counters according to the edge modification applied by the rule. Suppose Rule 2 applies to u and v, that is $\{u, v\}$ is added. Then, we have to update the counters for each pair containing v or u. For v, this can be done in $O(n)$ time, by checking for each $w \neq v$, whether u must be added to $N(v) \cap N(w)$ or added to $N(v) \setminus N[w]$ or removed from $N(w) \setminus N[v]$ (for each counter this can be done in $O(1)$ time by using the constructed adjacency matrix). For each updated counter, we also check in $O(1)$ time whether it needs to be added to/removed from the list of active counters. The case that Rule 1 applies to u and v can be shown analogously. Overall, we need $O(n^3)$ time initialize the counters and $O(n^3)$ time for the exhaustive application of the rules. □

The following reduction rule simply checks whether the instance contains vertices to which already more than t modifications have been applied. Clearly, in this case the instance is a "no"-instance.

Reduction Rule 3. *If there is a vertex $v \in V$ with $\tau(v) < 0$, then output "no".*

The final rule identifies isolated cliques whose removal does not destroy solutions of (d, t)-Constrained-Cluster Editing.

Reduction Rule 4. *If there is an isolated clique K in G such that $|K| > 2t$, then remove K from G and set $d := d - 1$.*

Lemma 6. *Rule 4 is correct and can be exhaustively performed in $O(m)$ time.*

We now show that the reduction rules above yield a problem kernel.

Theorem 4. (d,t)-Constrained-Cluster Editing *admits a $4dt$-vertex problem kernel which can be found in $O(n^3)$ time.*

Proof. We first show the kernel size and then bound the running time of the kernelization.

Let $(G=(V,E),d,t,k)$ be an input instance of (d,t)-Constrained-Cluster Editing and let G be reduced with respect to Rules 1–4. We show the following:

(G,d,t,k) is a yes-instance $\Rightarrow$ G has at most $4dt$ vertices.

Let S be a solution of the input instance and let G' be the cluster graph that results from applying S to G. We show that every cluster K_i of G' has size at most $4t$. Assume towards a contradiction that there is some K_i in G' with $|K_i|>4t$. Since G is reduced with respect to Rule 4, there must be either an edge $\{u,v\}$ in G such that $u\in K_i$ and $v\in V\setminus K_i$ or a pair of vertices $u,v\in K_i$ such that $\{u,v\}$ is not an edge in G.
Case 1: $u\in K_i$, $v\in V\setminus K_i$ and $\{u,v\}\in E$. Since at most $t-1$ edge additions are incident to u, it has in G at least $3t+1$ neighbors in K_i. Furthermore, since at most t edge deletions are incident to v, it has in G at most t neighbors in K_i. Hence, there are at least $2t+1$ vertices in K_i that are neighbors of u but not neighbors of v. Therefore, Rule 1 applies in G, a contradiction to the fact that G is reduced with respect to this rule.
Case 2: $u,v\in K_i$ and $\{u,v\}\notin E$. Both u and v are in G adjacent to at least $|K_i|-(t-1)$ vertices of $K_i\setminus\{u,v\}$. Since $|K_i|>4t$ they thus have in G at least $2t+1$ common neighbors. Therefore, Rule 2 applies in G, a contradiction to the fact that G is reduced with respect to this rule.

We have shown that $|K_i|\le 4t$ for each cluster K_i of G'. Since G' has at most d clusters, the overall bound on the number of vertices follows.

It remains to bound the running time of exhaustively applying Rules 1–4. By Lemma 5, the exhaustive application of Rules 1 and 2 runs in $O(n^3)$ time. After these two rules have been exhaustively applied, Rules 3 and 4 can be exhaustively applied in $O(m)$ time. □

Corollary 1. (d,t)-Constrained-Cluster Editing *is fixed-parameter tractable with respect to the parameter (d,t).*

The data reduction rules can be adapted to the case that only edge deletions are allowed. Indeed, we can show a $2dt$-vertex kernel for (d,t)-Constrained-Cluster Deletion by just replacing $2t$ by t in Rules 1 and 4 (no further data reduction rule is needed). Then, with a slightly modified analysis, we arrive at the following (details omitted).

Theorem 5. (d,t)-Constrained-Cluster Deletion *admits a $2dt$-vertex kernel which can be found in $O(n^3)$ time. It is thus fixed-parameter tractable with respect to the parameter (d,t).*

References

[1] Bansal, N., Blum, A., Chawla, S.: Correlation clustering. Machine Learning 56(1-3), 89–113 (2004)

[2] Böcker, S., Briesemeister, S., Bui, Q.B.A., Truß, A.: Going weighted: Parameterized algorithms for cluster editing. Theor. Comput. Sci. 410(52), 5467–5480 (2009)

[3] Böcker, S., Briesemeister, S., Klau, G.W.: Exact algorithms for cluster editing: Evaluation and experiments. Algorithmica (2009) (to appear)

[4] Chen, J., Meng, J.: A $2k$ kernel for the cluster editing problem. In: Thai, M.T., Sahni, S. (eds.) COCOON 2010. LNCS, vol. 6196, pp. 459–468. Springer, Heidelberg (2010)

[5] Damaschke, P.: Fixed-parameter enumerability of cluster editing and related problems. Theory Comput. Syst. 46(2), 261–283 (2010)

[6] Dehne, F.K.H.A., Langston, M.A., Luo, X., Pitre, S., Shaw, P., Zhang, Y.: The cluster editing problem: Implementations and experiments. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 13–24. Springer, Heidelberg (2006)

[7] Demaine, E.D., Hajiaghayi, M., Marx, D.: Open problems – parameterized complexity and approximation algorithms. In: Parameterized Complexity and Approximation Algorithms. Dagstuhl Seminar Proceedings, vol. 09511. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany (2010)

[8] Fellows, M.R., Guo, J., Komusiewicz, C., Niedermeier, R., Uhlmann, J.: Graph-based data clustering with overlaps. In: Ngo, H.Q. (ed.) COCOON 2009. LNCS, vol. 5609, pp. 516–526. Springer, Heidelberg (2009)

[9] Fellows, M.R., Langston, M.A., Rosamond, F.A., Shaw, P.: Efficient parameterized preprocessing for Cluster Editing. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, pp. 312–321. Springer, Heidelberg (2007)

[10] Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Graph-modeled data clustering: Exact algorithms for clique generation. Theory Comput. Syst. 38(4), 373–392 (2005)

[11] Guo, J.: A more effective linear kernelization for cluster editing. Theor. Comput. Sci. 410(8-10), 718–726 (2009)

[12] Guo, J., Komusiewicz, C., Niedermeier, R., Uhlmann, J.: A more relaxed model for graph-based data clustering: s-plex cluster editing. SIAM Journal on Discrete Mathematics (2010) (to appear)

[13] Hüffner, F., Komusiewicz, C., Moser, H., Niedermeier, R.: Fixed-parameter algorithms for cluster vertex deletion. Theory Comput. Syst. 47(1), 196–217 (2010)

[14] Křivánek, M., Morávek, J.: NP-hard problems in hierarchical-tree clustering. Acta Inform. 23(3), 311–323 (1986)

[15] Niedermeier, R.: Reflections on multivariate algorithmics and problem parameterization. In: Proc. 27th STACS, vol. 5, pp. 17–32. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, LIPIcs (2010)

[16] Protti, F., da Silva, M.D., Szwarcfiter, J.L.: Applying modular decomposition to parameterized cluster editing problems. Theory Comput. Syst. 44(1), 91–104 (2009)

[17] Shamir, R., Sharan, R., Tsur, D.: Cluster graph modification problems. Discrete Appl. Math. 144(1–2), 173–182 (2004)

The Complexity of Finding kth Most Probable Explanations in Probabilistic Networks[⋆]

Johan H.P. Kwisthout, Hans L. Bodlaender, and Linda C. van der Gaag

Department of Information and Computing Sciences,
Utrecht University, Utrecht, The Netherlands

Abstract. In modern decision-support systems, probabilistic networks model uncertainty by a directed acyclic graph quantified by probabilities. Two closely related problems on these networks are the Kth MPE and Kth Partial MAP problems, which both take a network and a positive integer k for their input. In the Kth MPE problem, given a partition of the network's nodes into evidence and explanation nodes and given specific values for the evidence nodes, we ask for the kth most probable combination of values for the explanation nodes. In the Kth Partial MAP problem in addition a number of unobservable intermediate nodes are distinguished; we again ask for the kth most probable explanation. In this paper, we establish the complexity of these problems and show that they are $\mathsf{FP}^{\mathsf{PP}}$- and $\mathsf{FP}^{\mathsf{PP}^{\mathsf{PP}}}$-complete, respectively.

1 Introduction

For modern decision-support systems, probabilistic networks are rapidly becoming the models of choice for representing and reasoning with uncertainty. Applications of these networks have been developed for a range of problem domains which are fraught with uncertainty. Most notably, applications are being realised in the biomedical field where they are designed to support medical and veterinary practitioners in their diagnostic reasoning processes; examples from our own engineering experiences include a network for diagnosing ventilator-associated pneumonia in critically ill patients [1] and a network for the early detection of an infection with the Classical Swine Fever virus in pigs [2].

A probabilistic network is a concise model of a joint probability distribution over a set of stochastic variables [3]. It consists of a directed acyclic graph, encoding the relevant variables and their probabilistic interdependencies, and an associated set of conditional probabilities. Various algorithms have been designed for probabilistic inference, that is, for computing probabilities of interest from a probabilistic network. These algorithms typically exploit structural properties of the network's graph to decompose the computations involved. Probabilistic inference is known to be PP-complete in general [4]. Many other problems to be

⋆ This work was supported by the Netherlands Organisation for Scientific Research (NWO), project *Complexity of Probabilistic Networks*.

solved in practical applications of probabilistic networks are also known to have a quite unfavourable complexity.

In many practical applications, the nodes of a probabilistic network are partitioned into evidence nodes, explanation nodes and intermediate nodes. The evidence nodes model variables whose values can be observed in reality; in a medical application, these nodes typically model a patient's observable symptoms. The explanation nodes model the variables for which a most likely value needs to be found; these nodes typically capture possible diagnoses. The intermediate nodes are included in the network to correctly represent the probabilistic dependencies among the variables; in a medical application, these nodes often model physiological processes hidden in a patient. An important problem in probabilistic networks now is to find the most likely value combination for the explanation nodes given a specific joint value for the evidence nodes. When the network's set of intermediate nodes is empty, the problem is known as the most probable explanation, or MPE, problem; the problem is coined the partial maximum aposteriori probability, or Partial MAP, problem otherwise. The MPE problem is known to have various NP-complete decision variants [5,6]; for the Partial MAP problem $\mathsf{NP}^{\mathsf{PP}}$-completeness was established [7].

In many applications of probabilistic networks, the user is interested not just in finding the most likely explanation for a combination of observations, but in finding alternative explanations as well. In biomedicine, for example, a practitioner may wish to start antibiotic treatment for multiple likely pathogens before the actual cause of infection in a patient is known; alternative explanations may also reveal whether or not further diagnostic testing can help distinguishing between possible diagnoses. In the absence of intermediate nodes in a network, the problem of finding the kth most likely explanation is known as the Kth MPE problem; it is called the Kth Partial MAP problem otherwise. While efficient algorithms have been designed for solving the kth most probable explanation problem with the best explanation as additional input [8], the Kth MPE problem without this extra information is NP-hard in general [9]. The complexity of the Kth Partial MAP problem is unknown as yet.

In this paper, we study the computational complexity of the Kth MPE and Kth Partial MAP problems in probabilistic networks and show that these problems are complete for the complexity classes $\mathsf{FP}^{\mathsf{PP}}$ and $\mathsf{FP}^{\mathsf{PP}^{\mathsf{PP}}}$, respectively. This finding suggests that the two problems are much harder than the (already intractable) restricted problems of finding a most likely explanation. Finding the kth most probable explanation in a probabilistic network given partial evidence to our best knowledge is the first problem with a practical application that is shown to be $\mathsf{FP}^{\mathsf{PP}^{\mathsf{PP}}}$-complete, which renders our result interesting at least from a theoretical point of view.

The paper is organised as follows. In Section 2, our notational conventions are introduced, as are the definitions used in the paper. We discuss the computational complexity of finding kth joint value assignments with full and partial evidence in the Sections 3 and 4, respectively. Section 5 concludes the paper.

2 Definitions

In this section, we provide definitions for the concepts used in the sequel. In Section 2.1, we briefly review probabilistic networks and introduce our notational conventions. In Section 2.2, we describe the problems under study. In Section 2.3, we review some complexity classes and state complete problems for these classes.

2.1 Probabilistic Networks

A probabilistic network is a model of a joint probability distribution over a set of stochastic variables. Before defining the concept of probabilistic network more formally, we introduce some notational conventions. Stochastic variables are denoted by capital letters with a subscript, such as X_i; we use bold-faced upper-case letters $\mathbf{X}$ to denote sets of variables. A lower-case letter x is used for a value of a variable X; a combination of values for a set of variables $\mathbf{X}$ is denoted by a bold-faced lower-case letter $\mathbf{x}$ and will be termed a joint value assignment to $\mathbf{X}$. In the sequel, we assume that all joint value assignments to a set of variables $\mathbf{X}$ are uniquely ordered by their (possibly posterior) probability in numerically descending order; if two joint value assignments $\mathbf{x}_i$ and $\mathbf{x}_j$ have the same probability, they are ordered lexicographically in descending order by their respective binary representation, taking the value for the variable X_1 to be the most significant element.

A probabilistic network now is a tuple $\mathcal{B} = (\mathbf{G}, \Gamma)$ where $\mathbf{G} = (\mathbf{V}, A)$ is a directed acyclic graph and Γ is a set of conditional probability distributions. Each node $V_i \in \mathbf{V}$ models a stochastic variable. The set of arcs A of the graph captures probabilistic independence: two nodes V_i and V_j are independent given a set of nodes $\mathbf{W}$, if either V_i or V_j is in $\mathbf{W}$, or if every chain between V_i and V_j in $\mathbf{G}$ contains a node from $\mathbf{W}$ with at least one emanating arc or a node V_k with two incoming arcs such that neither V_k itself nor any of its descendants are in $\mathbf{W}$. For a topological sort $V_1, \ldots, V_n$ of $\mathbf{G}$, we now have that any node V_i is independent of the preceding nodes $V_1, \ldots, V_{i-1}$ given its set of parents $\pi(V_i)$. The set Γ of the network includes for each node V_i the conditional probability distributions $\Pr(V_i \mid \pi(V_i))$ that describe the influence of the various assignments to V_i's parents $\pi(V_i)$ on the probabilities of the values of V_i itself.

A probabilistic network $\mathcal{B} = (\mathbf{G}, \Gamma)$ defines a joint probability distribution $\Pr(\mathbf{V}) = \prod_{V_i \in \mathbf{V}} \Pr(V_i \mid \pi(V_i))$ that respects the independencies portrayed by its digraph. Since it defines a unique distribution, a probabilistic network allows the computation of any probability of interest over its variables [10].

2.2 The *k*th Most Probable Explanation Problems

The main problem studied in this paper is the problem of finding a kth most probable explanation for a particular combination of observations, for arbitrary values of k. Formulated as a functional problem, it is defined as follows.

Kth MPE
Instance: A probabilistic network $\mathcal{B} = (\mathbf{G}, \Gamma)$, where $\mathbf{V}$ is partitioned into a set of evidence nodes $\mathbf{E}$ and a set of explanation nodes $\mathbf{M}$; a joint value assignment $\mathbf{e}$ to $\mathbf{E}$; and a positive natural number k.
Output: The kth most probable joint value assignment $\mathbf{m}_k$ to $\mathbf{M}$ given $\mathbf{e}$; if no such assignment exists, the output is $\perp$, that is, the universal *false*.

Note that the Kth MPE problem defined above includes the MPE problem as a special case with $k = 1$. From $\Pr(\mathbf{m} \mid \mathbf{e}) = \frac{\Pr(\mathbf{m},\mathbf{e})}{\Pr(\mathbf{e})}$, we further observe that $\Pr(\mathbf{e})$ can be regarded a constant if we are interested in the relative order only of the conditional probabilities $\Pr(\mathbf{m} \,|\, \mathbf{e})$ of all joint value assignments $\mathbf{m}$.

While for the Kth MPE problem a network's nodes are partitioned into evidence and explanation nodes only, the Kth Partial MAP problem discerns also intermediate nodes. We define a bounded variant of the latter problem.

Bounded Kth Partial MAP
Instance: A probabilistic network $\mathcal{B} = (\mathbf{G}, \Gamma)$, where $\mathbf{V}$ is partitioned into a set of evidence nodes $\mathbf{E}$, a set of intermediate nodes $\mathbf{I}$, and a set of explanation nodes $\mathbf{M}$; a joint value assignment $\mathbf{e}$ to $\mathbf{E}$; a positive natural number k; and rational numbers a, b with $0 \leq a \leq b \leq 1$.
Output: The tuple $(\mathbf{m}_k, p_k)$, where $\mathbf{m}_k$ is the kth most probable assignment to $\mathbf{M}$ given $\mathbf{e}$ from among all joint value assignments $\mathbf{m}_i$ to $\mathbf{M}$ with $p_i = \Pr(\mathbf{m}_i, \mathbf{e}) \in [a, b]$; if no such assignment exists, the output is $\perp$.

Note that the original Kth Partial MAP problem without bounds is a special case of the problem defined above with $a = 0$ and $b = 1$. Further note that the bounded problem can be transformed into a problem without bounds in polynomial time and vice versa, which renders the two problems Turing equivalent. In the sequel, we will use the bounded problem to simplify our proofs.

2.3 Complexity Classes and Complete Problems

We assume throughout the paper that the reader is familiar with the standard notion of a Turing machine and with the basic concepts from complexity theory. We further assume that the reader is acquainted with complexity classes such as $\mathsf{NP}^{\mathsf{PP}}$, for which certificates of membership can be verified in polynomial time given access to an oracle. For these classes, we recall that the defining Turing machine can write a string to an oracle tape and takes the next step conditional on whether or not the string on this tape belongs to a particular language; for further details on complexity classes involving oracles, we refer to [11,12,13].

While Turing machines are tailored to solving decision problems, halting either in an accepting state or in a rejection state, Turing transducers can generate functional results: if a Turing transducer halts in an accepting state, it returns a result on an additional output tape. The complexity classes FP and FNP now are the functional variants of P and NP, and are defined using Turing transducers instead of Turing machines. Just like a Turing machine, a Turing transducer can have access to an oracle; for example, $\mathsf{FP}^{\mathsf{NP}}$ is the class of functions computable in polynomial time by a Turing transducer with access to an NP oracle. Since the kth most probable explanation problems under study require the computation of a result, we will use Turing transducers in the sequel.

Metric Turing machines are used to show membership in complexity classes like P^{NP}or P^{PP}[12]. A metric Turing machine $\hat{\mathcal{M}}$ is a polynomial-time bounded non-deterministic Turing machine in which every computation path halts with a binary number on a designated output tape. $\mathrm{Out}_{\hat{\mathcal{M}}}(x)$ denotes the set of outputs of $\hat{\mathcal{M}}$ on input x; $\mathrm{Opt}_{\hat{\mathcal{M}}}(x)$ is the smallest number in $\mathrm{Out}_{\hat{\mathcal{M}}}(x)$, and $\mathrm{KthValue}_{\hat{\mathcal{M}}}(x, k)$ is the k-th smallest number in $\mathrm{Out}_{\hat{\mathcal{M}}}(x)$. Metric Turing transducers $\hat{\mathcal{T}}$ are defined likewise as Turing transducers with an additional output tape; these transducers are used for proving membership in $\mathsf{FP}^{\mathsf{NP}}$ or $\mathsf{FP}^{\mathsf{PP}}$.

A function f is polynomial-time one-Turing reducible to a function g, written $f \leq^{\mathsf{FP}}_{\text{1-T}} g$, if there exist polynomial-time computable functions T_1 and T_2 such that $f(x) = T_1(x, g(T_2(x)))$ for every x [13]. A function f now is in $\mathsf{FP}^{\mathsf{NP}}$ if and only if there exists a metric Turing transducer $\hat{\mathcal{T}}$ such that $f \leq^{\mathsf{FP}}_{\text{1-T}} \mathrm{Opt}_{\hat{\mathcal{T}}}$. Correspondingly, a set L is in P^{NP} if and only if a metric Turing machine $\hat{\mathcal{M}}$ can be constructed, such that $\mathrm{Opt}_{\hat{\mathcal{M}}}(x)$ is odd if and only if $x \in L$. Similar observations hold for $\mathsf{FP}^{\mathsf{PP}}$ and P^{PP}, and the $\mathrm{KthValue}_{\hat{\mathcal{M}}}$ and $\mathrm{KthValue}_{\hat{\mathcal{T}}}$ functions [12,13]. $\mathsf{FP}^{\mathsf{NP}}$- and $\mathsf{FP}^{\mathsf{PP}}$-hardness can be proved by a reduction from a known $\mathsf{FP}^{\mathsf{NP}}$- and $\mathsf{FP}^{\mathsf{PP}}$-hard problem, respectively, using a polynomial-time one-Turing reduction.

We define two functional variants of the well-known satisfiability problem and state their completeness in terms of functional complexity classes.

Kth Sat
Instance: A Boolean formula $\phi(X_1, \ldots, X_n), n \geq 1$; and a positive natural number k.
Output: The lexicographically kth largest truth assignment $\mathbf{x}_k$ to $\mathbf{X} = \{X_1, \ldots, X_n\}$ that satisfies ϕ; if no such assignment exists, the output is $\perp$.

The LexSat problem is the special case of the Kth Sat problem with $k = 1$. LexSat and Kth Sat are complete for $\mathsf{FP}^{\mathsf{NP}}$ and $\mathsf{FP}^{\mathsf{PP}}$, respectively [12,13].

KthNumSat
Instance: A Boolean formula $\phi(X_1, \ldots, X_m, \ldots, X_n), m \leq n, n \geq 1$; and positive natural numbers k, l.
Output: The lexicographically kth largest assignment $\mathbf{x}_k$ to $\{X_1, \ldots, X_m\}$ for which exactly l assignments $\mathbf{x}_l$ to $\{X_{m+1}, \ldots, X_n\}$ satisfy ϕ; the output is $\perp$ if no such assignment exists.

The LexNumSat problem is the special case of the KthNumSat problem with $k = 1$. KthNumSat and LexNumSat are $\mathsf{FP}^{\mathsf{PP}^{\mathsf{PP}}}$- and $\mathsf{FP}^{\mathsf{NP}^{\mathsf{PP}}}$-complete; proofs will be provided in a full paper [14].

To facilitate the reductions in our proofs in the sequel, we further introduce slightly modified variants of the Kth Sat and KthNumSat problems which serve to circumvent the need of explicitly dealing with outputs equal to $\perp$.

Kth Sat′
Instance: A Boolean formula $\phi(X_1, \ldots, X_n), n \geq 1$, with

$$\phi(X_1, X_2, \ldots, X_n) = (\neg X_1) \vee \phi'(X_2, \ldots, X_n)$$

for some Boolean formula $\phi'(X_2, \ldots, X_n)$; and a positive natural number $k \leq 2^{n-1}$.
Output: The lexicographically kth largest truth assignment $\mathbf{x}_k$ to $\mathbf{X} = \{X_1, \ldots, X_n\}$ that satisfies ϕ.

Note that the Kth Sat′ problem differs from the original Kth Sat problem only in that it never has the output $\perp$ for its solution: since the problem's formula ϕ has at least 2^{n-1} satisfying truth assignments, a kth largest satisfying assignment is guaranteed to exist for any value of k with $k \leq 2^{n-1}$. For the original Kth Sat problem on the other hand, the number of values k for which a satisfying truth assignment is returned, depends on the precise formula ϕ. For the Kth Sat′ problem we observe moreover that, within the descending lexicographic order, all satisfying truth assignments that set X_1 to *false* come after all assignments with $X_1 = true$ that satisfy ϕ'. We further note that a simple transformation suffices to show that Kth Sat′ is complete for the complexity class $\mathsf{FP}^{\mathsf{PP}}$.

Using a similar yet slightly more involved construction, we pose a variant of the KthNumSat problem which never has the output $\perp$ for its solution.
KthNumSat′
Instance: A Boolean formula $\phi(X_1, \ldots, X_m, \ldots, X_n)$, $m \leq n$, $n \geq 1$, with

$$\phi(X_1, \ldots, X_n) = \phi'(X_2, \ldots, X_n) \vee ((\neg X_1) \wedge \psi^l(X_{m+1}, \ldots, X_n))$$

for some Boolean formula $\phi'(X_2, \ldots, X_n)$ and terms $\psi^l(X_{m+1}, \ldots, X_n)$ which express that $X_{m+1} \cdots X_n$ seen as a binary number is at most l; and positive natural numbers $k \leq 2^{m-1}$, $l \leq 2^{n-m-1}$.
Output: The lexicographically kth largest assignment $\mathbf{x}_k$ to $\{X_1, \ldots, X_m\}$ with which exactly l assignments $\mathbf{x}_l$ to $\{X_{m+1}, \ldots, X_n\}$ satisfy ϕ.

Note that each term ψ^l in the Boolean formula above has exactly l satisfying truth assignments. We further note that a simple transformation again suffices to show that the KthNumSat′ problem is complete for $\mathsf{FP}^{\mathsf{PP}^{\mathsf{PP}}}$.

3 Complexity of Kth MPE

We study the complexity of the Kth MPE problem introduced in Section 2.2 and prove $\mathsf{FP}^{\mathsf{PP}}$-completeness. To prove membership of $\mathsf{FP}^{\mathsf{PP}}$, we show that the problem can be solved in polynomial time by a metric Turing transducer; we prove hardness by a reduction from the Kth Sat′ problem defined above.

We begin by describing the construction of a probabilistic network $\mathcal{B}_\phi$ from the Boolean formula ϕ of an instance of the Kth Sat′ problem; upon doing so, we use the formula $\phi_{\text{ex}} = (\neg X_1) \vee (\neg X_2 \wedge (X_3 \vee \neg X_4))$ for our running example. For each Boolean variable X_i in ϕ, we include a root node X_i in the network

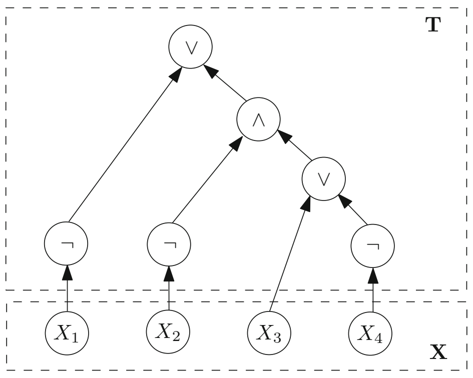


Fig. 1. The acyclic directed graph of the probabilistic network $\mathcal{B}_{\phi_{\text{ex}}}$ constructed from the Boolean formula $\phi_{\text{ex}} = (\neg X_1) \vee (\neg X_2 \wedge (X_3 \vee \neg X_4))$

$\mathcal{B}_\phi$, with *true* and *false* for its possible values; the nodes X_i with each other are called the variable-instantiation part $\mathbf{X}$ of the network. The prior probabilities $p_i = \Pr(X_i = \mathit{true})$ for the nodes X_i are chosen such that the prior probability of a joint value assignment $\mathbf{x}$ to $\mathbf{X}$ is higher than that of $\mathbf{x}'$ if and only if the corresponding truth assignment $\mathbf{x}$ to the Kth Sat′ variables $X_1, \ldots, X_n$ is ordered before $\mathbf{x}'$ in descending lexicographic order. More specifically, we set $p_i = \frac{1}{2} + \frac{2^{n+1-i}-1}{2^{n+1}}$; in our running example with four variables, the prior probabilities for the nodes $X_1, \ldots, X_4$ thus are set to $p_1 = \frac{31}{32}$, $p_2 = \frac{23}{32}$, $p_3 = \frac{19}{32}$, and $p_4 = \frac{17}{32}$. We observe that we have that $p_1 \cdot \ldots \cdot p_{i-1} \cdot \overline{p_i} > \overline{p_1} \cdot \cdots \cdot \overline{p_{i-1}} \cdot p_i$ for every i. Since the root nodes X_i are modelled as being mutually independent, the ordering property stated above is thereby satisfied in the network under construction. Further note that the assigned probabilities can be formulated using a number of bits which is polynomial in the number of variables of the Kth Sat′ instance.

For each logical operator in the Boolean formula ϕ, we create an additional node in the network $\mathcal{B}_\phi$. The parents of this node are the nodes corresponding with the subformulas joined by the operator; its conditional probability table is set to mimic the operator's truth table. The node associated with the top-level operator of ϕ will be denoted by V_ϕ. The operator nodes with each other constitute the truth-setting part $\mathbf{T}$ of the network. The probabilistic network $\mathcal{B}_{\phi_{\text{ex}}}$ that is constructed from the example formula ϕ_{ex} is shown in Figure 1. From the above construction, it is now readily seen that, given a value assignment $\mathbf{x}$ to the variable-instantiation part of the network, we have $\Pr(V_\phi = \mathit{true} \mid \mathbf{x}) = 1$ if and only if the truth assignment $\mathbf{x}$ to the Boolean variables X_i satisfies ϕ.

Theorem 1. Kth MPE *is* $\mathsf{FP}^{\mathsf{PP}}$*-complete.*

Proof. To prove membership of $\mathsf{FP}^{\mathsf{PP}}$ for the Kth MPE problem, we show that a metric Turing transducer can be constructed to solve the problem. Let $\hat{\mathcal{T}}$ be a metric Turing transducer that on input $(\mathcal{B}, \mathbf{e}, k)$ performs the following computations: it traverses a topological sort of the network's nodes $\mathbf{V}$; in each step i, it non-deterministically chooses a value v_i for node V_i (for a node E_i from the set $\mathbf{E}$ of evidence nodes, the value conform $\mathbf{e}$ is chosen) and multiplies the corresponding (conditional) probabilities along its path. Each computation path thereby establishes a joint probability $\Pr(\mathbf{v}) = \prod_{V_i \in \mathbf{V}} \Pr(v_i \mid \pi(V_i))$ for a thus constructed joint value assignment $\mathbf{v}$ to $\mathbf{V}$. Note that $\Pr(\mathbf{v}) = \Pr(\mathbf{m}, \mathbf{e})$ for the constructed assignment $\mathbf{m}$ to the explanation variables $\mathbf{M}$. Further note that the computations involved take a time which is polynomial in the number of variables in the Kth MPE instance. The output of the transducer is, for each computation path, a binary representation of $1 - \Pr(\mathbf{m}, \mathbf{e})$ with sufficient (but polynomial) precision, combined with the logical inverse of the binary representation of the assignment $\mathbf{m}$ itself. $\text{KthValue}_{\hat{\mathcal{T}}}(\mathcal{B}, \mathbf{e}, k)$ now returns an encoding of the kth most probable explanation for $\mathbf{e}$. We conclude that Kth MPE is in $\mathsf{FP}^{\mathsf{PP}}$.

To prove hardness for the class $\mathsf{FP}^{\mathsf{PP}}$, we reduce the Kth Sat′ problem defined in Section 2.3 to Kth MPE. Let (ϕ, k) be an instance of Kth Sat′. From the Boolean formula ϕ we construct the probabilistic network $\mathcal{B}_\phi$ as described above; we further let $\mathbf{E} = \{V_\phi\}$ and let $\mathbf{e}$ be the value assignment $V_\phi = \mathit{true}$. The thus constructed instance of the Kth MPE problem is $(\mathcal{B}_\phi, V_\phi = \mathit{true}, k)$; note that the construction can be performed in polynomial time. For any joint value assignment $\mathbf{x}$ to the variable-instantiation part $\mathbf{X}$ of $\mathcal{B}_\phi$, we now have that $\Pr(\mathbf{X} = \mathbf{x} \,|\, V_\phi = \mathit{true}) = \frac{\Pr(\mathbf{X}=\mathbf{x}, V_\phi=\mathit{true})}{\Pr(V_\phi=\mathit{true})} = \alpha \cdot \Pr(\mathbf{X} = \mathbf{x}, V_\phi = \mathit{true})$, where $\alpha = \Pr(V_\phi = \mathit{true})^{-1}$ can be regarded a normalisation constant. For any assignment $\mathbf{x}$ to the variables $\mathbf{X}$ that satisfies ϕ, we further find that $\Pr(\mathbf{X} = \mathbf{x} \,|\, V_\phi = \mathit{true}) = \alpha \cdot \Pr(\mathbf{X} = \mathbf{x})$; for any non-satisfying assignment $\mathbf{x}$ on the other hand, we have that $\Pr(\mathbf{X} = \mathbf{x}, V_\phi = \mathit{true}) = 0$ and hence $\Pr(\mathbf{X} = \mathbf{x} \,|\, V_\phi = \mathit{true}) = 0$. In terms of their posterior probabilities given $V_\phi = \mathit{true}$, therefore, all satisfying joint value assignments are ordered before all non-satisfying ones. Since the values of the nodes from the truth-setting part $\mathbf{T}$ are fully determined by the values of their parents, we thus have that, given evidence $V_\phi = \mathit{true}$, the kth MPE corresponds to the lexicographically kth satisfying value assignment to the variables in ϕ, and vice versa. Given an algorithm for solving Kth MPE, we can thus solve KthSat′ as well, which proves $\mathsf{FP}^{\mathsf{PP}}$-hardness of Kth MPE. □

We now turn to the case where $k = 1$, that is, to the basic MPE problem, for which we show $\mathsf{FP}^{\mathsf{NP}}$-completeness by a similar construction as above.

Proposition 1. MPE *is* $\mathsf{FP}^{\mathsf{NP}}$*-complete.*

Proof. To prove membership of $\mathsf{FP}^{\mathsf{NP}}$, a metric Turing transducer as above is constructed. $\text{Opt}_{\hat{\mathcal{T}}}(\mathcal{B}, \mathbf{e})$ then returns the most probable explanation given the evidence $\mathbf{e}$. To prove hardness, we apply the same construction as above to reduce, in polynomial time, the LexSat problem to the MPE problem. □

Note that the functional variant of the MPE problem is in $\mathsf{FP}^{\mathsf{NP}}$, while its decision variant is in NP [6]. This relation between the decision and functional variants of a problem is quite commonly found in optimisation problems: if the solution of a functional problem variant has polynomially bounded length, then there exists a polynomial-time Turing reduction from the functional variant to the decision variant of that problem, and hence if the decision variant is in NP, then the functional variant of the problem is in $\mathsf{FP}^{\mathsf{NP}}$ [15].

4 Complexity of K-th Partial MAP

While the decision variant of the MPE problem is complete for the class NP, the decision variant of the Partial MAP problem is known to be $\mathsf{NP}^{\mathsf{PP}}$-complete [7]. In the previous section, we proved that the functional variant of the Kth MPE problem is $\mathsf{FP}^{\mathsf{PP}}$-complete. Intuitively, these results suggest that the Kth Partial MAP problem is complete for the complexity class $\mathsf{FP}^{\mathsf{PP}^{\mathsf{PP}}}$. To the best of our knowledge, no complete problems have been discussed in the literature for this class. We will now show that the Kth Partial MAP problem indeed is complete for the class $\mathsf{FP}^{\mathsf{PP}^{\mathsf{PP}}}$, by a reduction from the KthNumSat$'$ problem.

We address the construction of a probabilistic network $\mathcal{B}_\phi$ from the Boolean formula ϕ of an instance of the KthNumSat$'$ problem. We recall from Section 2.3 that we defined KthNumSat$'$ so as to forestall the need of dealing with outputs equal to $\perp$. While formally our reduction should be from KthNumSat$'$, we decided, for ease of exposition, to use an instance of the original KthNum-Sat problem for our running example and to assume that the Boolean formula involved has 'sufficiently many' satisfying truth assignments, that is, sufficiently many in terms of the constants k, l. For our running example, we use the Boolean formula $\phi_{\text{ex}} = ((X_1 \vee \neg X_2) \wedge X_3) \vee \neg X_4$, for which we want to find the lexicographically second assignment to the variables $\{X_1, X_2\}$ with which exactly three truth assignments to $\{X_3, X_4\}$ satisfy ϕ_{ex}, that is, $k = 2$, $l = 3$; the reader can verify that the instance has the solution $X_1 = \mathit{true}, X_2 = \mathit{false}$.

As before, we create a root node X_i for each Boolean variable X_i from the formula ϕ, this time with a uniform prior probability distribution. The nodes $X_1, \ldots, X_m$ with each other constitute the variable-instantiation part $\mathbf{X}$ of the network $\mathcal{B}_\phi$; the nodes from this part will be taken as the explanation nodes for the Kth Partial MAP instance under construction. For the logical operators from the formula ϕ, we add nodes to the probabilistic network as before, with V_ϕ being the node associated with the top-level operator. Note that for any joint value assignment $\mathbf{x}$ to the explanation nodes $\mathbf{X}$, we now have that $\Pr(V_\phi = \mathit{true} \mid \mathbf{x}) = \frac{s}{2^{n-m}}$, where s is the number of truth value assignments to the Boolean variables $\{X_{m+1}, \ldots, X_n\}$ that, jointly with $\mathbf{x}$, satisfy ϕ.

We further construct an enumeration part $\mathbf{N}$ for the network. For this purpose, we add nodes $Y_1, \ldots, Y_m$, with the possible values *true* and *false*, to the variable-instantiation part $\mathbf{X}$ such that for all $i = 1, \ldots, m$, node Y_i has node X_i for its unique parent; for each such node Y_i, we set $\Pr(Y_i = \mathit{true} \mid X_i = \mathit{true}) = \frac{1}{2^{i+n-m+1}}$ and $\Pr(Y_i = \mathit{true} \mid X_i = \mathit{false}) = 0$. We further add a binary-tree structure to the network, composed of nodes E_j mimicking the disjunction operator; the arcs of this tree structure are directed from the leaves $Y_1, \ldots, Y_m$ to the root node, which will be denoted by E_ϕ. For our running example, we thus add nodes Y_1 and Y_2 as successors to the nodes X_1 and X_2, respectively, with $\Pr(Y_1 = \mathit{true} \mid X_1 = \mathit{true}) = \frac{1}{16}$ and $\Pr(Y_2 = \mathit{true} \mid X_2 = \mathit{true}) = \frac{1}{32}$. In addition, a single

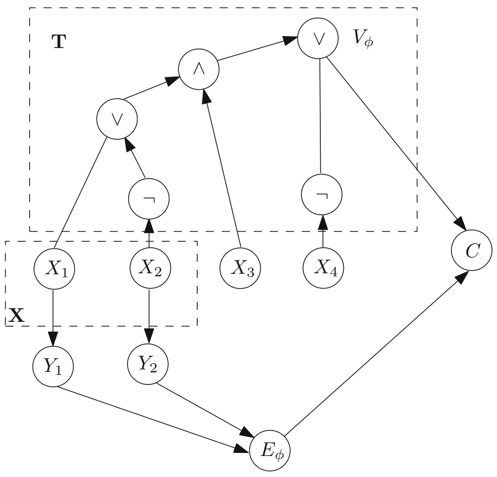


Fig. 2. The acyclic directed graph of the probabilistic network $\mathcal{B}_{\phi_{\rm ex}}$ constructed from the KthNumSat instance with the Boolean formula $\phi_{\rm ex} = ((X_1 \vee \neg X_2) \wedge X_3) \vee \neg X_4$ and the explanation nodes X_1, X_2

root node E_ϕ is added to the network under construction, with Y_1 and Y_2 for its parents. Note that for each joint value assignment $\mathbf{x}$ to the explanation nodes $X_1, \ldots, X_m$, we now have that $\Pr(E_\phi = \mathit{true} \,|\, \mathbf{x}) < \frac{1}{2^{n-m}}$; more specifically, for the jth lexicographically lowest ordered joint value assignment $\mathbf{x}$ to $X_1, \ldots, X_m$, we have $\Pr(E_\phi = \mathit{true} \,|\, \mathbf{x}) = \frac{j-1}{2^{n+1}}$.

To conclude the construction, we add to the network a node C with V_ϕ and E_ϕ for its parents, with the following conditional probability distributions:

$$\Pr(C = \mathit{true} \,|\, V_\phi, E_\phi) = \begin{cases} 1 & \text{if } V_\phi = \mathit{true}, E_\phi = \mathit{true} \\ \frac{1}{2} & \text{if } V_\phi = \mathit{true}, E_\phi = \mathit{false} \\ \frac{1}{2} & \text{if } V_\phi = \mathit{false}, E_\phi = \mathit{true} \\ 0 & \text{if } V_\phi = \mathit{false}, E_\phi = \mathit{false} \end{cases}$$

Note that since $\Pr(E_\phi = \mathit{true} \,|\, \mathbf{x}) < \frac{1}{2^{n-m}}$ for any joint value assignment $\mathbf{x}$ to the explanation nodes $X_1, \ldots, X_m$, these probabilities ensure that the posterior probability $\Pr(C = \mathit{true} \,|\, \mathbf{x})$ lies within the interval $[\frac{s}{2^{n-m+1}}, \frac{s+1}{2^{n-m+1}}]$, where s is the number of value assignments to the Boolean variables $\{X_{m+1}, \ldots, X_n\}$ that, jointly with $\mathbf{x}$, satisfy the formula ϕ. Figure 2 shows the graphical structure of the probabilistic network that is thus constructed from the example formula $\phi_{\rm ex}$.

Theorem 2. Bounded Kth Partial MAP *is* $\mathsf{FP}^{\mathsf{PP}^{\mathsf{PP}}}$*-complete.*

Proof. The membership proof for the Bounded Kth Partial MAP problem is quite similar to the membership proof for the Kth MPE problem from Theorem 1, that is, we construct a metric Turing transducer to solve the problem. Note that for the complexity class $\mathsf{FP}^{\mathsf{PP}^{\mathsf{PP}}}$ we are now allowed to consult a more powerful oracle than for the class $\mathsf{FP}^{\mathsf{PP}}$. We observe that for the Bounded Kth

Partial MAP problem, we actually *need* an oracle of higher power, since we need to solve the #P-complete problem of Exact Inference to compute the required joint probabilities: while for the Kth MPE problem we could efficiently compute probabilities for joint value assignments to all variables taking polynomial time, we must now compute probabilities of joint value assignments to a subset of the variables, which involves summing over all assignments to the intermediate variables involved. Now, if the probability $\Pr(\mathbf{m}, \mathbf{e})$ obtained for a joint value assignment $\mathbf{m}$ to the explanation variables $\mathbf{M}$ is within the interval $[a, b]$, the transducer outputs a binary representation of $1 - \Pr(\mathbf{m}, \mathbf{e})$ along with the logical inverse of the binary representation of $\mathbf{m}$; otherwise, it outputs $\perp$. Clearly, $\text{KthValue}_{\hat{\mathcal{T}}}$ returns an encoding of the kth most probable value assignment to the explanation variables in view of the evidence $\mathbf{e}$. We conclude that Bounded Kth Partial MAP is in $\mathsf{FP}^{\mathsf{PP}^{\mathsf{PP}}}$.

To prove hardness for the class $\mathsf{FP}^{\mathsf{PP}^{\mathsf{PP}}}$, we reduce the KthNumSat problem to Bounded Kth Partial MAP. Let (ϕ, k, l) be an instance of KthNumSat. From the Boolean formula ϕ we construct a probabilistic network $\mathcal{B}_\phi$ as described above; we further let $\mathbf{E} = \{C\}$ and let $\mathbf{e}$ be the value assignment $C = \mathit{true}$. The conditional probability distributions of the constructed network ensure that the posterior probability $\Pr(C = \mathit{true} \mid \mathbf{x})$ for a joint value assignment $\mathbf{x}$ to the nodes $\{X_1, \ldots, X_m\}$ with which exactly l truth value assignments to $\{X_{m+1}, \ldots, X_n\}$ satisfy ϕ, lies within the interval $[\frac{l}{2^{n-m+1}}, \frac{l+1}{2^{n-m+1}}]$. Moreover, if two assignments $\mathbf{x}$ and $\mathbf{x}'$ both are such that exactly l truth value assignments to $\{X_{m+1}, \ldots, X_n\}$ serve to satisfy ϕ, then $\Pr(C = \mathit{true} \mid \mathbf{x}) > \Pr(C = \mathit{true} \mid \mathbf{x}')$ if $\mathbf{x}$ is ordered before $\mathbf{x}'$ in descending lexicographic order. For the constructed instance of the Bounded Kth Partial MAP problem, we thus have that the kth most probable joint value assignment to the explanation nodes $X_1, \ldots, X_m$ corresponds with the lexicographically kth truth value assignment to the Boolean variables $\{X_1, \ldots, X_m\}$ with which exactly l assignments to $\{X_{m+1}, \ldots, X_n\}$ satisfy ϕ. Clearly, the above reduction is a polynomial-time one-Turing reduction from KthNumSat to Kth Partial MAP. Given an algorithm for solving Bounded Kth Partial MAP, we can thus solve the KthNumSat problem as well, which proves $\mathsf{FP}^{\mathsf{PP}^{\mathsf{PP}}}$-hardness of Bounded Kth Partial MAP. □

$\mathsf{FP}^{\mathsf{NP}^{\mathsf{PP}}}$-completeness of Bounded Partial MAP, which is the special case of Bounded Kth Partial MAP with $k = 1$, now follows by a very similar proof.

Proposition 2. Bounded Partial MAP *is* $\mathsf{FP}^{\mathsf{NP}^{\mathsf{PP}}}$*-complete.*

5 Conclusion

We addressed the computational complexity of two problems that arise in practical applications of probabilistic networks. Informally spoken, these problems ask for the kth most likely explanation for a given collection of observations in a network. For the Kth MPE problem, an explanation is defined as a joint value assignment to all non-observed variables; we showed that this problem is $\mathsf{FP}^{\mathsf{PP}}$-complete. For the Kth Partial MAP problem, a designated subset of non-observed variables is distinguished; these variables are taken as the explanation variables for which a joint value assignment is being sought. We showed that this particular problem is $\mathsf{FP}^{\mathsf{PP}^{\mathsf{PP}}}$-complete. In the future, we would like to further study the complexity of the two problems when the constant k is bounded by a polynomial function in the number of variables involved.

By our results we pinpointed the precise complexity of two practical problems, although it is fair to mention that from a practitioners' point of view knowing NP-hardness would have sufficed. Interesting from a theoretical point of view, however, is the observation that our complexity results are among the very few showing practically relevant problems to be complete for complexity classes that are as special as $\mathsf{FP}^{\mathsf{PP}}$and $\mathsf{FP}^{\mathsf{PP}^{\mathsf{PP}}}$.

References

1. Charitos, T., van der Gaag, L.C., Visscher, S., Schurink, C.A.M., Lucas, P.J.F.: A dynamic Bayesian network for diagnosing ventilator-associated pneumonia in ICU patients. Expert Systems with Applications 26, 1249–1258 (2009)
2. Geenen, P.L., Elbers, A.R.W., van der Gaag, L.C., Loeffen, W.L.A.: Development of a probabilistic network for clinical detection of classical swine fever. In: Proceedings of the 11th Symposium of the International Society for Veterinary Epidemiology and Economics, Cairns, Australia, pp. 667–669 (2006)
3. Pearl, J.: Probabilistic Reasoning in Intelligent Systems. Morgan Kaufmann, Palo Alto (1988)
4. Roth, D.: On the hardness of approximate reasoning. Artificial Intelligence 82, 273–302 (1996)
5. Shimony, S.E.: Finding MAPs for belief networks is NP-hard. Artificial Intelligence 68, 399–410 (1994)
6. Bodlaender, H.L., van den Eijkhof, F., van der Gaag, L.C.: On the complexity of the MPA problem in probabilistic networks. In: van Harmelen, F. (ed.) Proceedings of the 15th European Conference on Artificial Intelligence, pp. 675–679. IOS Press, Amsterdam (2002)
7. Park, J.D., Darwiche, A.: Complexity results and approximation settings for MAP explanations. Journal of Artificial Intelligence Research 21, 101–133 (2004)
8. Charniak, E., Shimony, S.E.: Cost-based abduction and MAP explanation. Acta Informatica 66, 345–374 (1994)
9. Abdelbar, A., Hedetniemi, S.: Approximating MAPs for belief networks is NP-hard and other theorems. Artificial Intelligence 102, 21–38 (1998)
10. Jensen, F.V., Nielsen, T.D.: Bayesian Networks and Decision Graphs. Springer, New York (2007)
11. Papadimitriou, C.H.: On the complexity of unique solutions. Journal of the ACM 31, 392–400 (1984)
12. Krentel, M.W.: The complexity of optimization problems. Journal of Computer and System Sciences 36, 490–509 (1988)
13. Toda, S.: Simple characterizations of P(#P) and complete problems. Journal of Computer and System Sciences 49, 1–17 (1994)
14. Kwisthout, J.H.P.: The Computational Complexity of Probabilistic Networks. PhD thesis, Universiteit Utrecht (2009)
15. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley, Reading (1994)

Optimal File-Distribution in Heterogeneous and Asymmetric Storage Networks

Tobias Langner[1], Christian Schindelhauer[2], and Alexander Souza[3]

[1] Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Switzerland
tobias.langner@tik.ee.ethz.ch
[2] Institut für Informatik, Universität Freiburg, Germany
schindel@informatik.uni-freiburg.de
[3] Institut für Informatik, Humboldt Universität zu Berlin, Germany
souza@informatik.hu-berlin.de

Abstract. We consider an optimisation problem which is motivated from storage virtualisation in the Internet. While storage networks make use of dedicated hardware to provide homogeneous bandwidth between servers and clients, in the Internet, connections between storage servers and clients are heterogeneous and often asymmetric with respect to upload and download. Thus, for a large file, the question arises how it should be fragmented and distributed among the servers to grant "optimal" access to the contents. We concentrate on the transfer time of a file, which is the time needed for one upload and a sequence of n downloads, using a set of m servers with heterogeneous bandwidths. We assume that fragments of the file can be transferred in parallel to and from multiple servers. This model yields a distribution problem that examines the question of how these fragments should be distributed onto those servers in order to minimise the transfer time. We present an algorithm, called FlowScaling, that finds an optimal solution within running time $\mathcal{O}(m \log m)$. We formulate the distribution problem as a maximum flow problem, which involves a function that states whether a solution with a given transfer time bound exists. This function is then used with a scaling argument to determine an optimal solution within the claimed time complexity.

Keywords: distributed storage, distribution problem, asymmetric bandwidths, distributed file system, flow scaling.

Categories: D.4.3 (Distributed file systems), F.2.2 (Computations on discrete structures) G.1.6 (Linear programming), G.2.1 (Combinatorial algorithms).

1 Introduction

This paper deals with the optimal distribution of fragments of a file across a potentially heterogeneous network of file servers. Our objective is to minimise the transfer time of a file, that is the time for one upload and a sequence of downloads. This goal and the underlying model are motivated from the concept of storage virtualisation in data centres, where the accumulated hard disk memory of multiple machines is to be provided as one large storage unit. Our paper transfers these ideas from the homogeneous setting in data centres to the heterogeneous situation in the Internet with highly differing bandwidth setups of the individual machines.

In order to provide storage virtualisation, data centres have to implement a *distributed file system* which has to provide file system operations like read, write, delete, etc. but, in contrast to a disk-based file system, can distribute files across multiple machines. For example, if a user, called *client*, wants to store a file in the file system, one way of doing so on the implementation side is that the data centre (the *virtual server*) stores the file as a whole on one of its servers. Another way is that the file be split into fragments and these are stored on different servers. This enables *parallel* up- and *parallel* download of the file, which can provide significant speed-up and hence is of particular interest for large files such as movies.

However, if connected to the Internet, the bandwidths of the individual users usually vary significantly. Furthermore, another – so far neglected – aspect comes into play: the asymmetry of end-users' Internet connections. A typical DSL or T1 connection provides significantly higher download bandwidths compared to the upload. When files are to be distributed through the Internet, and furthermore, specific quality of service requirements (QoS) have to be met (such as transfer time and throughput), the heterogeneity of the bandwidths certainly has to be taken into consideration.

The QoS parameter of interest for our work is the transfer time: the time for one file upload and multiple subsequent downloads as defined below. Hence, we consider the question of how to distribute fragments of a file *optimally* in a potentially heterogeneous network in a manner that minimises this transfer time.

The above problem contains several practical challenges, but due to the theoretic nature of our contribution, we shall abstract from most of them and focus on the core of the problem. For example, we ignore the fact that for some users, the actual bottleneck might not be the bandwidth of the accessed servers but rather the own Internet connection. This extended problem will be dealt with in a follow-up paper.

Related Work. In the seminal paper of Patterson et al. the case for redundant arrays of inexpensive disks was made [1]. Since then, the development of distributed file systems has lead to RAID-systems for small enterprises and households, storage area networks, and file systems distributed over the Internet.

Ghewamat et al. introduced a highly scalable distributed file system called Google File System (GFS) [2]. Being required to provide high data-throughput rates, a large number (> 1000) of inexpensive commodity computers (chunk servers) and a dedicated master server are connected by a high-speed network to form a GFS Cluster. GFS is optimised for storing large files (> 100 MB) by splitting them up into chunks of a fixed size (e.g. 64 MB) and storing multiple copies of each chunk independently. Upon a file-retrieval request, the master server is asked and responds the locations of the replicas to the client. The client then contacts *one* of the chunk servers and receives the desired chunk from this single server, yielding that *no* parallel data transfer is used.

Another distributed file system is Dynamo which was developed by DeCandia et al. for Amazon [3]. In contrast to GFS, it does not provide a directory structure but implements a simple key-value structure, only. It also does not have a central master server, but is completely decentralised by using consistent hashing to assign key-value-pairs to the servers. Dynamo is able to cope with heterogeneous servers by accounting for the storage capacity of each server in the hash-function. Similar to GFS, different connection speeds are not taken into account since all servers are (assumed to be) connected through the same network.

Various other distributed file systems are based on peer-to-peer networks. However, they concentrated on the reputation based election of storage devices [4], were optimised for high reliability of the stored files [5] or meant to be applied in malicious environments where servers might be managed by adversaries and thus cannot be trusted [6].

The assumption that the underlying network infrastructure is homogeneous is questionable. Firstly, data centre configurations "as such" evolve over time (i.e. not all components are replaced at once). Secondly and more importantly, in an Internet-based service, client connections and also the upload and download bandwidth of a specific connection can be significantly different. Albeit being a natural setting, only relatively few results on the heterogeneous variant of the distribution problem are available. To the best of our knowledge, no work on the asymmetric case exists.

One of the approaches for heterogeneous distributed storage is the Distributed Parallel File System (DPFS) by Shen and Choudhary [7]. When a file is to be stored, it is broken up into fragments called bricks that are then distributed across a network of participating storage servers, where each server receives a portion of the file. Of course, the striping-strategy affects the performance of the system, as it yields a distribution of bricks on servers. To account for the heterogeneous bandwidths of the involved servers, Shen and Choudhary proposed a greedy algorithm that prefers fast servers over slow servers according to a pre-calculated performance factor. Thus, if a server is k times as fast as another one, it is also allotted k times as many bricks.

Karger et al. introduced the concept of Distributed Hash Tables (DHT) [8] that are used to balance the network load on multiple mirror servers. Every available server receives a random value in the interval $[0, 1]$ using a hash-function. When a clients wants to access a server, it is also hashed into the interval $[0, 1]$ and then served by the "closest" server.

The idea of DHT was extended by Schindelhauer and Schomaker's Distributed Heterogeneous Hash Tables (DHHT) [9]. Instead of clients, the documents to be stored are projected into $[0, 1]$ using a hash function and are assigned to the "closest" server. The two hash functions for hashing the servers and the clients then account for the heterogeneity of the network (in terms of server capacity and connection bandwidth).

DHHT was further extended by its authors by the the notion of *transfer time* [10]: Let a matrix $(A_{d,i})$ describe the amount of document d being stored on server i, respectively. The bandwidth of server i is given by a quantity b_i. Then two different notions of transfer time of a document d are introduced. The sequential time $\mathrm{SeqTime}_A(d) = \sum_i A_{d,i}/b_i$ and the parallel time $\mathrm{ParTime}_A(d) = \max_i A_{d,i}/b_i$. Using popularity values assigned to the documents, the problems of minimising the average as well as the worst-case time of both measures can be solved optimally with linear programming. Additionally, closed solutions for minimising the average sequential and parallel time were given. In their model the time for uploading the data has not been considered at all, which is a major concern of our paper.

Our Contribution. Based on work of Langner [11], we introduce a model, which is motivated from the transfer time model, but also accounts for the following situation observed frequently in practise: Large files, e.g. movies, are usually uploaded once and remain unchanged for a substantial amount of time. In the meantime, they are downloaded many times. Moreover, our model covers asymmetric up- and download bandwidths and hence captures the actual technological situation contrasting the DHHT- and DPFS-models.

We consider the Distribution Problem where a file f is split into m disjoint fragments that are then uploaded in parallel to m servers (with possibly heterogeneous and asymmetric bandwidths) – one fragment per server. Our objective is to find the *optimal* partition of f that minimises the transfer time. This time is the total time it takes to complete a sequence of a parallel upload of the file and n subsequent parallel downloads. The parameter n is introduced in order to reflect the typical "one-to-many" situation mentioned above. Our main result is:

Theorem 1. *Algorithm* FlowScaling *solves the* Distribution Problem *optimally in time* $\mathcal{O}(m \log m)$.

Our approach is to formulate the problem as a linear program, which turns out to be related to a maximum flow problem. This already yields a strongly polynomial optimal algorithm. The central question is thus how to improve the running time. We define a function which states whether a solution with a given transfer time bound exists or not. This specific maximum flow problem is then solved using a scaling argument (explained later), yielding the claimed $\mathcal{O}(m \log m)$ time complexity.

2 Preliminaries

We are given a *distribution network* $\mathcal{N}$, i.e. a weighted directed graph $G = (V, A)$ where nodes represent computers and edges network connections between them. Each edge has a positive weight $b : A \mapsto \mathbb{R}^+$ indicating the bandwidth of the respective connection. The node set $V := S \cup \{c\}$ with $S = \{s_1, \ldots, s_m\}$ contains the client node c and m server nodes in S. The edge set $A := U \cup D$ consists of the upload edges in $U = \{(c, s_i) : i \in S\}$ and the download edges in $D = \{(s_i, c) : i \in S\}$. When convenient, we use a server s_i and its index i interchangeably.

The definition above implies that both in-degree and out-degree of the server nodes are one. The in- and out-degree of the client node c is m, consequently. Each server s_i has a pair of upload and download bandwidths which we denote as u_i and d_i. A graphical illustration of the above definition is given in Figure 1.

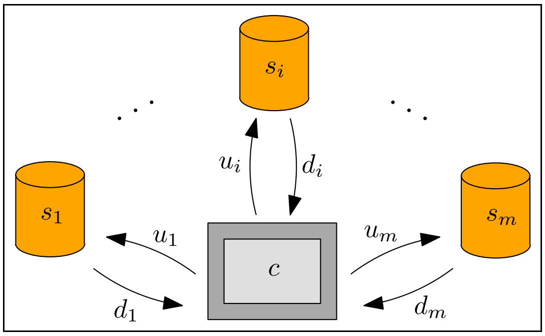


Fig. 1. Illustration of a distribution network. The upload bandwidths u_i and download bandwidths d_i denote the speed of the respective connection between client c and server s_i.

The Distribution Problem poses the question how we have to split a file f with size $|f|$ into fragments $x_1, \ldots, x_m$ for the m servers in a distribution network such that the time for one upload to the servers and n subsequent downloads from the servers is minimised. Since we want to be able to recover f from the fragments x_i, we require $\sum_{i=1}^{m} x_i = |f|$. We assume that the number of downloads n is known (or can be estimated) beforehand. Of course, the model would be more realistic, if n need not be known. However, we justify our assumption as follows: If the number n of downloads is unknown, then the optimal transfer time can not be approximated by any algorithm, see Lemma 1.

For a given *distribution* $\mathbf{x} = (x_1, \ldots, x_m)$, we define the upload time $t_u(\mathbf{x})$ and download time $t_d(\mathbf{x})$ as the maximal upload/download time over all individual servers, i.e.

$$t_u(\mathbf{x}) = \max_{i \in S}\left\{\frac{x_i}{u_i}\right\} \qquad \text{and} \qquad t_d(\mathbf{x}) = \max_{i \in S}\left\{\frac{x_i}{d_i}\right\} \ .$$

The objective value $\mathrm{val}(\mathbf{x})$ is the upload time of $\mathbf{x}$ plus n times the download time,

$$\mathrm{val}(\mathbf{x}) = t_u(\mathbf{x}) + n \cdot t_d(\mathbf{x}) \ .$$

Realistically, the exact number of downloads n is not known beforehand. This induces an online problem, where the value of n is learned gradually. Thus an online algorithm has to find a file-distribution, which is "good" for all values of n or adapt the current distribution on-the-fly when another download occurs. More precisely, we say that an algorithm for the online problem is *c-competitive*, if its transfer time is at most c times larger than the optimal transfer time (with n known beforehand). Lemma 1 states that there does not exist a c-competitive algorithm for any constant c.

Lemma 1. *There is no c-competitive algorithm for the online* Distribution Problem *for any constant $c \geq 1$.*

Proof. Let I be an instance of a Distribution Problem with two servers s_1 and s_2 and bandwidths $u_1 = a$, $d_1 = \frac{1}{a}$, $u_2 = 1$, and $d_2 = 1$ where $a \gg 1$. If there is only an initial upload (and no download), the optimal solution is $\mathbf{x}_{\text{OPT},0} = \left(\frac{a}{a+1}, \frac{1}{a+1}\right)$ with $\text{val}(\mathbf{x}_{\text{OPT},0}) = \frac{1}{a+1}$. An arbitrary c-competitive online algorithm ALG is allowed to upload at most $c \cdot \frac{1}{a+1}$ data units to s_2, as then the upload already takes time $c \cdot \text{val}(\mathbf{x}_{\text{OPT},0})$. Consequently, in $\mathbf{x}_{\text{ALG}}$ at least $\frac{a+1-c}{a+1}$ data units have to be uploaded to s_1 and we have

$$t_d(\mathbf{x}_{\text{ALG}}) \geq \frac{\frac{a+1-c}{a+1}}{\frac{1}{a}} = \frac{a(a+1-c)}{a+1} \ .$$

If one download occurs after the upload, the optimal solution is given by $\mathbf{x}_{\text{OPT},1} = \left(\frac{1}{a+1}, \frac{a}{a+1}\right)$ with $\text{val}(\mathbf{x}_{\text{OPT},1}) = 2 \cdot \frac{a}{a+1}$. For the ratio of the objective values of both solutions, we get

$$\frac{\text{val}(\mathbf{x}_{\text{ALG}})}{\text{val}(\mathbf{x}_{\text{OPT},1})} > \frac{t_d(\mathbf{x}_{\text{ALG}})}{\text{val}(\mathbf{x}_{\text{OPT},1})} \geq \frac{a+1-c}{2}$$

which means that ALG is not c-competitive, since a can be arbitrarily large. □

If we assume the number $n > 0$ of downloads to be known beforehand, then we can safely set $n = 1$, $|f| = 1$ and solve the resulting simplified problem instead, Lemma 2 below. Thus, in the sequel, we will consider the simplified version only.

Lemma 2. *Let $I = (\mathcal{N}, |f|, n)$ with $n > 0$ and $I' = (\mathcal{N}', 1, 1)$ be the* Distribution Problem *instances where $\mathcal{N}$ equals $\mathcal{N}'$ with the modification that for the download edge weights in $\mathcal{N}'$ we have $d'_i = d_i/n$. If $\mathbf{x}'$ is an optimal solution for I', then $\mathbf{x} = |f| \cdot \mathbf{x}'$ is optimal for I.*

Proof. For the objective value of the solution $\mathbf{x}$ within instance I, we have

$$\begin{aligned}
\text{val}_I(\mathbf{x}) &= t_u(\mathbf{x}) + n \cdot t_d(\mathbf{x}) \\
&= \max_{i \in S}\left\{\frac{x_i}{u_i}\right\} + n \cdot \max_{i \in S}\left\{\frac{x_i}{d_i}\right\} \\
&= \max_{i \in S}\left\{\frac{x_i}{u_i}\right\} + \max_{i \in S}\left\{\frac{x_i}{\frac{d_i}{n}}\right\} \\
&= \max_{i \in S}\left\{\frac{x'_i \cdot |f|}{u_i}\right\} + \max_{i \in S}\left\{\frac{x'_i \cdot |f|}{\frac{d_i}{n}}\right\} \\
&= |f| \cdot \left(\max_{i \in S}\left\{\frac{x'_i}{u'_i}\right\} + \max_{i \in S}\left\{\frac{x'_i}{d'_i}\right\}\right) \\
&= |f| \cdot \text{val}_{I'}(\mathbf{x}') \ .
\end{aligned}$$

Consequently, if $\mathbf{x}'$ is minimal for I', we have that $|f| \cdot \mathbf{x}'$ is minimal for I. □

The Distribution Problem can be formulated as a linear program by introducing two variables t_u and t_d for the upload and download time of the solution $\mathbf{x}$.

$$\begin{array}{lll} \text{minimise} & t_u + t_d & \\ \text{subject to} & \displaystyle\sum_{i=1}^{m} x_i = 1 & \\ & \dfrac{x_i}{u_i} \leq t_u & \text{for all } i \in \{1, \ldots, m\} \\ & \dfrac{x_i}{d_i} \leq t_d & \text{for all } i \in \{1, \ldots, m\} \\ & x_i \geq 0 & \text{for all } i \in \{1, \ldots, m\} \end{array}$$

This already yields that the optimal solution for the Distribution Problem can be found in polynomial time. In the next section, however, we shall prove Theorem 1 by showing that the time complexity $\mathcal{O}(m \log m)$ suffices.

3 Flow Scaling

We transfer an idea from Hochbaum and Shmoys [12] which yielded a polynomial approximation scheme for Makespan Scheduling. In that problem, we are given a set of identical processors and have to schedule a set of jobs, where the objective is to minimise the maximum completion time, i.e. the makespan. The principal approach is to "guess" the optimal makespan and then find a schedule that does not violate this makespan by "too much". The optimal makespan is actually determined by using binary search.

Distribution- and Flow-Problems. We employ a similar approach (but are able to avoid the binary search step). We assume that the total time $T = t_u + t_d$ and the upload time t_u are given. (Thus we obviously also have the download time $t_d = T - t_u$.) Then we can formulate a Distribution Problem instance I as a Maximum Flow instance G_I as given in Figure 2: We have a source s, a sink t, and m intermediate vertices $s_1, \ldots, s_m$. Source and sink correspond to the client in Figure 1, and the s_i correspond to the servers. For $i = 1, \ldots, m$, the upload edge (s, s_i) of server s_i has capacity $t_u \cdot u_i$. This is the maximum amount of data that can be transferred to this server in time t_u. Similarly, for $i = 1, \ldots, m$, the download edge (s_i, t) of server s_i has capacity $t_d \cdot d_i$ because this is the maximum amount of data that can be transferred from this server to the sink node t in time t_d.

The Maximum Flow formulation allows us to decide if it is possible to transfer (i.e. upload and download) a file f (w.l.o.g. having size $|f| = 1$) in time T. For this purpose, we define a function

$$f_{T,i}(t) = \min\{t \cdot u_i, (T - t) \cdot d_i\}$$

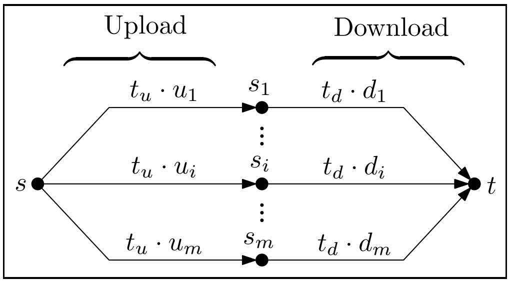


Fig. 2. Flow network G_I for given upload/download time t_u, t_d, and distribution problem instance I

for $i = 1, \ldots, m$ and $t \in [0, T]$. Notice that $f_{T,i}(t)$ equals the maximum value of an $s - s_i - t$-flow, when the upload time is $t_u = t$ and the download time hence $t_d = T - t$. Using these functions, we define the *total data function* by

$$\delta_T(t) = \sum_{i=1}^{m} f_{T,i}(t) \ . \tag{1}$$

Lemma 3. *An instance $I = (\mathcal{N}, 1, 1)$ of the* Distribution Problem *admits a feasible solution* $\mathbf{x}$ *with transfer time $T = t_u + t_d$ if and only if*

$$\delta_T(t) \geq 1 \quad \text{for some } t \in [0, T] \ . \tag{2}$$

Proof. The famous Max-Flow-Min-Cut theorem tells us that the maximal amount of data that can be uploaded to the servers and download again afterwards in an instance I is given by the capacity of a minimum cut in the flow network G_I defined above, see Figure 2.

Consequently, there exists a solution for a distribution problem instance I with upload time $t_u = t$ and download time $t_d = T - t$ if and only if the capacity of the minimum cut in the distribution flow graph G_I is at least 1, i.e. the file size. Observe that the function $\delta_T(t)$ equals the minimum cut capacity. □

Hence, we can use Equation 2 as the decision criterion that tells us whether there exists a distribution satisfying given upload and download times t_u and $t_d = T - t_u$.

For each summand $f_{T,i}$ of δ_T, we define the value $p_{T,i}$ by $p_{T,i} \cdot u_i = (T - p_{T,i}) \cdot d_i$, where its maximum is attained. That is, for $i = 1, \ldots, m$ we have

$$p_{T,i} = T \cdot \frac{d_i}{u_i + d_i} \ .$$

For convenience, we define $p_{T,0} = 0$ and $p_{T,m+1} = T$. Then, for $i = 0, \ldots, m+1$ we define

$$\delta_{T,i} = \delta_T(p_{T,i}) \ .$$

For simplicity of notation, we shall write p_i and δ_i instead of $p_{T,i}$ and $\delta_{T,i}$, respectively, in the sequel when appropriate.

The function δ_T is concave and piecewise linear in the interval $[0, T]$, but not differentiable at the p_i defined above. The points (p_i, δ_i) for $i = 0, \ldots, m+1$ are called the *vertices* of δ_T in the sequel. Figure 3 depicts an illustration of a sample total data function along with its summands. To determine whether

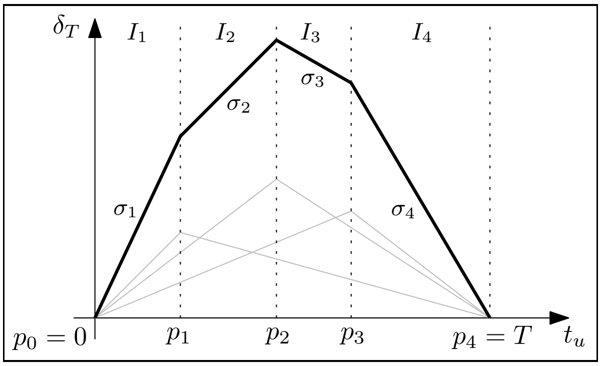


Fig. 3. The graph shows a sample total data function for $m = 3$ along with the three summands it comprises and the intervals I_1 to I_4 induced by its vertices

a distribution with total time T exists for a given DISTRIBUTION PROBLEM instance, we only have to check whether its maximum value is at least 1. If we implement this by evaluating the δ_i-values naïvely, we arrive at $\mathcal{O}(m^2)$ running time. Thus we have to be careful in evaluating this function.

Lemma 4. *The algorithm* EVALUATETOTALDATAFUNCTION *computes the vertices* (p_i, δ_i) *for* $i = 1, \ldots, m$ *in time* $\mathcal{O}(m \log m)$*.*

Proof. The algorithm works as follows: It renumbers the servers such that for two servers s_i and s_j we have $i < j$ if $p_i < p_j$. This takes time $\mathcal{O}(m \log m)$.

Then the total data function is made up of $m + 1$ linear segments in the intervals I_1 to I_{m+1} where $I_j = [p_{j-1}, p_j]$. Let σ_j denote the slope of δ_T in the interval I_j. Recalling the formulation of the total data function in Equation 1 we infer that the slope σ_1 of δ_T in $[p_0, p_1]$ is given by

$$\sigma_1 = \sum_{i=1}^{m} u_i \ .$$

For the slopes in the other intervals the following recursion formula holds (see Figure 3):

$$\sigma_i = \sigma_{i-1} - u_i - d_i \quad \text{for } i \in \{2, \ldots, m+1\} \ .$$

Using this observation, we get a recursive formula for the value of $\delta_T(p_i)$.

$$\delta_T(p_i) = \begin{cases} \sigma_1 \cdot p_1 & \text{if i = 1} \\ \delta_T(p_{i-1}) + \sigma_i \cdot (p_i - p_{i-1}) & \text{if } i > 1 \end{cases} \tag{3}$$

This formula can be evaluated for the positions p_i efficiently in $\mathcal{O}(m)$ steps with a simple scan-line approach as implemented in Algorithm 1. The critical step determining the overall run-time is the renumbering in line 4 that involves sorting the servers according to their value of p_i and thereby incurs a run-time cost of $\mathcal{O}(m \log m)$. □

Algorithm 1. EVALUATETOTALDATAFUNCTION($\mathcal{N}, T$)

Input. A distribution network $\mathcal{N}$ and a total time bound T.

Output. A tuple of two m-dimensional vectors $\boldsymbol{p}$ and $\boldsymbol{\delta}$ where δ_i is the total data function value at p_i.

```
 1: for $i \leftarrow 1$ to $m$ do
 2:     $p_i \leftarrow T \cdot \frac{d_i}{u_i + d_i}$                    ▷ Calculate the positions $p_i$
 3: end for
 4: renumber($\mathbf{p}$, $p_i < p_j$ for $i < j$)      ▷ Renumber servers according to $p$-values
 5: $\sigma \leftarrow \sum_{i=1}^{m} u_i$
 6: $\delta_0 = 0$
 7: for $i = 1$ to $m$ do
 8:     $\delta_i \leftarrow \delta_{i-1} + \sigma \cdot (p_i - p_{i-1})$
 9:     $\sigma \leftarrow \sigma - u_i - d_i$
10: end for
11: return ($\boldsymbol{p}, \boldsymbol{\delta}$)
```

It is straightforward how Algorithm 1 can be modified in order to check whether we have $\delta_T(p_i) \geq 1$ for some p_i and return the respective p_i. According to the deliberations above, $\delta_T(p_i) \geq 1$ implies that there is a solution to the DISTRIBUTION PROBLEM with the time bounds $t_u = p_i$ and $t_d = T - p_i$. So what remains to be shown is how we can find the actual distribution vector $\mathbf{x} = (x_1, \ldots, x_m)$ for a given pair (t_u, t_d) specifying how the file f should be distributed among the m servers. This is accomplished with Algorithm 2.

Lemma 5. *For given values of t_u and $t_d = T - t_u$ with $\delta_T(t_u) \geq 1$, algorithm* CALCULATEDISTRIBUTION *computes a feasible distribution $\mathbf{x}$ in time $\mathcal{O}(m)$.*

Proof. Algorithm 2 iterates through all m servers and assigns to each server the maximum amount of data that can be uploaded to it in time t_u and downloaded from it in time t_d. More formally, the amount $f_{T,i}(t_u) = \min\{t_u \cdot u_i, t_d \cdot d_i\}$ (or less if the file size is already almost exhausted) is assigned to server s_i. Recalling Equation 2, a valid solution with upload time t_u and download time $t_d = T - t_u$ exists if the inequality

$$\delta_T(t_u) = \sum_{i=1}^{m} f_{T,i}(t_u) \geq 1$$

is satisfied. If so, then Algorithm 2 terminates with a valid distribution for t_u and t_d within running time $\mathcal{O}(m)$. □

Algorithm 2. CalculateDistribution$(\mathcal{N}, t_u, t_d)$

Input. A distribution network $\mathcal{N}$ and target upload and download times t_u and t_d.

Output. A distribution $\mathbf{x} = (x_1, \ldots, x_m)$ obeying the target times.

```
1: $f \leftarrow 1,\ i \leftarrow 1$
2: $\mathbf{x} \leftarrow (0, \ldots, 0)$          ▷ initialise $\mathbf{x}$ to be a zero vector of dimension $m$
3: while $f > 0$ do
4:     $x_i \leftarrow \min\{f,\ \min\{t_u \cdot u_i,\ t_d \cdot d_i\}\}$
5:     $f \leftarrow f - x_i$
6:     $i \leftarrow i + 1$
7: end while
8: return $\mathbf{x}$
```

Scaling. Using Algorithm 1 and 2, we can – for a fixed total time T – determine whether a solution exists and if so, calculate it. However, it still remains to show how we can find the minimal total time which still allows for a solution. One way of doing so is binary search. However, we can do better by having a closer look at δ_T yielding a scaling property.

Lemma 6. *For $t \in [0, 1]$ and $T > 0$ we have*

$$\delta_T(p_{T,i}) = T \cdot \delta_1(p_{1,i}) \ .$$

Proof. Recall that we have $p_{T,i} = T \cdot d_i/(u_i + d_i) = T \cdot p_{1,i}$ and hence for any $j = 1, \ldots, m$

$$\begin{aligned} f_{T,j}(p_{T,i}) &= \min\left\{T \cdot \frac{u_j d_i}{u_i + d_i}, \left(T - T \cdot \frac{d_i}{u_i + d_i}\right) \cdot d_j\right\} \\ &= T \cdot \min\left\{\frac{u_j d_i}{u_i + d_i}, \frac{u_i d_j}{u_i + d_i}\right\} = T \cdot f_{1,j}(p_{1,i}) \ . \end{aligned}$$

Using $p_{T,i} = T \cdot p_{1,i}$ and $f_{T,j}(p_{T,i}) = T \cdot f_{1,j}(p_{1,i})$ yields

$$\delta_T(p_{T,i}) = \sum_{j=1}^{m} f_{T,j}(p_{T,i}) = T \cdot \sum_{j=1}^{m} f_{1,j}(p_{1,i}) = T \cdot \delta_1(p_{1,i}) \ .$$

Thus, as the vertices of the piecewise linear function scale, the whole function scales. □

The geometric structure of the total data function δ_T is illustrated in Figure 4 where we can clearly see the straight edges of the polytope that represent the points $(p_{T,i}, T, \delta_{T,i})$ for varying values of T.

Previously we have shown how the total data function δ_T for a fixed value of T can be evaluated at the positions $p_{T,i}$ in time $\mathcal{O}(m \log m)$ using Algorithm 1.

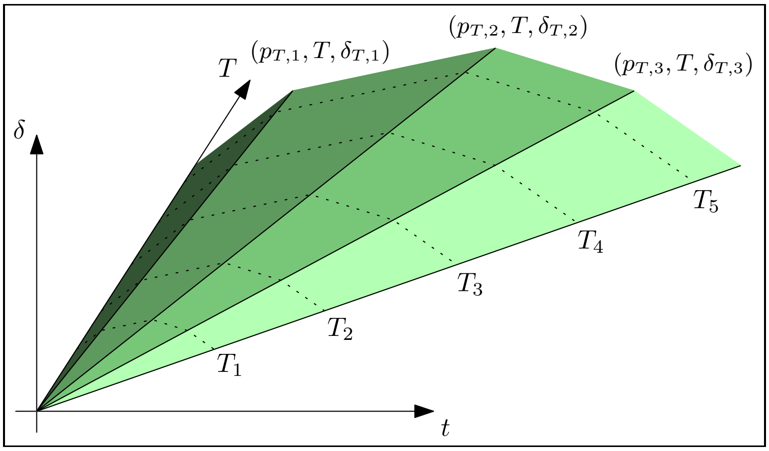

T
$(p_{T,1}, T, \delta_{T,1})$
$(p_{T,2}, T, \delta_{T,2})$
$(p_{T,3}, T, \delta_{T,3})$
δ
t
T_1
T_2
T_3
T_4
T_5

Fig. 4. The total data function δ_T as a two-variable function of upload time t and total time T

Now we will show that evaluating $\delta_1(t)$ and then using the scaling property given in Lemma 6 yields an algorithm for computing the optimal total time T.

Lemma 7. *Algorithm* FlowScaling *computes an optimal solution* **x** *for an instance* $I = (\mathcal{N}, 1, 1)$ *of the* Distribution Problem.

Proof. The algorithm FlowScaling evaluates the total data function using Algorithm 1 for $T = 1$ and obtains coordinate pairs (p_i, δ_i) for $i = 1, \ldots, m$. Let (p_k, δ_k) be a pair, where δ_k is maximum among all δ_i. We now choose

$$T = \frac{1}{\delta_k} \ ,$$

and

$$t_u = p_{T,k} = T \cdot p_{1,k} = T \cdot p_k = \frac{p_k}{\delta_k}$$

to obtain

$$\delta_T(t_u) = \delta_T(p_{T,k}) = T \cdot \delta_1(p_{1,k}) = T \cdot \delta_1(p_k) = \frac{1}{\delta_k} \cdot \delta_k = 1 \ ,$$

by Lemma 6. Thus we have that the maximum value of δ_T is equal to one. Lemma 5 yields that we can compute a feasible solution. The equality $\delta_T(t_u) = 1$ and Lemma 3 show that the solution is optimal, because there does not exist a feasible solution with total transfer time strictly smaller than T. □

The asymptotic running time of Algorithm 3 is obviously determined by the call to EvaluateTotalDataFunction and is thus $\mathcal{O}(m \log m)$. As a consequence, we have established Theorem 1.

Algorithm 3. FlowScaling($\mathcal{N}$)

Input. A distribution network $\mathcal{N}$.

Output. The optimal distribution $\mathbf{x}^* = (x_1, \ldots, x_m)$ for $\mathcal{N}$.

```
1: $(\mathbf{p}, \boldsymbol{\delta}) \leftarrow$ EvaluateTotalDataFunction($\mathcal{N}, 1$)
2: $k \leftarrow \arg\max_{i=1,\ldots,m}\{\delta_i\}$
3: $T \leftarrow \frac{1}{\delta_k}$
4: $t_u \leftarrow \frac{p_k}{\delta_k}$
5: return CalculateDistribution($\mathcal{N}$, $t_u$, $T - t_u$)
```

4 Conclusion

We introduced a new distribution problem that asks how a file with given size should be split and uploaded in parallel onto a set of servers such that the time for this upload and a number of subsequent parallel downloads is minimised. In contrast to other work in this area, our problem setting resembles the technological connection situation by allowing asymmetric upload and download bandwidths of the individual servers. The FlowScaling algorithm determines an optimal solution for this distribution problem by making use of a decision criterion derived from maximum-flow theory stating whether a solution with given time bounds exists. This predicate is then used to formulate the total data function which gives the maximum amount of data that can be uploaded and downloaded again within total time T. A natural scaling argument finally yields an elegant algorithm for solving the distribution problem in time $\mathcal{O}(m \log m)$.

References

1. Patterson, D.A., Gibson, G., Katz, R.H.: A case for redundant arrays of inexpensive disks (raid). In: Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data, SIGMOD 1988, pp. 109–116. ACM, New York (1988)
2. Ghemawat, S., Gobioff, H., Leung, S.-T.: The google file system. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, SOSP 2003, pp. 29–43. ACM, New York (2003)
3. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: Amazon's highly available key-value store. SIGOPS Operating Systems Review 41(6), 205–220 (2007)
4. The Wuala Project: Wuala (2008), http://www.wuala.com (Online accessed July 22, 2010)
5. Druschel, P., Rowstron, A.I.T.: Past: A large-scale, persistent peer-to-peer storage utility. In: HotOS, pp. 75–80 (2001)
6. Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Wells, C., Zhao, B.: Oceanstore: An architecture for global-scale persistent storage. In: Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS-IX, pp. 190–201. ACM, New York (2000)

7. Shen, X., Choudhary, A.: DPFS: A distributed parallel file system. In: Proceedings of the International Conference on Parallel Processing, ICPP 2001, p. 533. IEEE Computer Society, Washington (2001)
8. Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., Lewin, D.: Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In: Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing, STOC 1997, pp. 654–663. ACM, New York (1997)
9. Schindelhauer, C., Schomaker, G.: Weighted distributed hash tables. In: Proceedings of the 17th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA 2005, July17–20, pp. 218–227. ACM Press, New York (2005)
10. Schindelhauer, C., Schomaker, G.: SAN optimal multi parameter access scheme. In: Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, ICNICONSMCL 2006, p. 28. IEEE Computer Society Press, Washington (2006)
11. Langner, T.: Distributed storage in heterogeneous and asymmetric networks. Master's thesis, Albert-Ludwigs-Universität Freiburg, Germany (November 2009)
12. Hochbaum, D., Shmoys, D.: A polynomial approximation scheme for scheduling on uniform processors: Using the dual approximation approach. SIAM Journal on Computing 17(3), 539–551 (1988)

On the Complexity of the Metric TSP under Stability Considerations

Matúš Mihalák, Marcel Schöngens, Rastislav Šrámek, and Peter Widmayer

Department of Computer Science, ETH Zurich, Zurich, Switzerland

Abstract. We consider the metric Traveling Salesman Problem (Δ-TSP for short) and study how stability (as defined by Bilu and Linial [3]) influences the complexity of the problem. On an intuitive level, an instance of Δ-TSP is γ-stable ($\gamma > 1$), if there is a unique optimum Hamiltonian tour and any perturbation of arbitrary edge weights by at most γ does not change the edge set of the optimal solution (i.e., there is a significant gap between the optimum tour and all other tours). We show that for $\gamma \geq 1.8$ a simple greedy algorithm (resembling Prim's algorithm for constructing a minimum spanning tree) computes the optimum Hamiltonian tour for every γ-stable instance of the Δ-TSP, whereas a simple local search algorithm can fail to find the optimum even if γ is arbitrary. We further show that there are γ-stable instances of Δ-TSP for every $1 < \gamma < 2$. These results provide a different view on the hardness of the Δ-TSP and give rise to a new class of problem instances which are substantially easier to solve than instances of the general Δ-TSP.

1 Introduction

NP-hardness is a common concept of quantifying the complexity of an optimization problem. It can be seen as quite a pessimistic approach, since it considers the worst case running time of an algorithm for all instances of a problem. Hence, many other views on the difficulty of algorithmic problems exist. Specifically, Bilu and Linial [3] observed that an optimization problem that is NP-hard in general may turn out to be easy (i.e., polynomial time solvable) if there is one optimum solution that stands out. The notion of stability captures this idea: If the solution to a combinatorial optimization problem does not change even if we multiply the input parameters by a given factor, we call the problem instance *stable* with respect to this factor. Stability models a variety of practical considerations, such as measurement errors in input parameters. In this paper we study the well-known Traveling Salesman Problem (TSP) in the context of stability. We limit ourselves to the case in which edge weights satisfy the triangle inequality, the NP-hard metric TSP (Δ-TSP).

Our Results. We show a tight upper bound of 2 on the stability of any Δ-TSP instance. We prove that any 1.8-stable instance of Δ-TSP can be solved in polynomial time by a greedy algorithm, but we provide instances of stability 5/3 on which the same algorithm fails. In the end, we provide a class of Euclidean instances that are 2-stable and we show that on these instances a simple local search algorithm fails. This result also holds for non-Euclidean instances of arbitrary stability.

Related Work. Bilu and Linial [3] considered the MAX-CUT Problem and showed that γ-stable instances can be solved correctly in polynomial time on (i) simple graphs of minimum degree δ, when $\gamma > 2n/\delta$, where n is the number of vertices and (ii) weighted graphs of maximal degree ζ, when $\gamma > \sqrt{\zeta n}$. Balcan et al. [2] studied clusterings and restricted themselves to instances that have the (c, ϵ)-property, i.e., instances where any c-approximation of the given objective function (e.g. k-median, k-means, etc.) is ϵ-close to the optimal clustering; two clusterings are considered ϵ-close if they differ only in an ϵ-fraction of points. They showed that for such instances one can produce ϵ-close clusterings in polynomial time, even for values of c where a c-approximation is provably NP-hard. Awasthi et al. [1] proved that for center-based clustering objectives a constant stability (as defined by Bilu and Linial [3]) is sufficient to obtain an optimal clustering in polynomial time. Further, they relaxed the requirements of the (c, ϵ)-property and showed that one can still find optimal or near optimal clusterings in polynomial time. Spielman and Teng [4] introduced smoothed analysis, which on an intuitive level states that hard input instances occur very rarely at discrete points in solution space and therefore a small perturbation yields a polynomial time solvable input instance. Note that stability focuses on the structure of an instance rather than on the topology of the solution space.

In the following we introduce some notation and provide formal definitions of the required notions. Let $K_n = (V_n, E_n)$ be the complete graph on n vertices where V_n is the vertex set and E_n is the set of all undirected edges on V_n. Let $w : E_n \to \mathbb{R}^+$ be a function that assigns a positive weight to each edge. Throughout this paper we assume that w satisfies the triangle inequality, i.e. for any $u, v, x \in V_n$ it holds that $w(u, v) \leq w(u, x) + w(x, v)$. To simplify notation, we may write w'_i instead of $w(e'_i)$, as long as it does not affect readability, and for the weight $w(S) = \sum_{e \in S} w(e)$ of a set S of edges we may write w_S. Any sequence of n distinct vertices of K_n defines a Hamiltonian tour H. The set of all Hamiltonian tours is denoted by $\mathcal{S}(K_n)$. The edge set of H is denoted by $E(H)$ and we may write w_H instead of $w_{E(H)}$. Using the above notation the Δ-TSP is defined as follows. For an input instance $I = (K_n, w)$, find a $\hat{H} \in \mathcal{S}(K_n)$ such that for all $H' \in \mathcal{S}(K_n)$ we have $w_{\hat{H}} \leq w_{H'}$. We call $\hat{H}$ an optimal solution or optimal Hamiltonian tour of I.

We now formalize the notion of stability. When each edge weight of I is multiplied by an individual factor of at least 1 and at most γ, we say that the instance the instance I is perturbed by at most γ. Assume that I has a unique optimal Hamiltonian tour $\hat{H}$ (this assumption is justified later). Consider a non-optimal tour H and let $A(H)$ be the set of edges that are part of H but not of $\hat{H}$. Further, let $D(H)$ be the set of edges that are part of $\hat{H}$ but not of H. A perturbation causes $\hat{H}$ to become non-optimal iff there is a non-optimal H s.t. the perturbation causes $w_{D(H)}$ to become greater than $w_{A(H)}$. Thus, there exists a Hamiltonian tour H, s.t. the weight ratio of $A(H)$ and $D(H)$ limits the size of a perturbation that does not change the optimal solution.

Definition 1. *For a non-optimal Hamiltonian tour H we define the* disjoint ratio *of H to be $\mathcal{R}(H) = w_{A(H)}/w_{D(H)}$.*

The disjoint ratio is consequently always strictly greater than 1 as otherwise H would be optimal. An instance is as stable as the smallest disjoint ratio w.r.t. all non-optimal tours H (see Figure 3a).

Definition 2. *Let I be an input instance of the Δ-TSP with a unique optimal Hamiltonian tour $\hat{H}$. We say I is* γ-stable *or has* stability γ *if for all non-optimal Hamiltonian tours H it holds that $\mathcal{R}(H) > \gamma$.*

Now we can see why the requirement for an instance with unique optimal solution is relevant to our definition. An instance I' with more than one optimal Hamiltonian tour has the property that even a small perturbation on the edge weights may transform any optimal solution H of the original instance I' to a *unique* optimal solution of the perturbed instance. W.r.t. our definition such instances are highly unstable, we may thus state that if I has more than one optimal solution it is 1-stable.

In order to simplify some proofs in later sections, we mention an equivalent view on stability and show some immediate consequences of the definitions. A non-optimal Hamiltonian tour H can always be constructed out of $\hat{H}$ by two consecutive steps. First, by deleting all edges $D(H)$ from $\hat{H}$, then by adding all edges $A(H)$. Note that $|A(H)| = |D(H)|$ and $|A(H)| > 1$. The disjoint ratio of any Hamiltonian tour H bounds the γ-stability of I from above. Further, if I is γ-stable, then I is also γ_1-stable for any $\gamma_1 \leq \gamma$. Likewise, if I is not γ-stable, then I is not γ_2-stable for any $\gamma_2 \geq \gamma$.

We conclude this section by stating a simple observation to bound fractions of sums more tightly than trivial estimates.

Observation 1. *Let $a \geq b > 0$ and $x \geq c > 0$ be real numbers. Then*

$$\frac{a+x}{b+x} \leq \frac{a+c}{b+c}.$$

The rest of the paper is structured as follows. In Section 2 we show that the stability of the Δ-TSP has a tight upper bound of 2. In Section 3 we show that a simple greedy algorithm solves every 1.8-stable input instance correctly in polynomial time, but it may fail for 5/3-stable instances as we show in Section 4. In Section 5 we show that a simple local search algorithm cannot exploit the properties of stable instances.

2 A Tight Upper Bound on the Stability of Δ-TSP

In this section we show that any input instance of the Δ-TSP has stability smaller than 2, and, to show tightness, we provide a sequence of instances with stability converging to this bound.

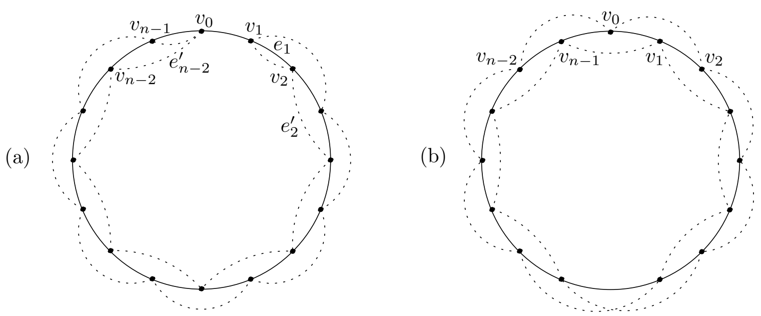


Fig. 1. Two input instances of Δ-TSP, where the solid line in the unique optimum tour $\hat{H}$, and the dashed line is a tour with disjoint ratio of at most 2. (a) shows the case where n is even and (b) shows the case where n is odd.

Note that in any Hamiltonian tour H we can find an edge that has weight greater than or equal to that of its two neighbors. We call such an edge a *local maximum* for H.

In the following, we show that any input instance I with a unique optimal Hamiltonian tour $\hat{H}$ of the Δ-TSP has stability of less than 2.

The basic idea is that for any I we can find a non-optimal Hamiltonian tour H that has at most double the weight of $\hat{H}$. We can construct such an H from $\hat{H}$ by walking along the edges of $\hat{H}$, and for any two consecutive edges $(u, v), (v, x) \in E(\hat{H})$ we take the direct edge (u, x) for H.

Formally, we do a case differentiation between input instances of even and odd number of vertices. W.l.o.g., let $\hat{H} = (v_0, v_1, \dots, v_{n-1})$ be the optimal Hamiltonian tour.

Case 1: (number of vertices n is even)

Consider the non-optimal Hamiltonian cycle

$$H = (v_1, v_2, v_4, v_6, \dots, v_{n-2}, v_0, v_{n-1}, v_{n-3}, v_{n-5}, \dots, v_5, v_3),$$

which formalizes the intuitive construction from above. W.l.o.g. let the edge $e_0 = (v_0, v_1)$ be a local maximum. Let $e_i = (v_i, v_{i+1})$ for $1 \le i \le n-1$, so that the edges removed from the optimal cycle will be $D(H) = \{e_0\} \cup \bigcup_{i=2}^{n-2} \{e_i\}$. Let e'_i be the edges connecting every second vertex, i.e., $e'_i = (v_i, v_{i+2})$ for $0 \le i \le (n-1)$. Then the edges substituted for $D(H)$ will be $A(H) = \bigcup_{i=1}^{n-2} \{e'_i\}$. For illustration see Figure 1a. Since the triangle inequality holds, we get $w'_i \le w_i + w_{i+1}$ and since e_0 is a locally maximal edge, we get that $w_{n-1} + w_1 \le 2w_0$. Then we can bound the disjoint ratio:

$$\mathcal{R}(H) = \frac{\sum_{i=1}^{n-2} w'_i}{w_0 + \sum_{i=2}^{n-2} w_i} \le \frac{\sum_{i=1}^{n-2}(w_i + w_{i+1})}{w_0 + \sum_{i=2}^{n-2} w_i} = \frac{w_1 + w_{n-1} + 2\sum_{i=2}^{n-2} w_i}{w_0 + \sum_{i=2}^{n-2} w_i} \le 2$$

Case 2: (number of vertices n is odd)

In this case we use all the edges e'_i without reusing any edge of the optimal

Hamiltonian cycle. H will be $(v_0, v_2, \ldots, v_{n-1}, v_1, v_3, \ldots, v_{n-2})$ (see Figure 1b, dashed edges). Then, again using triangle inequality, we get

$$\mathcal{R}(H) = \frac{\sum_{i=0}^{n-1} w'_i}{\sum_{i=0}^{n-1} w_i} \leq \frac{w_{n-1} + w_0 + \sum_{i=0}^{n-2}(w_i + w_{i+1})}{\sum_{i=0}^{n-1} w_i} = 2$$

Thus, for any input instance one can find a non-optimal Hamiltonian tour with disjoint ratio at most 2, which bounds the stability from above.

Now, to show the tightness of this bound, we provide an infinite sequence $(I_n)_{n\geq 4}$ of input instances that are at least $\gamma(n)$-stable and for which $\gamma(n)$ converges to 2 as n tends to infinity.

Intuitively, the n-th instance of the sequence is a complete graph K_n with n vertices distributed equidistantly onto a unit circle, and edge weights that correspond to the Euclidean distance. The stability of I_n grows monotonically in n.

Formally, for the instance $I_n = (K_n, w_n)$ of a sequence $(I_n)_{n\geq 4}$ we only need to define w_n. Let $f : V_n \to \mathbb{R}^2, f(v_i) = i * 2\pi/n$ map the i-th vertex of K_n to the point that lies on a unit circle in two dimensional Euclidean space, and has radian measure $f(v_i)$. Then, $w_n(u, v) = ||f(u) - f(v)||_2$.

Now, we bound the stability of I_n from below. By simple trigonometry the edges (v_i, v_{i+1}), for $0 \leq i \leq n-1$, have weight $2\sin(\pi/n)$ and form a unique optimal Hamiltonian tour $\hat{H}$. All other edges have weight at least $2\sin(2\pi/n)$. Thus, the disjoint ratio of any non-optimal tour H differing in k edges from $\hat{H}$ is

$$\mathcal{R}(H) = \frac{w_{A(H)}}{w_{D(H)}} \geq \frac{k2\sin(2\pi/n)}{k2\sin(\pi/n)} = \frac{2\cos(\pi/n)\sin(\pi/n)}{\sin(\pi/n)} = 2\cos(\pi/n)\,.$$

Therewith, instance I_n is $(2\cos(\pi/n) - \epsilon)$-stable for any $\epsilon > 0$. Since $2\cos(\pi/n) - \epsilon$ is monotonic increasing in n, any suffix of the sequence $(I_n)_{n\geq 4}$ starting from I_m is $(2\cos(\pi/m) - \epsilon)$-stable.

3 A Polynomial Time Algorithm for 1.8-Stable Instances

In this section we give a polynomial time algorithm for the Δ-TSP and prove that it works correctly if the input is 1.8-stable.

The algorithm works similarly to Prim's algorithm for computing a minimum spanning tree: Intuitively, in each step it merges two paths until a Hamiltonian path is obtained, which can be connected to a Hamiltonian tour. In the following we describe the algorithm formally. Let $I = (K_n, w)$ be a γ-stable input instance and let $C_i = (v_{i_0}, v_{i_1}, \ldots, v_{i_m})$ be a vertex disjoint path in K_n of length m. Initially, the algorithm treats any vertex as a path of length 0. Let $C_j = (v_{j_0}, v_{j_1}, \ldots, v_{j_{m'}})$ be another vertex disjoint path in K_n. Paths C_i and C_j can be merged to a path of length $m + m' + 1$ by using one of the edges (v_{i_0}, v_{j_0}), $(v_{i_0}, v_{j_{m'}})$, (v_{i_m}, v_{j_0}) or $(v_{i_m}, v_{j_{m'}})$ as a link. We call such edge an *inter-path edge.* To find the two paths to merge, the algorithm runs through every combination of two paths C_i, C_j to find a minimum weight inter-path edge. In case of ties we take the minimum weight inter-path edge which is found last. The algorithm iterates until only one path is left. This is a Hamiltonian path and by connecting its endpoints we obtain a Hamiltonian tour.

We prove the correctness of the described algorithm. The algorithm terminates because in each step the number of paths decreases until there is only one path left. The Hamiltonian path found by the algorithm consists of all n vertices of the complete graph and therefore represents a Hamiltonian tour.

It remains to be shown that the obtained Hamiltonian tour has minimum weight if the input is 1.8-stable. For the sake of contradiction, we assume that the algorithm does not work correctly and under this assumption we show that the input instance is not 1.8-stable. Let the optimal Hamiltonian tour be $\hat{H} = (v_0, v_1, \ldots, v_{n-1})$. The algorithm errs in some round for the first time when it chooses two paths which are connected by an edge e of minimum weight but e is not a part of the optimal Hamiltonian tour. Let the endpoints of e be v_x and v_y.

For each edge $e' \in E(\hat{H})$, that is adjacent to a vertex not contained in a path of length greater than 0, it holds

$$w_e \leq w_{e'}, \tag{1}$$

as otherwise e' would have been chosen by the algorithm instead of e. We consider two possible cases. First, e is chosen such that at least one endpoint, v_x or v_y, is the only vertex of its path. The second case deals with the situation where both v_x and v_y are not paths consisting of a single vertex. For each case we show that the input is not 1.8-stable.

Case 1: (v_x or v_y is in a path of length 0)
W.l.o.g. assume that v_y is in a path of length 0 and v_x may or may not be the only vertex of its path. Further, w.l.o.g. we assume that $e_x = (v_x, v_{x+1})$ has not yet been chosen (see Figure 2). Thus, condition (1) holds for $e_{y-1} = (v_{y-1}, v_y), e_y = (v_y, v_{y+1})$ and for e_x. We consider the Hamiltonian tour $H = (v_x, v_y, v_{x+1}, v_{x+2}, \ldots, v_{y-1}, v_{y+1}, \ldots v_{x-1})$ that differs from $\hat{H}$ in the edges

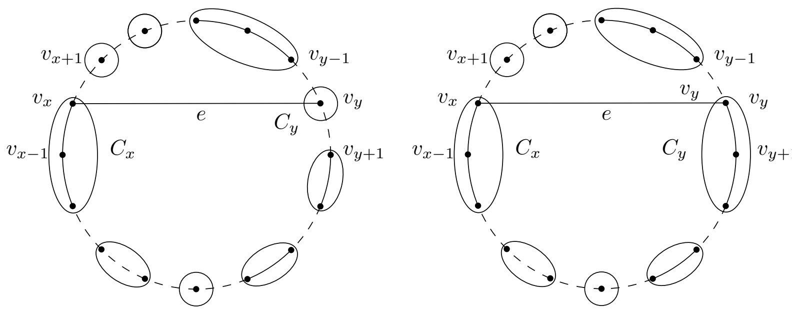


Fig. 2. The algorithm at some step of execution: There are paths already constructed in previous rounds and the algorithm errs for the first time by chosing the edge e. Left: v_y is the only vertex of its path. Right: v_x and v_y are contained in paths with at least two vertices. Note that the graph is complete, but not all edges are drawn.

$D(H) = \{e_x, e_{y-1}, e_y\}$ and $A(H) = \{e, e_{h_1}, e_{h_2}\}$ where $e_{h_1} = (v_y, v_{x+1})$ and $e_{h_2} = (v_{y-1}, v_{y+1})$. Thus, the disjoint ratio of H is

$$\mathcal{R}(H) = \frac{w(e) + w(e_{h_1}) + w(e_{h_2})}{w(e_x) + w(e_{y-1}) + w(e_y)} \leq \frac{2w(e) + w(e_x) + w(e_{y-1}) + w(e_y)}{w(e_x) + w(e_{y-1}) + w(e_y)}, \tag{2}$$

where the last inequality follows from the triangle inequality. Remember that condition (1) holds for all edges of the latter equation. We further estimate the disjoint ratio using Observation 1, since trivial estimates would yield a much worse bound. We get $\mathcal{R}(H) \leq 5w(e)/3w(e) = 5/3$. In this case, the input instance is not 5/3-stable (and thus not 1.8-stable).

Case 2: (v_x and v_y are contained in paths of length at least 1)
Denote the paths connected by e with C_x and C_y. Because v_x is not the only vertex in C_x we assume w.l.o.g. that $v_{x-1} \in C_x$ and therefore $e_{x-1} \in C_x$. By the same argument $e_y \in C_y$. For e_{x-1} and e_y it holds that $w(e) \geq w(e_{x-1})$ and $w(e) \geq w(e_y)$, because otherwise e would have been chosen in an earlier round of the algorithm, before e_{x-1} and e_y were chosen. We now partition all possible instances I by another case differentiation. Let $l \geq 1$ be some real number.

Case 2a: ($w(e_x)/w(e_{x-1}) > l$ or $w(e_{y-1})/w(e_y) > l$ for $l \geq 1$)
W.l.o.g. assume that $w(e_x)/w(e_{x-1}) > l$. Then we can find three consecutive edges in $\hat{H} = (v_1, v_2, \ldots, v_n)$ denoted by e_{k_1}, e_{k_2} and e_{k_3} for which the following two conditions hold: Condition 1 is $w(e_{k_1}) > lw(e_{k_2})$ and Condition 2 is $w(e_{k_2}) \leq lw(e_{k_3})$. Otherwise each edge on the Hamiltonian tour would have to be larger than the previous one.
Consider the Hamiltonian tour $H = (v_1, \ldots, v_{k_1}, v_{k_3}, v_{k_2}, v_{k_4}, \ldots, v_n)$ that differs from $\hat{H} = (v_1, \ldots, v_{k_1}, v_{k_2}, v_{k_3}, v_{k_4}, \ldots, v_n)$ only in the two edges e_{k_1} and e_{k_3}. The disjoint ratio of H is

$$\mathcal{R}(H) = \frac{w(v_{k_1}, v_{k_3}) + w(v_{k_2}, v_{k_4})}{w(e_{k_1}) + w(e_{k_3})} \leq \frac{w(e_{k_1}) + w(e_{k_2}) + w(e_{k_3}) + w(e_{k_2})}{w(e_{k_1}) + w(e_{k_3})},$$

where the last inequality is due to the triangle inequality. As in Case 1, to get a better estimate on the disjoint ratio, we apply Observation 1. Since $w(e_{k_1})/w(e_{k_2}) > l$ and $w(e_{k_2})/w(e_{k_3}) \leq l$, we get,

$$\mathcal{R}(H) \leq \frac{lw(e_{k_2}) + 2w(e_{k_2}) + 1/l \cdot w(e_{k_2})}{lw(e_{k_2}) + 1/l \cdot w(e_{k_2})} < \frac{l^2 + 2l + 1}{l^2 + 1}.$$

For the considered case this directly yields an upper bound for the γ-stability of I, i.e., $\gamma < \frac{l^2+2l+1}{l^2+1}$ for $l \geq 1$.

Case 2b: ($w(e_x)/w(e_{x-1}) \leq l$ and $w(e_{y-1})/w(e_y) \leq l$ for $l \geq 1$)
Again let $\hat{H} = (v_x, v_{x+1} \ldots, v_{y-1}, v_y, v_{y+1}, \ldots, v_{x-1})$ be the optimal Hamiltonian tour in I. As in Case 1 let the suboptimal Hamiltonian tour be $H = (v_x, v_y, v_{x+1}, \ldots, v_{y-1}, v_{y+1}, \ldots, v_{x-1})$ (see Figure 3b), and consequently the disjoint ratio of H is also bounded by inequality (2).

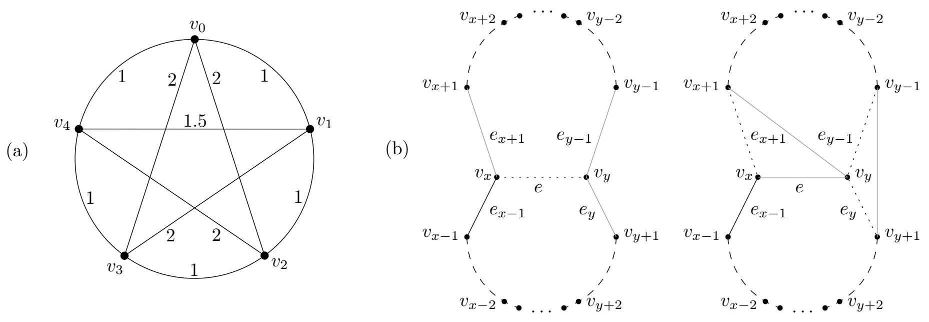


Fig. 3. (a) This is an input $I = (K_5, w)$ of the Δ-TSP with optimal Hamiltonian tour $\hat{H} = (v_0, v_1, v_2, v_3, v_4)$. (i) Consider the non-optimal tour $H = (v_0, v_1, v_4, v_3, v_2)$. Note that $A(H) = \{(v_0, v_2), (v_1, v_4)\}$ and $D(H) = \{(v_0, v_4), (v_1, v_2)\}$. The disjoint ratio is $\mathcal{R}(H) = w_{A(H)}/w_{D(H)} = 2.5/2 = 1.25$. (ii) Instance I is $(1.25 - \epsilon)$-stable $(\epsilon > 0)$ because there is no non-optimal tour H' with $\mathcal{R}(H') < \mathcal{R}(H)$. (b) On the left: optimal Hamiltonian cycle $\hat{H}$ and edge e. On the right: Other Hamiltonian cycle H differing from $\hat{H}$ only in three edges (gray). Note that the graph is complete and unimportant edges are omitted in the figure.

By using inequalities (1), the premise of this case, and by applying Observation 1, we obtain

$$\mathcal{R}(H) \leq \frac{w(e) + w(e) + w(e) + 1/l \cdot w(e) + w(e)}{w(e) + 1/l \cdot w(e) + w(e)} = \frac{4l+1}{2l+1}.$$

Thus, the disjoint ratio of H is at most $\frac{4l+1}{2l+1}$, and consequently $\gamma < \mathcal{R}(H) \leq \frac{4l+1}{2l+1}$ for an arbitrary input instance I satisfying the conditions of this case.

We combine the two cases (2a and 2b) to obtain an upper bound on the stability of the input instance I. Because we do not know which case holds for I, we search for an l that is best in both cases, i.e., we search for

$$\min_{l \geq 1} \max \left\{ \frac{l^2 + 2l + 1}{l^2 + 1}, \frac{4l+1}{2l+1} \right\} ,$$

which is an upper bound for γ. The minimum of 1.8 is obtained for $l = 2$ and thus $\gamma < 1.8$. This is now a contradiction to the assumption that I is 1.8-stable. This completes the proof. □

4 The Algorithm Fails for $(5/3 - \epsilon)$-Stable Instances

In this section we present a family $\mathcal{F}^B$ of input instances for the Δ-TSP that are $(5/3 - \epsilon)$-stable for $\epsilon > 0$ and for which the algorithm errs. We proceed by defining $\mathcal{F}^B$ formally and showing that the instances in $\mathcal{F}^B$ obey the triangle inequality. Subsequently we prove that the greedy algorithm errs on any instance in $\mathcal{F}^B$ and that any instance is $(5/3 - \epsilon)$-stable.

Let $G = (V_n, E')$ be a graph on an even number of vertices where $E' = \{(v_i, v_{i+1}) \in E_n \mid 0 \leq i \leq n-2\} \cup \{(v_0, v_{n-1})\} \cup \left\{(v_0, v_{n/2})\right\}$ (the solid lines in Figure 4a). All edges in E' have unit weight. We define $w_n(v_i, v_j)$ to be the distance between v_i and v_j in G. Let $\mathcal{F}^B = \{I_n = (K_n, w_n) \mid n = 2k, k \geq 5\}$. Note that w_n defines the shortest-path metric of G on K_n and therefore it satisfies the triangle inequality. Therefore every $I_n \in \mathcal{F}^B$ is an instance of the Δ-TSP.

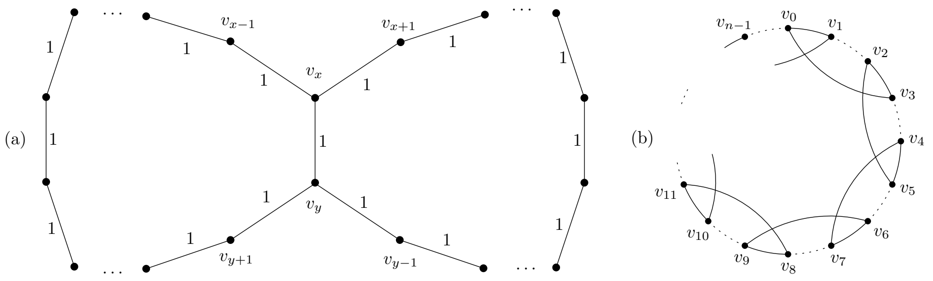


Fig. 4. (a) The algorithm chooses the edge in the middle to merge to components. This yields a non-optimal Hamiltonian tour. (b) The solid edges have weight 1, the dashed edges weight 2. The solid lines form the optimal solution. A local optimum is the tour $(v_0, v_1, \ldots, v_{n-1})$.

First notice that every instance $I_n \in \mathcal{F}^B$ has exactly one optimal Hamiltonian tour $\hat{H} = (v_0, v_1, \ldots, v_{n-1})$. This follows from the following argument: A Hamiltonian cycle H cannot differ in only one edge, so it must differ in at least two. Observe that all edges *not* in $\hat{H}$ have weight greater than 1 except for $(v_0, v_{n/2})$. Thus, the weight of H must be greater than the weight of $\hat{H}$.

We show that the algorithm errs on each instance in $\mathcal{F}^B$. In any step of its execution, the algorithm identifies an edge of minimum weight among all edges that can connect two paths. Let us suppose that the edge $(v_0, v_{n/2})$ (which is not part of $\hat{H}$) is chosen first by the algorithm. Thus, for all $I_n \in \mathcal{F}^B$ the algorithm fails.

In the following we show that $I_n \in \mathcal{F}^B$ is $(5/3 - \epsilon)$-stable, i.e., each non-optimal Hamiltonian cycle H in I_n has a disjoint ratio greater or equal to 5/3. We do a case differentiation and first consider every H that can be obtained by deleting and adding $k = 2$ edges, i.e., $|D(H)| = |A(H)| = 2$. Then, we consider every H with $|D(H)| = |A(H)| \geq 3$.

Case 1: $(|D(H)| = 2)$

Let $D(H) = \{(v_i, v_{i+1}), (v_j, v_{j+1})\}$ with $0 \leq i, j \leq n-1$. Note that $j \neq i+1$ and $j+1 \neq i$ as otherwise $H = \hat{H}$. We then have, $A(H) = \{(v_i, v_j), (v_{i+1}, v_{j+1})\}$. Note that all edges in $D(H)$ have weight 1.

Assume $A(H)$ does not contain $(v_0, v_{n/2})$, thus by definition each added edge has weight at least 2. Therewith, we derive $\mathcal{R}(H) \geq 2$.

Assume $(v_0, v_{n/2})$ is in $A(H)$. Then both edges in $D(H)$ must have the following property: the edges must be adjacent to $(v_0, v_{n/2})$ as otherwise we cannot get an Hamiltonian tour by adding $(v_0, v_{n/2})$. To avoid cycles, it follows that the edges must either be $(v_0, v_1), (v_{n/2}, v_{n/2+1})$ or $(v_{n-1}, v_0), (v_{n/2-1}, v_{n/2})$. Thus, for the second edge in $A(H)$ there are only two choices, and for both we obtain a weight of 3. This yields $\mathcal{R}(H) \geq (1+3)/2 = 2$.

Case 2: $(|D(H)| > 2)$

Recall that H can be constructed out of $\hat{H}$ by removing $|D(H)|$ edges and adding $|A(H)|$ edges. The deletion of $D(H)$ decomposes $\hat{H}$ in $k > 2$ paths C_i for $1 \leq i \leq k$. In $\hat{H}$ each C_i is connected to other paths by exactly two edges of $D(H) \subset E(\hat{H})$. The edges connecting C_i to other paths are denoted by $D(C_i)$. After removing all edges in $D(H)$ we have to reconnect each path C_i with exactly two edges of $A(H) \not\subset E(\hat{H})$ so that we obtain the Hamiltonian tour H. We denote these edges by $A(C_i)$. Accordingly, the disjoint ratio is

$$\mathcal{R}(H) = \frac{\sum_{i=1}^{k} w_n(A(C_i))}{\sum_{i=1}^{k} w_n(D(C_i))}.$$

We say $w_n(D(C_i))$ is the *negative* and $w_n(A(C_i))$ is the *positive contribution of* C_i to the disjoint ratio.

For any path the weight of the two edges disconnecting C_i from the optimal Hamiltonian tour $\hat{H}$ is $w_n(D(C_i)) = 2$. In the following we differentiate between paths incident and not incident to $(v_0, v_{n/2})$.

Case 2a: $((v_0, v_{n/2})$ is incident to a vertex in $C_i)$

In the worst case C_i could be connected to a path using $(v_0, v_{n/2})$, i.e. $(v_0, v_{n/2}) \in A(H)$. The second edge needed to connect C_i to another path must have a weight of at least 2 since the only edges of weight 1 are those in $E(\hat{H}) \cup \{(v_0, v_{n/2})\}$. Thus, $w_n(A(C_i)) \geq 1 + 2 = 3$.

Case 2b: $((v_0, v_{n/2})$ is not incident to a vertex in $C_i)$

To construct H we have to connect C_i to paths via edges that have weight at least 2, as otherwise we would use an edge of $D(C_i)$. Thus, $w_n(A(C_i)) \geq 2 + 2 = 4$.

Below we combine both types of paths to get a lower bound for the disjoint ratio of any non-optimal Hamiltonian tour H. In Case 1, where we only exchange 2 edges, we get a lower bound on the disjoint ratio of $\mathcal{R}(H) \geq 2$. In Case 2 we remove $k > 2$ edges from $\hat{H}$. Note that at most two of the emerging paths are of type (2a). As a result, for the disjoint ratio H we have

$$\mathcal{R}(H) = \frac{\sum_{i=1}^{k} w_n(A(C_i))}{\sum_{i=1}^{k} w_n(D(C_i))} \geq \frac{3 + 3 + \sum_{i=1}^{k-2} 4}{\sum_{i=1}^{k} 2}.$$

This is minimized for $k = 3$ which yields $\mathcal{R}(H) \geq \frac{5}{3}$. Thus, the input instance I_n is $(5/3 - \epsilon)$-stable for ananyy $\epsilon > 0$.

5 Stability is Not Good for Local Search

In this section we show that a local search algorithm does not benefit from stable instances. The idea is to present a family $\mathcal{F}^C$ of $(2-\epsilon)$-stable $(\epsilon > 0)$ input instances which have local optima that are not global optima.

To introduce local search formally, let H be a not necessarily optimal Hamiltonian tour of the Δ-TSP. Then, a Hamiltonian tour H' is called a *neighbor* of H, if one can substitute two non-adjacent edges (u, v), (x, y) of H by (u, y),(v, x) to obtain H'. The set of all neighbors of H is called the *neighborhood* of H. A Hamiltonian tour H is called a *local optimum* if all neighbors H' have greater or equal weight. A global optimum $\hat{H}$ is the optimal solution of the given input instance I, i.e., $\forall H \in \mathcal{S}(I) : w_{\hat{H}} \le w_H$. Our local search algorithm takes as an input a Hamiltonian tour H, and explores all $\mathcal{O}\left(n^2\right)$ Hamiltonian tours of H's neighborhood to find a minimum-weight Hamiltonian tour H'. If $w_{H'} < w_H$, then the algorithm recurs on H', otherwise it stops and outputs H.

Consider the family of stable instance $\mathcal{F}^C = \{I_n = (K_n, w_n) \mid n \ge 6 \text{ even}\}$ (see Figure 4b) where the weight function on the edges is defined as:

$$w_n(e) = \begin{cases} 1 & \text{if } e = (v_{2i}, v_{2i+1}) \text{ or } e = (v_{2i}, v_{2i+3}) \text{ for } 0 \le i \le n/2 - 1 \\ 2 & \text{otherwise .} \end{cases}$$

Any instance of this family satisfies the triangle inequality since there are only weights 1 and 2.

To prove a stability of $(2 - \epsilon)$ for all instances in $\mathcal{F}^C$, note that all edges of weight 1 define the unique optimal Hamiltonian tour $\hat{H}$. All other edges have weight 2. For any non-optimal Hamiltonian tour H we have that $D(H)$ contains edges of weight 1 only, and $A(H)$ contains edges of weight 2 only. Since $|D(H)| = |A(H)|$ holds for any H, the disjoint ratio is $\mathcal{R}(H) = 2$.

We show that an instance I_n in $\mathcal{F}^C$ has a local optimum that is not a global optimum. The idea is to prove that the solution $H^{LO} = (v_0, v_1, \ldots, v_{n-1})$ has no neighboring solution of strictly lower weight. We describe a best possible local search step: one of the two added edges must have weight 1 as otherwise we would have no improvement. Consider any edge e not in H^{LO} with $w(e) = 1$ and add it to H^{LO}. The resulting graph $H^{LO'}$ is not a cycle anymore. To obtain this property again, we have to delete two edges adjacent to e that are not incident to the same vertex and in different cycles in H^{LO}. No matter which two edges we choose, by the construction of the class $\mathcal{F}^C$ we have to remove one edge with weight 2 and one with weight 1. Finally, we must add an edge with weight 2 to obtain a Hamiltonian tour again. See Figure 4b for illustration. Our relative improvement in weight is thus $1 - 2 - 1 + 2 = 0$, which is not strictly better. Thus, we are stuck in a local optimum that is not the global optimum $\hat{H}$.

Note that if we abandoned the restriction to metric TSP and used arbitrary weights for those edges that have weight 2 in the metric instance, we would see that the local search algorithm can get stuck in local optima even for general TSP instances of arbitrary stability.

6 Conclusion

In this paper we studied properties of the metric TSP in the context of stability. We showed that a Prim style algorithm works correctly for any instance which is at least 1.8-stable. On the other hand, we showed that high stability does not lead to very simple instances since there exist instances with arbitrary stability, on which a local improvement algorithm will fail. There are many further questions even in our very limited scope. Is there a polynomial time algorithm that can determine the stability of a given instance? Are there any interesting classes of graphs that are stable? What stability can be expected from instances generated by randomly picking points in the Euclidean plane? Is it possible to eliminate the gap between 5/3 and 1.8 in the analysis of the algorithm? Our conjecture is that the lower bound of 1.8 can be improved further.

We believe that stability as defined earlier is worthy of research focus, since it is a new way of dissecting NP-complete problems into smaller classes, some of which are efficiently solvable.

Acknowledgments. We would like to thank the anonymous reviewers for their useful comments and suggestions.

References

1. Awasthi, P., Blum, A., Sheffet, O.: Clustering Under Natural Stability Assumptions. Tech. rep., Computer Science Department (2010)
2. Balcan, M.-F., Blum, A., Gupta, A.: Approximate clustering without the approximation. In: Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, pp. 1068–1077 (2009)
3. Bilu, Y., Linial, N.: Are Stable Instances Easy? In: The First Symposium on Innovations in Computer Science, ICS 2010, pp. 332–341 (2010)
4. Spielman, D.A., Teng, S.-H.: Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In: Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, STOC 2001, pp. 296–305 (2001)

Structural Properties of Hard Metric TSP Inputs[⋆]
(Extended Abstract)

Tobias Mömke

KTH — Royal Institute of Technology, Stockholm, Sweden[⋆⋆]
moemke@kth.se

Abstract. The metric traveling salesman problem is one of the most prominent APX-complete optimization problems. An important particularity of this problem is that there is a large gap between the known upper bound and lower bound on the approximability (assuming $P \neq NP$). In fact, despite more than 30 years of research, no one could find a better approximation algorithm than the 1.5-approximation provided by Christofides. The situation is similar for a related problem, the metric Hamiltonian path problem, where the start and the end of the path are prespecified: the best approximation ratio up to date is 5/3 by an algorithm presented by Hoogeveen almost 20 years ago.

In this paper, we provide a tight analysis of the combined outcome of both algorithms. This analysis reveals that the sets of the hardest input instances of both problems are disjoint in the sense that any input is guaranteed to allow at least one of the two algorithms to achieve a significantly improved approximation ratio. In particular, we show that any input instance that leads to a 5/3-approximation with Hoogeveen's algorithm enables us to find an optimal solution for the traveling salesman problem. This way, we determine a set S of possible pairs of approximation ratios. Furthermore, for any input we can identify one pair of approximation ratios within S that forms an upper bound on the achieved approximation ratios.

1 Introduction

While being one of the hardest problems with respect to approximability in its general formulation [21], the metric traveling salesman problem (ΔTSP) is well know to be APX-complete. Unless $P = NP$, it does not permit an approximation ratio that is better than 220/219 [19]. The best algorithm available is a 1.5-approximation algorithm by Christofides [10]. The situation is very similar for the metric Hamiltonian path problem with prespecified start and end vertex (ΔHPP$_2$): the lower bound is closely related to that of the ΔTSP and the

[⋆] The research was partially funded by SNF grant 200021-109252/1 and ERC Advanced investigator grant 226203.

[⋆⋆] The results of this paper were obtained at ETH Zurich, Switzerland.

5/3-approximative algorithm by Hoogeveen [16] was not improved so far. An alternative proof for the same result was given by Guttmann-Beck et al. [15]. For both problems, the upper bounds on the approximability have resisted all attempts of improvement for many years.

The two problems are closely related, since both of them take a complete metric graph as input and the goal of both problems is to visit each of the vertices. The ΔTSP is basically the $\Delta \mathrm{HPP}_2$, where the start vertex and the end vertex are the same.

In this paper, we significantly improve our former result from [8]. We characterize hard input instances for both Christofides' and Hoogeveen's algorithm and show that the sets of worst-case instances for these algorithms are disjoint in the sense that a hard instance for one problem allows a significantly improved approximation for the other one. We determine the set S of all possible pairs of approximation ratios that are achieved this way (depicted in Figure 1). This includes the guarantee that a worst-case input for Hoogeveen's algorithm, for which we can only compute a 5/3-approximative solution, enables us to compute an optimal solution for ΔTSP on that input. We are guaranteed that the cost of this optimal solution is exactly 4/3 times as high as that of an optimal solution for $\Delta \mathrm{HPP}_2$. In addition to the results on the structure of hard inputs, we show that for each input, we can determine a pair of approximation ratios from S that forms an upper bound on the achieved ratios.

To show that our analysis is tight, we present a class of hard input instances for each of the possible pairs from S that forms an upper bound.

Our detailed analysis of these algorithms provides deep insight of the core of the hardness involved in classes of input instances for which we cannot provide an improved approximation. We show for instance that in any worst case instance for Hoogeveen's algorithm, the minimum spanning tree involved in the algorithm contains a path between the end vertices of cost exactly 1/3 of the cost of an optimal solution and a gradual relaxation of this bound for inputs that do not cause worst-case behavior. This generalizes some of the results from [15]. The properties revealed in this work restrict the types of inputs that a possible improved algorithm for the $\Delta \mathrm{HPP}_2$ has to cope with. This might be helpful for creating improved algorithms for the $\Delta \mathrm{HPP}_2$ or the ΔTSP.

1.1 Related Known Results

The result of this paper is a win/win strategy for approximation algorithms. The concept of win/win strategies is to specify a parameter of the input instance and to guarantee — for any value of the parameter — that we can compute an improved solution for one of two problems according to some complexity measure. Here, the parameter is the computed bound on the approximation ratio for the Hamiltonian path problem and the complexity measure is the approximation ratio.

Win/win strategies fit well into the framework of parameterized complexity [11,18] as well as stability of approximation [5,14,7], because all of these approaches are based on studying the "hardness" of their problem instances.

In parameterized algorithms, win/win strategies are a tool used for kernelization [1], which is a technique used in order to reduce the size of the problem instance and the parameter. Prieto and Sloper presented such a kernelization of the k-internal spanning tree problem by using a win/win strategy that relates the k-internal spanning tree problem and the vertex cover problem [20]. An overview on the use of win/win strategies in parameterized algorithms can be found in [13].

The concept of win/win strategies relates to the design of hybrid algorithms as proposed by Vassilevska et al. [22]. They presented algorithms that allow either an improved approximation ratio or an improved (but still exponential) runtime for computing exact solutions.

Win/win strategies for approximation were independently introduced in our paper [8] and by Eppstein [12]. Eppstein uses the name *paired approximation* for this concept. He was able to use win/win as an upper bound technique and he showed for some pairs of problems that they do not have such a relation.

The win/win result from [8] is related to the result of this paper, but the achieved pairs of approximation ratios of this paper are significantly improved (see Figure 1). The result from [8] is existential in the sense that it does not provide the possibility to identify pairs of approximation ratios within the given boundaries.

Our results open an interesting connection to another field of algorithmics called *reoptimization.* In reoptimization, one is given an optimal or almost optimal solution for some input instance. Now the problem is to find a solution for a different input instance that is closely related to the given one. Some approximation results on reoptimization can be found in [2,3,4,6]. In [9], one can find an overview on reoptimization. Let us consider the reoptimization problem of $\Delta\mathrm{HPP}_2$, where the modification is to change one of the end vertices. If there is an approximation algorithm for this reoptimization problem that is better than 5/3-approximative, then we can use this algorithm for $\Delta\mathrm{HPP}_2$: for a given worst-case instance I of Hoogeveen's algorithm, we determine an optimal ΔTSP solution and use this as input for the reoptimization problem by declaring the start vertex of I to be the start vertex as well as the end vertex. Then I is the modified instance that is to be solved by the reoptimization algorithm. However, to improve the approximation ratio for $\Delta\mathrm{HPP}_2$ by a constant factor, we depend on the ability of the reoptimization algorithm to handle a broader range of input instances: instead of requiring an optimal solution for the given input graph, it has to be able to accept solutions that deviate by a (small) constant factor from an optimal solution.

1.2 Organization of the Paper

Section 2 fixes the notation used in the paper. The core of this paper is located in Section 3, where we show the combined upper bounds on the approximation ratios achieved by Christofides' and Hoogeveen's algorithm. Section 4 then provides a more detailed analysis of hard instances. Finally, Section 5 shows that the analyses of all upper bounds shown in this paper are tight.

2 Preliminaries

For graphs, we use a notation similar to [23]. In a graph $G = (V, E)$, the edges are sets of two vertices. A *trail* from u to v is a sequence of adjacent edges leading from u to v, where no edge may be used more than once. A trail is uniquely defined by a list of vertices $uw_1w_2 \dots w_iv$, where consecutive vertices describe the edges of the trail. We say that $w_1 \dots w_i$ are the *inner vertices*. The *length of a trail* is the number of its edges. A trail, where each vertex is used at most once, is a *path*. A closed trail, i. e., a trail that starts and ends with the same vertex, is a *circuit*. A circuit, where each inner vertex is visited only once, is a *cycle*. In a graph $G = (V, E)$, a *Hamiltonian path* from u to v is a path of length $|V| - 1$ from u to v and a *Hamiltonian tour* is a cycle of length $|V|$. Let $[n]$ denote the set $\{1, 2, \dots, n\}$, where n is an integer.

We call a complete graph $G = (V, E)$ with cost function $c : E \to \mathbb{Q}^+$ *metric*, if the edge costs satisfy the triangle inequality $c(\{u, v\}) \leq c(\{u, w\}) + c(\{w, v\})$ for any pairwise distinct vertices $u, v, w \in V$.

The *metric traveling salesman problem*, ΔTSP, is the problem of finding a minimum-cost Hamiltonian tour in a complete metric graph. The *metric minimum-cost Hamiltonian path problem* in complete graphs, where the two end vertices are fixed, is called ΔHPP_2.

Given a graph $G = (V, E)$ and two vertices u and v in G, then we define $G + \{u, v\}$ as $(V, E \cup \{\{u, v\}\})$. In a graph, a vertex is *odd* or *even*, if its degree is odd or even.

3 A Win/Win Strategy for ΔTSP and ΔHPP_2

In this section, we provide an improved analysis of a simple algorithm that combines the two well-known algorithms from [10] and [16]. The algorithm is exactly that from [8]. For completeness, we state this algorithm here.

We first bound the costs of the matchings involved in the algorithm. Let Opt_P and Opt_C be optimal solutions for the ΔHPP_2 and the ΔTSP, respectively.

Algorithm 1. Path and Cycle [8]

```
Input: A complete graph $G = (V, E)$, a metric cost function $c : E \to \mathbb{Q}^+$, and two
    vertices $s$ and $t$.
 1: Compute a minimum spanning tree $T$ in $G$.
 2: Compute a minimum perfect matching $M_C$ on the odd vertices of $T$ in $G$.
 3: Compute a minimum perfect matching $M_P$ on the odd vertices of the multigraph
    $T + \{s, t\}$ in $G$.
 4: Compute an Eulerian tour $\text{Eul}_C$ in the multigraph $T \cup M_C$ and an Eulerian path
    $\text{Eul}_P$ in the multigraph $T \cup M_P$.
 5: Shorten $\text{Eul}_C$ and $\text{Eul}_P$ to a Hamiltonian tour $H_C$ and a Hamiltonian path $H_P$,
    respectively.
Output: $H_C$ and $H_P$.
```

Lemma 1.

$$c(M_P) + c(M_C) \leq \min\{c(\mathrm{Opt}_P), c(\mathrm{Opt}_C)\}$$

Proof. First we show that $c(M_P)+c(M_C) \leq c(\mathrm{Opt}_P)$ holds. Let P be an optimal Hamiltonian path from s to t in G and let $v_1, v_2, \ldots, v_k$ be the odd vertices of T. Let us assume without loss of generality that they are in the order as they appear in Opt_P. It is clear that k is even. Then we define the matching M'_C as the set of edges $\{v_i, v_{i+1}\}$, where i is odd. Analogously, M'_P is the matching containing the edges $\{v_j, v_{j+1}\}$, where j is even. Additionally, M'_P contains $\{s, v_1\}$ if $v_1 \neq s$ and $\{v_k, t\}$ if $v_k \neq t$. Observe that M'_C is a perfect matching on the odd vertices of T and M'_P is a perfect matching on the odd vertices of $T + \{s, t\}$. Since M'_C and M'_P are disjoint, due to the triangle inequality $c(M'_P) + c(M'_C) \leq c(\mathrm{Opt}_P)$. Since M_P and M_C are minimal, $c(M_P) \leq c(M'_P)$ and $c(M_C) \leq c(M'_C)$.

Now we show that $c(M_P) + c(M_C) \leq c(\mathrm{Opt}_C)$. Note that M_C is a minimum-cost perfect matching of $v_1, v_2, \ldots, v_k$. By Christofides' analysis, we have $c(M_C) \leq c(\mathrm{Opt}_\mathrm{C})/2$: due to the triangle inequality, the cycle formed by $v_1, v_2, \ldots, v_k$ in the order as these vertices appear in an optimal Hamiltonian tour Opt_C is not more expensive than Opt_C itself. Since this cycle has two disjoint perfect matchings, the cheaper one has a cost of at most half of the cycle's cost. An analogous analysis shows

$$c(M_P) \leq c(\mathrm{Opt}_\mathrm{C})/2. \tag{1}$$

The only difference is the set of vertices that forms the minimum cost perfect matching which is composed of the odd vertices from $T + \{s, t\}$. □

Let $\alpha := c(H_P)/c(\mathrm{Opt}_\mathrm{P})$ be the approximation ratio for the computed Hamiltonian path and let $\beta := c(H_C)/c(\mathrm{Opt}_\mathrm{C})$ be the approximation ratio for the computed Hamiltonian tour for a given input G, c, s, t, where H_P and H_C are the solutions of Algorithm 1. Furthermore, we determine a value p from the costs of intermediate graphs in Algorithm 1 as

$$p := \max\{c(T), c(M_P) + c(M_C), 1.5c(M_P)\}.$$

We will show in the following, that p forms a lower bound on the cost of an optimal solution for $\Delta\mathrm{HPP}_2$.

Let α' be the value such that $c(H_P) = \alpha' p$. Thus, we can determine the value of α', whereas we do not know the value α. We will use α' as a parameter that determines a guarantee for the achieved approximation ratios for ΔTSP and $\Delta\mathrm{HPP}_2$.

Lemma 2. *For any input of Algorithm 1,*

$$(2\alpha - 2)c(\mathrm{Opt}_P) \leq (2\alpha' - 2)p \leq c(\mathrm{Opt}_C) \leq (3 - \alpha')p \leq (3 - \alpha)c(\mathrm{Opt}_P).$$

Proof. We first show that $p \leq c(\mathrm{Opt}_\mathrm{P})$ holds. Since Opt_P is a spanning tree, $c(T) \leq c(\mathrm{Opt}_\mathrm{P})$. Due to Lemma 1, also $c(M_P) + c(M_C) \leq c(\mathrm{Opt}_\mathrm{P})$. Due to the analysis of Hoogeveen's algorithm in [16], $c(M_P) \leq 2c(\mathrm{Opt}_\mathrm{P})/3$ and thus

$3c(M_P)/2 \leq c(\mathrm{Opt_P})$. Therefore, also the maximum of the three values is at most $c(\mathrm{Opt_P})$.

Since p is a lower bound on $c(\mathrm{Opt_P})$,

$$(2\alpha - 2)c(\mathrm{Opt}_P) = 2c(H_P) - 2c(\mathrm{Opt_P}) \leq 2c(H_P) - 2p = (2\alpha' - 2)p,$$

which shows the first inequality of the lemma.

We continue the proof by showing $(2\alpha' - 2)p \leq c(\mathrm{Opt}_C)$. Since $c(H_P) = \alpha' p \leq c(T) + c(M_P)$,

$$(2\alpha' - 2)p \leq 2(c(T) + c(M_P)) - 2p \leq 2(c(T) + c(M_P)) - 2c(T) = 2c(M_P).$$

Now the second inequality follows because of (1).

For the third inequality $c(\mathrm{Opt}_C) \leq (3 - \alpha')p$, we note that $c(H_C) \leq c(T) + c(M_C)$. Due to the definition of p, $c(M_C) + c(M_P) \leq p$ and $c(T) \leq p$. Therefore,

$$c(T) + c(M_C) \leq 2p - c(M_P).$$

Since $c(M_P) \geq c(H_P) - c(T)$ holds,

$$c(M_P) \geq \alpha' p - c(T) = \left(\alpha' - \frac{c(T)}{p}\right) p \geq (\alpha' - 1)p.$$

Therefore we get

$$c(H_C) \leq 2p - (\alpha' - 1)p = (3 - \alpha')p. \tag{2}$$

The last inequality follows, since

$$(3 - \alpha')p = 3p - \alpha' p = 3p - c(H_P) \leq 3c(\mathrm{Opt_P}) - \alpha c(\mathrm{Opt_P}) = (3 - \alpha)c(\mathrm{Opt_P}).$$

□

As a first consequence of Lemma 2, we can relate the actual approximation ratio α that A achieved for the given input for $\Delta\mathrm{HPP}_2$ to that for $\Delta\mathrm{TSP}$, β.

Theorem 1. *For any input of A,*

$$\beta \leq \min\left\{1.5, \frac{1}{\alpha - 1} - \frac{1}{2}\right\} \quad \textit{and} \quad \alpha \leq \min\left\{\frac{5}{3}, \frac{1}{\beta + 1/2} + 1\right\}.$$

Proof. Due to the analysis of Christofides and Hoogeveen, we can bound β and α from above by 1.5 and 5/3. For the second bound of β we note that, due to Lemma 2 and (2), $c(H_C) \leq (3 - \alpha)c(\mathrm{Opt_P})$. Since Lemma 2 also states that $c(\mathrm{Opt_C}) \geq (2\alpha - 2)c(\mathrm{Opt_P})$, we get

$$\beta = \frac{c(H_C)}{c(\mathrm{Opt}_C)} \leq \frac{(3 - \alpha)c(\mathrm{Opt}_P)}{(2\alpha - 2)c(\mathrm{Opt}_P)} = \frac{3 - \alpha}{2\alpha - 2} = \frac{2 - (\alpha - 1)}{2(\alpha - 1)} = \frac{1}{\alpha - 1} - \frac{1}{2}. \tag{3}$$

The remaining statement of the theorem now follows immediately, since $\beta \leq 1/(\alpha - 1) - 1/2$ implies $\alpha \leq 1/(\beta + 1/2) + 1$. □

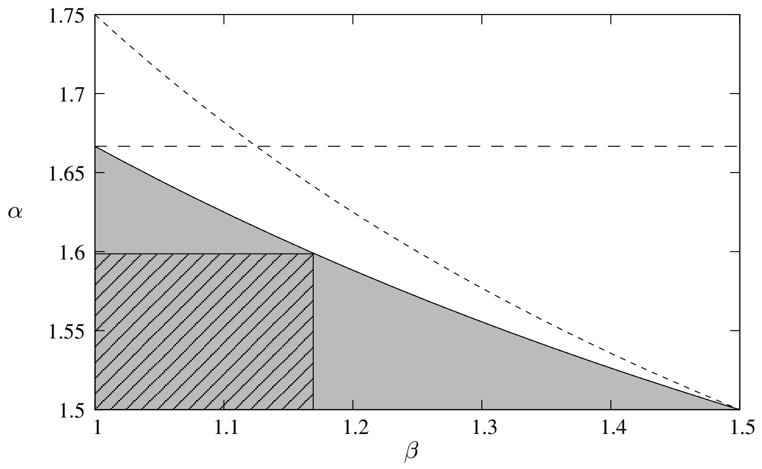


Fig. 1. The gray area describes the set S of all combinations of the approximation ratios α and β for the ΔHPP$_2$ and the ΔTSP achieved by Algorithm 1. The solid line describes the upper bound on the approximation ratios achieved by that algorithm. The dashed lines represent the upper bounds on the approximation ratios proven in [8]. The hatched area contains the set of possible pairs of solutions given that α' coincides with the highest value of α within the area.

Figure 1 gives a graphical representation of the theorem. The set S of all valid combinations of approximation ratios achieved by Algorithm 1 is represented as the gray area.

The following corollary follows from Theorem 1 by setting $\alpha = 5/3$.

Corollary 1. *Any worst-case instance for Hoogeveen's algorithm for* ΔHPP$_2$ *allows us to compute an optimal Hamiltonian cycle in* G*.*

Theorem 1 described properties that belong to the core of the relation between the two problems. Now we will change the focus and describe how to use the revealed relations in order to guarantee improved approximations according to parameters that we can measure, namely α'. More precisely, we determine the approximation ratios according to the spanning tree T and the matchings M_P and M_C. Let $\delta = p - c(T)$. Note that $\delta \geq 0$ and $\alpha \leq \alpha'$ holds.

Theorem 2. *For any input of* A*,*

$$\beta \leq \min\{1.5, 1/(\alpha' - 1) - 1/2\} - \delta/c(\mathrm{Opt_C}).$$

Proof. Analogous to the proof of Theorem 1, $c(H_C) \leq c(\mathrm{Opt_C}) \cdot (1/(\alpha' - 1) - 1/2)$ and $c(H_C) \leq c(\mathrm{Opt_C}) \cdot 1.5$. In this analysis, however, we estimated $c(T)$ by $c(\mathrm{Opt_P})$. The cost of the actual solution H_C is at least δ cheaper than we estimated previously. Therefore, the claim of the theorem follows. □

The effect of Theorem 2 is depicted in the hatched area in Figure 1.

4 Implications of the Win/Win Strategy

In this section, we classify hard input instances for the ΔHPP$_2$. To this end, similar to [15] we combine Algorithm 1 and a variant of the well-known tree-doubling algorithm for the ΔTSP, namely Algorithm 2, which enables us to restrict the class of hard input instances.

Algorithm 2. Tree Doubling

```
Input: A complete graph $G = (V, E)$, a metric cost function $c : E \to \mathbb{Q}^+$, and two
    vertices $s$ and $t$.
 1: Compute a minimum spanning tree $T$ in $G$.
 2: Let $P_{st}$ be the unique path in $T$ that connects $s$ and $t$.
 3: Find an Eulerian tour $\mathrm{Eul}_P$ in the multi-graph $T + (T - P_{st})$.
 4: Shorten $\mathrm{Eul}_P$ to a Hamiltonian path $H_P$.
Output: $H_P$.
```

In particular, we focus on the distance of s and t in G. Let A be the algorithm that runs both Algorithm 1 and Algorithm 2. The output of A is the cycle H_C from Algorithm 1 and the path H_P that is the smaller one of the two computed Hamiltonian paths. We introduce $\tilde{\alpha}$ and $\tilde{\alpha}'$ similar to α and α', but with a slightly extended meaning: these values are based on A instead of Algorithm 1. For simplicity, we assume both algorithms involved in A to use the same spanning tree T.

Theorem 3. *For any input of A,*

$$\begin{aligned}(2\tilde{\alpha} - 3)c(\mathrm{Opt}_P) \le (2\tilde{\alpha}' - 2)p - c(\mathrm{Opt}_P) \le c(\{s,t\}) \le c(P_{st})\\ \le (2 - \tilde{\alpha}')p \le (2 - \tilde{\alpha})c(\mathrm{Opt}_P).\end{aligned}$$

Proof. The first and the last inequality hold, similar to Lemma 2, since $c(H_P) = \tilde{\alpha}c(\mathrm{Opt_P}) = \tilde{\alpha}'p$ and $p \le c(\mathrm{Opt_P})$.

For the second inequality, note that any Hamiltonian path from s to t can be made a Hamiltonian cycle by adding the edge $\{s,t\}$. Therefore, $c(\mathrm{Opt}_C) \le c(\mathrm{Opt}_P) + c(\{s,t\})$ and thus, applying Lemma 2, $(2\tilde{\alpha}' - 2)p \le c(\mathrm{Opt}_P) + c(\{s,t\})$, which implies the second inequality.

The third inequality $c(\{s,t\}) \le c(P_{st})$ trivially holds due to the triangle inequality.

For the fourth inequality, we first have to analyze Algorithm 2. The algorithm is correct because in $T + (T - P_{st})$, all vertices but s and t have an even degree, which ensures the existence of Eul_P. Due to the triangle inequality, $c(H_P) \le c(\mathrm{Eul}_P)$. Therefore, $c(H_P) \le c(T + (T - P_{st}))$.

By the definitions of $\tilde{\alpha}'$ and p, we have $\tilde{\alpha}'p \le 2c(T) - c(P_{st})$ and thus the fifth inequality follows:

$$c(P_{st}) \le 2c(T) - \tilde{\alpha}'p \le 2p - \tilde{\alpha}'p.$$

□

Theorem 3 reveals several properties of hard input instances. For instance, by setting $\tilde{\alpha}$ to 5/3 in Theorem 3, we conclude that in each worst-case instance for ΔHPP$_2$, $c(P_{st}) = c(\{s,t\}) = c(\mathrm{Opt_P})/3$ holds. This means that according to Theorem 1, adding the edge $\{s,t\}$ to an optimal Hamiltonian path from s to t yields an optimal Hamiltonian tour in the same graph.

Furthermore, since P_{st} is a part of T, we can take into account the number of edges in P_{st}. Let γ be the value such that $c(P_{st}) = \gamma c(\mathrm{Opt_P})$. In other words γ is the fraction of $c(\mathrm{Opt_P})$ that is formed by P_{st}.

Theorem 4. *Suppose that there are k or fewer edges in P_{st}. Then there exists an algorithm that achieves an approximation ratio of*

$$(3-\tilde{\alpha})\left(\frac{1}{\tilde{\alpha}-1}-\frac{1}{2}\right)+\left(1-\frac{2}{k}\right)\gamma$$

for ΔHPP$_2$.

Proof. Let $e = \{u,v\}$ be the edge of maximal cost in P_{st} such that the four vertices are in the order s,u,v,t within P_{st}. Given the Eulerian cycle Eul_C from Algorithm 1, we remove e from Eul_C and add the two edges $\{s,u\}$ and $\{v,t\}$. The resulting graph has an Eulerian path from s to t. Let H'_p be that tour shortened to a Hamiltonian path. Then the cost of H'_P is at most

$$c(T)+c(M_C)-c(e)+(c(P_{st})-c(e)).$$

Since there are at most k edges in P_{st}, $c(e) \geq c(P_{st})/k$. The value of β is based on the cost of H_C, which is $c(T)+c(M_C)$. Therefore, Theorem 1 implicitly states that $c(T)+c(M_C) \leq (1/(\tilde{\alpha}-1)-1/2)c(\mathrm{Opt_C})$ holds and we can bound the cost of H'_P from above by

$$\begin{aligned} c(H'_P) &\leq \left(\frac{1}{\tilde{\alpha}-1}-1/2\right)c(\mathrm{Opt_C})+\left(1-\frac{2}{k}\right)c(P_{st}) \\ &\leq \left(\frac{1}{\tilde{\alpha}-1}-1/2\right)(3-\tilde{\alpha})c(\mathrm{Opt_P})+\left(1-\frac{2}{k}\right)\gamma c(\mathrm{Opt_P}). \end{aligned}$$

The last inequality holds because of Lemma 2. Dividing this value by $c(\mathrm{Opt_P})$ yields the claimed approximation ratio. □

Note that in the special case that $\tilde{\alpha} = 5/3$, Theorem 4 together with Theorem 3 implies that the cost of the computed Hamiltonian path is at most

$$\left(\frac{5}{3}-\frac{2}{3k}\right)\cdot c(\mathrm{Opt}_P).$$

5 Combined Hard Input Instances

In this section, we show that the analysis of Theorem 1 is tight. To this end, we construct a class of graphs that can be adapted to any choice of $1.5 < \hat{\alpha} < 5/3$,

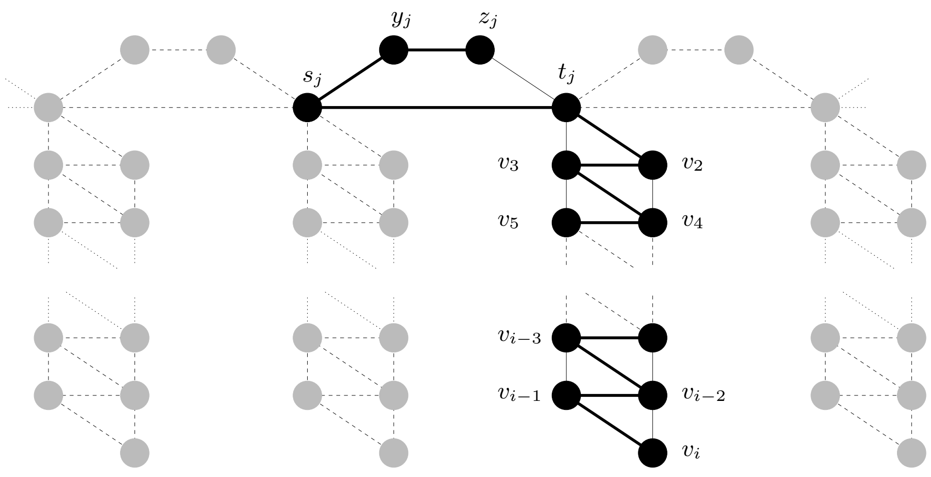


Fig. 2. The graph $G_{\hat{\alpha},i,k}$. The bold lines form a part of the spanning tree T.

where $\hat{\alpha}$ is the aimed-for lower bound on the approximation ratio achieved for ΔHPP$_2$. We define $\hat{\beta} := \max\{1.5, 1/(\hat{\alpha}-1)-1/2\}$, the guaranteed upper bound for ΔTSP according to Theorem 1. Hence, we aim for a lower bound $\hat{\beta}$ on the achieved approximation ratio for ΔTSP.

The basic building blocks of the graphs are the well known hard input instances for Christofides' algorithm from [10] (also described, e. g., in the textbook [17]) and for Hoogeveen's algorithm from [16].

Let $i \geq 4$ be an even number. Then we construct the graph $H_{i,\rho} = (V_i, E_{i,\rho})$, where $V_i = \{v_1, v_2, \ldots, v_i\}$ and ρ is a value that depends on the specific pair of bounds that we aim for. We specify $E_{i,\rho}$ by determining the edges of cost ρ/i. All remaining edges have the cost of the shortest path between the corresponding vertices. We say that two vertices are connected, if they are connected by an edge of cost ρ/i. For any $j \in [i-1]$, v_j and v_{j+1} are connected. Furthermore, for any $j \in [i-2]$, v_j and v_{j+2} are connected.

Now, for $k \in \mathbb{N}$, we construct a graph $G_{\hat{\alpha},i,k}$ with $n = 1+(i+2)k$ vertices (k copies of $H_{i,\rho}$ and $2k+1$ additional vertices) as depicted in Figure 2.

For each $j = 1, 2, \ldots, k$, we create a cycle $s_j y_j z_j t_j s_j$ such that each edge of the cycle has cost 1. The remaining two edges between these vertices are of cost 2. To each vertex t_j, we attach a copy of $H_{i,\rho}$ with $\rho = \frac{5-3\hat{\alpha}}{\hat{\alpha}-3/2}$ such that $t_j = v_1$. Now we join all k components such that for $j \in [k-1]$, $t_j = s_{j+1}$. Again, all remaining edges of the resulting graph cost as much as the shortest path between the corresponding vertices. The end vertices are $s = s_1$ and $t = t_j$.

Theorem 5. *For each $3/2 < \hat{\alpha} < 5/3$ and each $\varepsilon > 0$, there are integers i and k such the combined result of Algorithm 1, Algorithm 2, and Theorem 4 is not $(\hat{\alpha}-\varepsilon)$-approximative for ΔHPP$_2$ and not $(\hat{\beta}-\varepsilon)$-approximative for ΔTSP for the input $G_{\hat{\alpha},i,k}$.*

6 Conclusion

We have shown a tight bound of the combined approximation ratio of Hoogeveen's algorithm and Christofides' algorithm and for any input, we provided a pair of approximation ratios that is guaranteed to be achieved. We revealed a strong relation between the two problems and characterized properties of hard instances. These properties might be helpful in order to find an improved algorithm for $\Delta\mathrm{HPP}_2$ or ΔTSP. Since the described properties of hard input instances are very specific, the results of this paper show that for most of the practical input instances, we can guarantee better approximation ratios than in the worst-case.

Acknowledgement

The author would like to thank Hans-Joachim Böckenhauer for helping to improve the presentation.

References

1. Abu-Khzam, F.N., Collins, R.L., Fellows, M.R., Langston, M.A., Suters, W.H., Symons, C.T.: Kernelization algorithms for the vertex cover problem: Theory and experiments. In: Arge, L., Italiano, G.F., Sedgewick, R. (eds.) Proc. of the 6th Workshop on Algorithmic Engineering and Experiments (ALENEX 2004), pp. 62–69. Society for Industrial and Applied Mathematics (2004)
2. Archetti, C., Bertazzi, L., Speranza, M.G.: Reoptimizing the traveling salesman problem. Networks 42(3), 154–159 (2003)
3. Ausiello, G., Escoffier, B., Monnot, J., Paschos, V.T.: Reoptimization of minimum and maximum traveling salesman's tours. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 196–207. Springer, Heidelberg (2006)
4. Böckenhauer, H.J., Forlizzi, L., Hromkovič, J., Kneis, J., Kupke, J., Proietti, G., Widmayer, P.: On the approximability of TSP on local modifications of optimally solved instances. Algorithmic Oper. Res. 2(2), 83–93 (2007)
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. Theor. Comput. Sci. 285(1), 3–24 (2002)
6. Böckenhauer, H.J., Hromkovič, J., Mömke, T., Widmayer, P.: On the hardness of reoptimization. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 50–65. Springer, Heidelberg (2008)
7. Böckenhauer, H.J., Hromkovič, J., Seibert, S.: Stability of approximation. In: Gonzalez, T.F. (ed.) Handbook of Approximation Algorithms and Metaheuristics. Chapman & Hall/CRC Computer and Information Science Series, ch. 31. Chapman & Hall/CRC (2007)
8. 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)

9. Böckenhauer, H.J., Komm, D.: Reoptimization of the metric deadline TSP. Journal of Discrete Algorithms 8(1), 87–100 (2010)
10. Christofides, N.: Worst-case analysis of a new heuristic for the travelling salesman problem. Tech. Rep. 388, Graduate School of Industrial Administration, Carnegie-Mellon University (1976)
11. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science. Springer, New York (1999)
12. Eppstein, D.: Paired approximation problems and incompatible inapproximabilities. In: Charikar, M. (ed.) Proc. of the 21st Annual ACM-SIAM Symp. on Discrete Algorithms (SODA 2010), pp. 1076–1086. Society for Industrial and Applied Mathematics (2010)
13. Fellows, M.R.: Blow-ups, win/win's, and crown rules: Some new directions in FPT. In: Bodlaender, H.L. (ed.) WG 2003. LNCS, vol. 2880, pp. 1–12. Springer, Heidelberg (2003)
14. Forlizzi, L., Hromkovič, J., Proietti, G., Seibert, S.: On the stability of approximation for Hamiltonian path problems. Algorithmic Oper. Res. 1(1), 31–45 (2006)
15. Guttmann-Beck, N., Hassin, R., Khuller, S., Raghavachari, B.: Approximation algorithms with bounded performance guarantees for the clustered traveling salesman problem. Algorithmica 28(4), 422–437 (2000)
16. Hoogeveen, J.A.: Analysis of Christofides' heuristic: some paths are more difficult than cycles. Oper. Res. Lett. 10(5), 291–295 (1991)
17. Hromkovič, J.: Algorithmics for Hard Problems. Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics. Texts in Theoretical Computer Science. An EATCS Series. Springer, Berlin (2003)
18. Niedermeier, R.: Invitation to Fixed Parameter Algorithms. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, USA (2006)
19. Papadimitriou, C.H., Vempala, S.: On the approximability of the traveling salesman problem. Combinatorica 26(1), 101–120 (2006)
20. Prieto, E., Sloper, C.: Either/or: using vertex cover structure in designing FPT-algorithms—the case of k-Internal Spanning Tree. In: Dehne, F.K.H.A., Sack, J.R., Smid, M.H.M. (eds.) WADS 2003. LNCS, vol. 2748, pp. 474–483. Springer, Heidelberg (2003)
21. Sahni, S., Gonzalez, T.F.: P-complete approximation problems. J. ACM 23(3), 555–565 (1976)
22. Vassilevska, V., Williams, R., Woo, S.L.M.: Confronting hardness using a hybrid approach. In: Proc. of the 17th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA 2006), pp. 1–10. Society for Industrial and Applied Mathematics, New York (2006)
23. West, D.B.: Introduction to Graph Theory, 2nd edn. Prentice Hall Inc., Upper Saddle River (2000)

An Automata-Theoretical Characterization of Context-Free Trace Languages[⋆]

Benedek Nagy[1] and Friedrich Otto[2]

[1] Department of Computer Science, Faculty of Informatics
University of Debrecen, 4032 Debrecen, Egyetem tér 1., Hungary
nbenedek@inf.unideb.hu
[2] Fachbereich Elektrotechnik/Informatik, Universität Kassel
34109 Kassel, Germany
otto@theory.informatik.uni-kassel.de

Abstract. We present a characterization of the class of context-free trace languages in terms of cooperating distributed systems (CD-systems) of stateless deterministic restarting automata with window size 1 that are governed by an external pushdown store.

1 Introduction

In [10] we studied cooperating distributed systems (CD-systems) of stateless deterministic restarting automata that have a read/write window of size 1. Although the restarting automata of this type have a severely restricted expressive power, we obtain a device that is surprizingly expressive by combining several such automata into a CD-system. These systems accept a class of semi-linear languages that contains all rational trace languages [10]. In fact, we derived a characterization of the rational trace languages in terms of a particular class of these CD-systems. Further, the class of languages that are accepted by these CD-systems is closed under union, product, Kleene star, commutative closure, and disjoint shuffle, but it is not closed under intersection with regular languages, complementation, or ε-free morphisms. In addition, for these CD-systems the emptiness and the finiteness problems are easily solvable, while the regularity, the inclusion, and the equivalence problems are undecidable in general [11].

Here we extend these CD-systems by an external pushdown store that is used to determine the successor of the current automaton. When the active automaton performs a delete operation, then one of its successor automata is chosen based on the symbol deleted and on the topmost symbol on this pushdown store. In addition, after the successor has been chosen the pushdown content is modified by either erasing the topmost symbol, or by replacing it by a symbol or a word of

[⋆] This work was supported by grants from the Balassi Intézet Magyar Ösztöndíj Bizottsága (MÖB) and the Deutsche Akademischer Austauschdienst (DAAD). The first author was also supported by the TÁMOP 4.2.1/B-09/1/KONV-2010-0007 project, which is implemented through the New Hungary Development Plan, co-financed by the European Social Fund and the European Regional Development Fund.

length 2. Essentially such a system can be interpreted as a traditional pushdown automaton, in which the operation of reading an input symbol has been replaced by a stateless deterministic R(1)-automaton. Hence, not the first symbol is necessarily read, but some symbol that can be reached by this automaton by moving across a prefix of the current input word. In this way our CD-systems can be interpreted as pushdown automata with *translucent letters*. Other variants of pushdown automata that do not simply read their input sequentially from left to right have been studied before. For example, in [4] pushdown automata are considered that can reverse their input.

2 CD-Systems of Stateless Deterministic R(1)-Automata Governed by an External Pushdown

Stateless types of restarting automata were introduced in [8]. Here we are only interested in the most restricted form of them, the *stateless deterministic* R-*automaton* of window size 1. A *stateless deterministic* R(1)-*automaton* is a one-tape machine that is described by a 5-tuple $M = (\Sigma, ¢, \$, 1, \delta)$, where Σ is a finite alphabet, the symbols $¢, \$ \notin \Sigma$ serve as markers for the left and right border of the work space, respectively, the size of the *read/write window* is 1, and $\delta : \Sigma \cup \{¢, \$\} \to \{\mathsf{MVR}, \mathsf{Accept}, \varepsilon\}$ is the (partial) *transition function*. There are three types of transition steps: *move-right steps* (MVR), which shift the window one step to the right, combined *rewrite/restart steps* (denoted by ε), which delete the content u of the window, thereby shortening the tape, and place the window over the left end of the tape, and *accept steps* (Accept), which cause the automaton to halt and accept. Finally we use the notation $\delta(a) = \emptyset$ to express the fact that the function δ is undefined for the symbol a. Some additional restrictions apply in that the sentinels ¢ and \$ must not be deleted, and that the window must not move right on seeing the \$-symbol.

A *configuration* of M is described by a pair (α, β), where either $\alpha = \varepsilon$ (the empty word) and $\beta \in \{¢\} \cdot \Sigma^* \cdot \{\$\}$ or $\alpha \in \{¢\} \cdot \Sigma^*$ and $\beta \in \Sigma^* \cdot \{\$\}$; here $\alpha\beta$ is the current content of the tape, and it is understood that the head scans the first symbol of β. A *restarting configuration* is of the form $(\varepsilon, ¢w\$)$, where $w \in \Sigma^*$; to simplify the notation a restarting configuration $(\varepsilon, ¢w\$)$ is usually simply written as $¢w\$$. By $\vdash_M$ we denote the single-step computation relation of M, and $\vdash_M^*$ denotes the reflexive transitive closure of $\vdash_M$.

The automaton M proceeds as follows. Starting from an initial configuration $¢w\$$, the window moves right until a configuration of the form $(¢x, uy\$)$ is reached such that $\delta(u) = \varepsilon$. Now the latter configuration is transformed into the restarting configuration $¢xy\$$. This sequence of computational steps, which is called a *cycle*, is expressed as $w \vdash_M^c xy$. A computation of M now consists of a finite sequence of cycles that is followed by a tail computation, which consists of a sequence of move-right operations possibly followed by an accept step. An input word $w \in \Sigma^*$ is *accepted* by M, if the computation of M which starts with the initial configuration $¢w\$$ finishes by executing an accept step. By $L(M)$ we denote the language consisting of all words accepted by M.

If $M = (\Sigma, ¢, \$, 1, \delta)$ is a stateless deterministic $\mathsf{R}(1)$-automaton, then we can partition its alphabet Σ into four disjoint subalphabets:

$$\begin{array}{ll} (1.)\ \Sigma_1 = \{\, a \in \Sigma \mid \delta(a) = \mathsf{MVR}\,\}, & (3.)\ \Sigma_3 = \{\, a \in \Sigma \mid \delta(a) = \mathsf{Accept}\,\}, \\ (2.)\ \Sigma_2 = \{\, a \in \Sigma \mid \delta(a) = \varepsilon\,\}, & (4.)\ \Sigma_4 = \{\, a \in \Sigma \mid \delta(a) = \emptyset\,\}. \end{array}$$

It has been shown in [10] that the language $L(M)$ can be characterized as

$$L(M) = \begin{cases} \Sigma^*, & \text{if } \delta(¢) = \mathsf{Accept}, \\ (\Sigma_1 \cup \Sigma_2)^* \cdot \Sigma_3 \cdot \Sigma^*, & \text{if } \delta(¢) = \mathsf{MVR} \text{ and } \delta(\$) \neq \mathsf{Accept}, \\ (\Sigma_1 \cup \Sigma_2)^* \cdot ((\Sigma_3 \cdot \Sigma^*) \cup \{\varepsilon\}), & \text{if } \delta(¢) = \mathsf{MVR} \text{ and } \delta(\$) = \mathsf{Accept}. \end{cases}$$

Cooperating distributed systems (CD-systems) of restarting automata were introduced and studied in [9]. Here we study an extended variant of the CD-systems of stateless deterministic $\mathsf{R}(1)$-automata of [10].

A *pushdown CD-system of stateless deterministic* $\mathsf{R}(1)$*-automata*, PD-CD-$\mathsf{R}(1)$-system for short, consists of a CD-system of stateless deterministic $\mathsf{R}(1)$-automata and an external pushdown store. Formally, it is defined as a tuple $\mathcal{M} = (I, \Sigma, (M_i, \sigma_i)_{i\in I}, \Gamma, \bot, I_0, \delta)$, where

- I is a finite set of indices,
- Σ is a finite input alphabet,
- for all $i \in I$, $M_i = (\Sigma, ¢, \$, 1, \delta_i)$ is a stateless deterministic $\mathsf{R}(1)$-automaton on Σ, and $\sigma_i \subseteq I$ is a non-empty set of possible successors for M_i,
- Γ is a finite pushdown alphabet,
- $\bot \notin \Gamma$ is the bottom marker of the pushdown store,
- $I_0 \subseteq I$ is the set of initial indices, and
- $\delta : (I \times \Sigma \times (\Gamma \cup \{\bot\})) \to 2^{I\times(\Gamma\cup\{\bot\})^*}$ is the successor relation. For each $i \in I$, $a \in \Sigma$, and $A \in \Gamma$, $\delta(i, a, A)$ is a subset of $\sigma_i \times \Gamma^{\leq 2}$, and $\delta(i, a, \bot)$ is a subset of $\sigma_i \times (\bot \cdot \Gamma^{\leq 2})$.

A *configuration* of $\mathcal{M}$ is a triple $(i, ¢w\$, \alpha)$, where $i \in I$ is the index of the active component automaton M_i, the word $¢w\$$ ($w \in \Sigma^*$) is a restarting configuration of M_i, and the word $\alpha \in \bot \cdot \Gamma^*$ is the current content of the pushdown store with the first symbol of α at the bottom and the last symbol of α at the top. For $w \in \Sigma^*$, an *initial configuration* of $\mathcal{M}$ on input w has the form $(i_0, ¢w\$, \bot)$ for any $i_0 \in I_0$, and an *accepting configuration* has the form $(i, \mathsf{Accept}, \bot)$.

The *single-step computation relation* $\Rightarrow_{\mathcal{M}}$ that $\mathcal{M}$ induces on the set of configurations is defined by the following three rules, where $i \in I$, $w \in \Sigma^*$, $\alpha \in \bot \cdot \Gamma^*$, $A \in \Gamma$, and, for each $i \in I$, $\Sigma_1^{(i)}$ and $\Sigma_2^{(i)}$ are the subsets of Σ according to the above definition that correspond to the automaton M_i:

$$\begin{array}{lll} (1) & (i, ¢w\$, \alpha A) \Rightarrow_{\mathcal{M}} (j, ¢w'\$, \alpha\eta) & \text{if } \exists u \in {\Sigma_1^{(i)}}^*, a \in \Sigma_2^{(i)}, v \in \Sigma^* \text{ such that} \\ & & w = uav, w' = uv, \text{ and } (j, \eta) \in \delta(i, a, A); \\ (2) & (i, ¢w\$, \bot) \Rightarrow_{\mathcal{M}} (j, ¢w'\$, \bot\eta) & \text{if } \exists u \in {\Sigma_1^{(i)}}^*, a \in \Sigma_2^{(i)}, v \in \Sigma^* \text{ such that} \\ & & w = uav, w' = uv, \text{ and } (j, \bot\eta) \in \delta(i, a, \bot); \\ (3) & (i, ¢w\$, \bot) \Rightarrow_{\mathcal{M}} (i, \mathsf{Accept}, \bot) & \text{if } \exists u \in {\Sigma_1^{(i)}}^*, a \in \Sigma_3^{(i)}, v \in \Sigma^* \text{ such that} \\ & & w = uav, \text{ or } w \in {\Sigma_1^{(i)}}^* \text{ and } \delta_i(\$) = \mathsf{Accept}. \end{array}$$

By $\Rightarrow_{\mathcal{M}}^*$ we denote the *computation relation* of $\mathcal{M}$, which is simply the reflexive and transitive closure of the relation $\Rightarrow_{\mathcal{M}}$. The language $L(\mathcal{M})$ accepted by $\mathcal{M}$ consists of all words for which $\mathcal{M}$ has an accepting computation, that is,

$$L(\mathcal{M}) = \{\, w \in \Sigma^* \mid \exists i_0 \in I_0 \, \exists i \in I : (i_0, ¢w\$, \bot) \Rightarrow_{\mathcal{M}}^* (i, \mathsf{Accept}, \bot) \,\}.$$

A $\mathsf{PD\text{-}CD\text{-}R(1)}$-system $\mathcal{M} = (I, \Sigma, (M_i, \sigma_i)_{i \in I}, \Gamma, \bot, I_0, \delta)$ is called a *one-counter CD-system of stateless deterministic* $\mathsf{R(1)}$*-automata* ($\mathsf{OC\text{-}CD\text{-}R(1)}$-system for short), if $|\Gamma| = 1$, that is, if there is only a single pushdown symbol in addition to the bottom marker $\bot$. By $\mathcal{L}(\mathsf{PD\text{-}CD\text{-}R(1)})$ we denote the class of languages that are accepted by $\mathsf{PD\text{-}CD\text{-}R(1)}$-systems, and $\mathcal{L}(\mathsf{OC\text{-}CD\text{-}R(1)})$ is the class of languages accepted by $\mathsf{OC\text{-}CD\text{-}R(1)}$-systems.

Example 1. Let $L = \{\, a^n v \mid v \in \{b, c\}^*, |v|_b = |v|_c = n, n \geq 0 \,\}$. As $L \cap a^* \cdot b^* \cdot c^* = \{\, a^n b^n c^n \mid n \geq 0 \,\}$ is not context-free, we see that L itself is not context-free, either. Further, there is no regular sublanguage of L that is letter-equivalent to L. Hence, L is not accepted by any $\mathsf{stl\text{-}det\text{-}local\text{-}CR\text{-}R(1)}$-system (see [10]). However, we claim that L is accepted by the $\mathsf{OC\text{-}CD\text{-}R(1)}$-system $\mathcal{M} = (I, \Sigma, (M_i, \sigma_i)_{i \in I}, \Gamma, \bot, I_0, \delta)$ that is defined as follows:

– $I = \{a, b, c, +\}$, $\Sigma = \{a, b, c\}$, and $\Gamma = \{C\}$,
– M_a, M_b, M_c, and M_+ are defined by the following transition functions:

$$\begin{array}{llll}
\delta_a(¢) = \mathsf{MVR}, & \delta_b(¢) = \mathsf{MVR}, & \delta_c(¢) = \mathsf{MVR}, & \delta_+(¢) = \mathsf{MVR}, \\
\delta_a(a) = \varepsilon, & \delta_b(b) = \varepsilon, & \delta_c(c) = \varepsilon, & \delta_+(\$) = \mathsf{Accept}, \\
 & \delta_b(c) = \mathsf{MVR}, & \delta_c(b) = \mathsf{MVR}, &
\end{array}$$

– $\sigma_a = \{a, b\}$, $\sigma_b = \{c\}$, $\sigma_c = \{b, +\}$, $\sigma_+ = \{+\}$, and $I_0 = \{a, +\}$, and
– δ is defined as follows:

$$\begin{array}{ll}
(1)\ \delta(a, a, \bot) = \{(a, \bot C), (b, \bot C)\}, & (3)\ \delta(b, b, C) = \{(c, C)\}, \\
(2)\ \delta(a, a, C) = \{(a, CC), (b, CC)\}, & (4)\ \delta(c, c, C) = \{(b, \varepsilon), (+, \varepsilon)\},
\end{array}$$

and for all other tripels, δ yields the empty set.

The automaton M_+ just accepts the empty word, while M_a deletes the first letter, if it is an a; otherwise, it gets stuck. The automaton M_b reads across c's and deletes the first b it encounters, and analogously, M_c reads across b's and deletes the first c it encounters. Thus, we see from the successor sets that $\mathcal{M}$ can only accept certain words of the form $a^m v$ such that $v \in \{b, c\}^*$. However, when M_a deletes an a, then a symbol C is pushed onto the pushdown store, and when M_c deletes a c, then a symbol C is popped from the pushdown store. As M_b and M_c work alternatingly, this means that the same number of b's and c's are deleted. Thus, if M is to accept, then $|v|_b = |v|_c = n$ holds for some $n \geq 0$.

If $m < n$, then after deleting the first m occurrences of b and c, the pushdown store only contains the bottom marker $\bot$, and then $\mathcal{M}$ gets stuck as seen from the definition of δ. On the other hand, if $m > n$, then the pushdown still contains some occurrences of the symbol C when the word $a^m v$ has been erased completely. Hence, in this situation $\mathcal{M}$ does not accept, either. Finally, if $m = n$, then after erasing the last occurrence of c, also the last occurrence of the symbol C is popped from the pushdown store, and then M_+ accepts starting from the configuration $(+, ¢\$, \perp)$. Hence, we see that $L(\mathcal{M}) = L$ holds.

Thus, already the language class $\mathcal{L}(\mathsf{OC\text{-}CD\text{-}R(1)})$ contains a language that is neither context-free nor accepted by any stl-det-local-CD-R(1)-system. Next we will show that the class of languages that are accepted by the latter type of CD-systems is contained in $\mathcal{L}(\mathsf{OC\text{-}CD\text{-}R(1)})$.

Proposition 1. $\mathcal{L}(\mathsf{stl\text{-}det\text{-}local\text{-}CD\text{-}R(1)}) \subsetneq \mathcal{L}(\mathsf{OC\text{-}CD\text{-}R(1)})$.

Proof. Let $\mathcal{M} = ((M_i, \sigma_i)_{i\in I}, I_0)$ be a stl-det-local-CD-R(1)-system, and let $L = L(\mathcal{M})$. We obtain a OC-CD-R(1)-system $\mathcal{M}' = (I, \Sigma, (M_i, \sigma_i)_{i\in I}, \emptyset, \perp, I_0, \delta)$, where Σ is the tape alphabet of $\mathcal{M}$, by defining the transition function δ as follows for all $i \in I$:

$$\begin{aligned} \delta(i, a, \perp) &= \{\, (j, \perp) \mid j \in \sigma_i \,\} && \text{for all } a \in \Sigma_2^{(i)},\\ \delta(i, a, \perp) &= \emptyset && \text{for all } a \in \Sigma \smallsetminus \Sigma_2^{(i)}. \end{aligned}$$

Then there is a one-to-one correspondence between the accepting computations of $\mathcal{M}$ and the accepting computations of $\mathcal{M}'$. Thus, $L(\mathcal{M}') = L$. This yields the announced inclusion. Its properness follows from the previous example. $\square$

On the other hand, PD-CD-R(1)-systems accept all context-free languages.

Proposition 2. $\mathsf{CFL} \subsetneq \mathcal{L}(\mathsf{PD\text{-}CD\text{-}R(1)})$.

Proof. Let $L \subseteq \Sigma^+$ be a context-free language. Then there exists a context-free grammar $G = (V, \Sigma, S, P)$ in quadratic Greibach normal form for L, that is, for each production $(A \to r) \in P$, the right-hand side r is of the form $r = a\alpha$, where $a \in \Sigma$ and $\alpha \in V^{\leq 2}$. In addition, we can assume that the start symbol S does not occur on the right-hand side of any production. The standard construction of a pushdown automaton from a context-free grammar yields a pushdown automaton $\mathcal{A}$ without ε-moves that, given a word $w \in \Sigma^+$ as input, simulates a left-most G-derivation of w from S. In analogy to this construction we build a PD-CD-R(1)-system $\mathcal{M} = (I, \Sigma, (M_i, \sigma_i)_{i\in I}, V, \perp, \{S\}, \delta)$, where $I = V \cup \{+\}$, the stateless deterministic R(1)-automata M_A $(A \in V)$ and M_+ are defined as follows:

$$\begin{aligned} &\delta_+(¢) = \mathsf{MVR}, && \delta_A(¢) = \mathsf{MVR}, \\ &\delta_+(\$) = \mathsf{Accept}, && \delta_A(a) = \varepsilon, \qquad \text{if there exists } \gamma \in V^{\leq 2} : (A \to a\gamma) \in P, \end{aligned}$$

the sets of successors are defined by $\sigma_A = \sigma_+ = I$ for all $A \in V$, and the successor relation δ is defined as follows, where $A \in V$ and $a \in \Sigma$:

$$\begin{aligned} (1)\ \delta(S, a, \perp) &= \{\, (+, \perp) \mid (S \to a) \in P \,\} \\ &\cup \{\, (B, \perp B) \mid (S \to aB) \in P \,\} \\ &\cup \{\, (B, \perp CB) \mid (S \to aBC) \in P \,\}, \\ (2)\ \delta(A, a, A) &= \{\, (B, \varepsilon) \mid B \in V \smallsetminus \{S\} \text{ and } (A \to a) \in P \,\} \\ &\cup \{\, (+, \varepsilon) \mid (A \to a) \in P \,\} \\ &\cup \{\, (B, B) \mid (A \to aB) \in P \,\} \\ &\cup \{\, (B, CB) \mid (A \to aBC) \in P \,\}, \end{aligned}$$

and δ yields the empty set for all other values. Then, for all $w \in \Sigma^*$ and all $a \in \Sigma$,

$$\begin{aligned} wa \in L \text{ iff } & S \Rightarrow_G^+ wA \Rightarrow_G wa \\ \text{iff } & (S, ¢wa\$, \bot) \Rightarrow_{\mathcal{M}}^* (A, ¢a\$, \bot A) \Rightarrow_{\mathcal{M}} (+, ¢\$, \bot) \Rightarrow_{\mathcal{M}} (+, \mathsf{Accept}, \bot). \end{aligned}$$

Hence, it follows that $L(\mathcal{M}) = L(G) = L$.

If the given context-free language includes the empty word, then we can apply the above construction to the language $L \smallsetminus \{\varepsilon\}$. The resulting PD-CD-R(1)-system will accept this language. By adding the component $+$ to the set of initial components, we then obtain a PD-CD-R(1)-system for the language L. This yields the intended inclusion, which is proper by Example 1. □

Next we consider the so-called *one-counter automata* and the class of languages accepted by them. However, one finds several different non-equivalent definitions for one-counter automata in the literature. Here we take a definition that is equivalent to the one used by Jančar et. al. in [7] (see also [3]).

A pushdown automaton $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \bot, \delta, F)$ is called a *one-counter automaton* if $|\Gamma| = 1$, and if the bottom marker $\bot$ cannot be removed from the pushdown store. Thus, if C is the only symbol in Γ, then the pushdown content $\bot C^m$ can be interpreted as the integer m for all $m \geq 0$. Accordingly, the pop operation can be interpreted as the decrement -1. It is assumed in addition that the only other pushdown operations leave the value m unchanged or increase it by 1, that is, the pushdown is not changed or exactly one additional C is pushed onto it. Finally, $\mathcal{A}$ has to read an input symbol in each step, that is, it cannot make any ε-steps.

A word $w \in \Sigma^*$ is accepted by $\mathcal{A}$, if $(q_0, w, \bot) \vdash_{\mathcal{A}}^* (q, \varepsilon, \bot)$ holds for some final state $q \in F$. Observe that $\mathcal{A}$ can only distinguish between two states of its pushdown store: either the topmost symbol is C, which is interpreted by saying that *the counter is positive*, or it is the bottom marker $\bot$, which is interpreted as *the counter is zero*. By OCL we denote the class of languages that are accepted by one-counter automata. It is well-known that $\mathsf{REG} \subsetneq \mathsf{OCL} \subsetneq \mathsf{CFL}$ holds.

Proposition 3. $\mathsf{OCL} \subsetneq \mathcal{L}(\mathsf{OC\text{-}CD\text{-}R(1)})$.

Proof. Let $\mathcal{A} = (Q, \Sigma, \{C\}, q_0, \bot, \delta_{\mathcal{A}}, F)$ be a *one-counter automaton*, and let $L = L(\mathcal{A}) \subseteq \Sigma^*$ be the language it accepts. We simulate $\mathcal{A}$ through a OC-CD-R(1)-system $\mathcal{M} = (I, \Sigma, (M_i, \sigma_i)_{i \in I}, \{C\}, \bot, \{(q_0, =), +\}, \delta)$, where

1. $I = (Q \times \{=, >\}) \cup \{+\}$, and $\sigma_{(q,>)} = \sigma_{(q,=)} = \sigma_+ = I$ for all $q \in Q$,
2. the stateless deterministic R(1)-automata $M_{(q,>)}$, $M_{(q,=)}$ $(q \in Q)$, and M_+ are defined as follows:

$$\begin{array}{lll} (1)\ \delta_{(q,=)}(¢) & = \mathsf{MVR}, & \\ (2)\ \delta_{(q,=)}(a) & = \varepsilon & \text{if } \delta_{\mathcal{A}}(q, a, \bot) \text{ is defined}, \\ (3)\ \delta_{(q,>)}(¢) & = \mathsf{MVR}, & \\ (4)\ \delta_{(q,>)}(a) & = \varepsilon & \text{if } \delta_{\mathcal{A}}(q, a, C) \text{ is defined}, \\ (5)\ \delta_{+}(¢) & = \mathsf{MVR}, & \\ (6)\ \delta_{+}(\$) & = \mathsf{Accept}, & \end{array}$$

3. and the successor relation δ is defined as follows, where $q \in Q$, $a \in \Sigma$, and $i \in \{1,2\}$:

$$\begin{aligned}
(1)\ \delta((q,=),a,\bot) &= \{\, ((q',=),\bot) \mid (q',\bot) \in \delta_{\mathcal{A}}(q,a,\bot)\,\} \\
&\cup \{\, (+,\bot) \mid \exists q' \in F : (q',\bot) \in \delta_{\mathcal{A}}(q,a,\bot)\,\} \\
&\cup \{\, ((q',>),\bot C) \mid (q',\bot C) \in \delta_{\mathcal{A}}(q,a,\bot)\,\}, \\
(2)\ \delta((q,>),a,C) &= \{\, ((q',>),C^i) \mid (q',C^i) \in \delta_{\mathcal{A}}(q,a,C)\,\} \\
&\cup \{\, ((q',>),\varepsilon),((q',=),\varepsilon) \mid (q',\varepsilon) \in \delta_{\mathcal{A}}(q,a,C)\,\} \\
&\cup \{\, (+,\varepsilon) \mid \exists q' \in F : (q',\varepsilon) \in \delta_{\mathcal{A}}(q,a,C)\,\},
\end{aligned}$$

while δ yields the empty set for all other values.

Observe that each time $\mathcal{A}$ decreases its counter, $\mathcal{M}$ also decreases its counter, and in addition it has the option of activating the final component M_+, if the state entered is final. However, M_+ can only accept, if at that moment the input has been processed completely, and $\mathcal{M}$ only accepts if, in addition, the counter is zero. It follows that there is a one-to-one correspondence between the accepting computations of the one-counter automaton $\mathcal{A}$ and the system $\mathcal{M}$. Hence, we have $L(\mathcal{M}) = L(\mathcal{A}) = L$. This yields the intended inclusion, which is proper by Example 1. □

Definition 1. *A* PD-CD-R(1)*-system* $\mathcal{M} = (I, \Sigma, (M_i, \sigma_i)_{i \in I}, \Gamma, \bot, I_0, \delta)$ *is in* strong normal form *if it satisfies the following conditions, where, for all* $i \in I$, $\Sigma_1^{(i)}, \Sigma_2^{(i)}, \Sigma_3^{(i)}, \Sigma_4^{(i)}$ *is the partitioning of alphabet* Σ *for the automaton* M_i *as described above:*

$$\begin{aligned}
&(1)\ \exists\, i_+ \in I : \delta_{i_+}(¢) = \mathsf{MVR}, \delta_{i_+}(\$) = \mathsf{Accept},\ \textit{and}\ \Sigma_4^{(i_+)} = \Sigma; \\
&(2)\ \forall i \in I \smallsetminus \{i_+\} : \delta_i(¢) = \mathsf{MVR}, |\Sigma_2^{(i)}| = 1, \Sigma_3^{(i)} = \emptyset,\ \textit{and}\ \delta_i(\$) = \emptyset.
\end{aligned}$$

Thus, if $\mathcal{M}$ is in strong normal form, then it has a unique component M_{i_+} that can execute accept instructions, but it only accepts the empty word, while all other components each delete a single kind of letter. In particular, a word $w \in L(\mathcal{M})$ is first erased completely by executing $|w|$ many cycles, and then the empty word is accepted by activating component M_{i_+}. As OC-CD-R(1)-systems are a special type of PD-CD-R(1)-systems, this definition also applies to them. The following technical result shows that we can restrict our attention to PD-CD-R(1)-systems in strong normal form.

Lemma 1. *From a* PD-CD-R(1)*-system* $\mathcal{M}$ *one can construct a* PD-CD-R(1)*-system* $\mathcal{M}'$ *in strong normal form such that* $L(\mathcal{M}') = L(\mathcal{M})$*. If* $\mathcal{M}$ *is a* OC-CD-R(1)*-system, then* $\mathcal{M}'$ *can be constructed to be a* OC-CD-R(1)*-system, too.*

Our next result implies that all languages from the language class $\mathcal{L}$(PD-CD-R(1)) are semi-linear, that is, if $L \subseteq \Sigma^*$ belongs to this language class, and if $|\Sigma| = n$, then the Parikh image $\psi(L)$ of L is a semi-linear subset of $\mathbb{N}^n$.

Theorem 1. *Each language* $L \in \mathcal{L}$(PD-CD-R(1)) *contains a context-free sublanguage* E *such that* $\psi(L) = \psi(E)$ *holds. In fact, a pushdown automaton for* E *can be constructed effectively from a* PD-CD-R(1)*-system for* L.

Proof. Let $\mathcal{M} = (I, \Sigma, (M_i, \sigma_i)_{i \in I}, \Gamma, \perp, I_0, \delta)$ be a PD-CD-R(1)-system, and let $L = L(\mathcal{M})$. By Lemma 1 we can assume that $\mathcal{M}$ is in strong normal form, that is, there exists a unique index $+ \in I$ such that M_+ accepts the empty word, and for each other index $i \in I_r := I \smallsetminus \{+\}$, M_i does not execute any accept instructions and $|\Sigma_2^{(i)}| = 1$. To simplify the notation in the following we denote the letter $a \in \Sigma_2^{(i)}$ simply by $a_{(i)}$. From $\mathcal{M}$ we construct a pushdown automaton $P = (Q, \Sigma, \Gamma, q_0, \perp, \delta_P, F)$ as follows:

– $Q = I \cup \{q_0\}$, where q_0 is a new state, $F = \{+\}$, and
– the transition relation δ_P is defined as follows for all $i \in I_r$, $a \in \Sigma$, and $A \in \Gamma$:

$$\begin{aligned} &(1)\ \delta_P(q_0, \varepsilon, \perp) = \{\, (i, \perp) \mid i \in I_0 \,\},\\ &(2)\ \delta_P(i, a, \perp) \ = \{\, (j, \eta) \mid (j, \eta) \in \delta(i, a, \perp) \,\},\\ &(3)\ \delta_P(i, a, A) \ = \{\, (j, \alpha) \mid (j, \alpha) \in \delta(i, a, A) \,\}. \end{aligned}$$

Then $E = L(P)$ is a context-free language. It can be shown that it is a sublanguage of L that is letter-equivalent to L. □

In the proof of Theorem 1 the pushdown automaton P constructed from the given PD-CD-R(1)-system $\mathcal{M}$ is in fact a one-counter automaton if $\mathcal{M}$ is a OC-CD-R(1)-system. Thus, we also have the following result.

Corollary 1. *Each language $L \in \mathcal{L}$(OC-CD-R(1)) contains a sublanguage E that is a one-counter language such that $\psi(L) = \psi(E)$ holds. In fact, a one-counter automaton for E can be constructed effectively from a* OC-CD-R(1)*-system for L.*

As each context-free language has a semi-linear Parikh image, Theorem 1 has the following consequence.

Corollary 2. *The language class $\mathcal{L}$(PD-CD-R(1)) only contains semi-linear languages, that is, if a language L over $\Sigma = \{a_1, \ldots, a_n\}$ is accepted by a* PD-CD-R(1)*-system, then its Parikh image $\psi(L)$ is a semi-linear subset of $\mathbb{N}^n$.*

The semi-linear language $L = \{\, a^n b^n c^n \mid n \geq 0 \,\}$ does not contain a context-free sublanguage that is letter-equivalent to the language itself. Hence, Theorem 1 yields the following negative result.

Proposition 4. *The language $L = \{\, a^n b^n c^n \mid n \geq 0 \,\}$ is not accepted by any* PD-CD-R(1)*-system.*

The language $L_{pal} = \{\, wcw^R \mid w \in \{a, b\}^* \,\}$ is a context-free language that is not a one-counter language (see, e.g., [2]). As a context-free language it is accepted by some PD-CD-R(1)-system by Proposition 2, but based on Corollary 1 the following result can be shown.

Proposition 5. *The language $L_{pal} = \{\, wcw^R \mid w \in \{a, b\}^* \,\}$ is not accepted by any* OC-CD-R(1)*-system.*

3 Context-Free Trace Languages

Let Σ be a finite alphabet, and let D be a binary relation on Σ that is reflexive and symmetric, that is, $(a,a) \in D$ for all $a \in \Sigma$, and $(a,b) \in D$ implies that $(b,a) \in D$, too. Then D is called a *dependency relation* on Σ, and the relation $I_D = (\Sigma \times \Sigma) \smallsetminus D$ is called the corresponding *independence relation*. Obviously, the relation I_D is irreflexive and symmetric. The dependency relation D induces a binary relation $\equiv_D$ on Σ^* that is defined as the smallest congruence relation (with respect to the operation of concatenation) containing the set of pairs $\{ (ab,ba) \mid (a,b) \in I_D \}$. For $w \in \Sigma^*$, the congruence class of w mod $\equiv_D$ is denoted by $[w]_D$, that is, $[w]_D = \{ z \in \Sigma^* \mid w \equiv_D z \}$. These congruence classes are called *traces*, and the factor monoid $M(D) = \Sigma^*/\equiv_D$ is a *trace monoid*. In fact, $M(D)$ is the *free partially commutative monoid* presented by (Σ, D) (see, e.g., [6]). By φ_D we denote the morphism $\varphi_D : \Sigma^* \to M(D)$ that is defined by $w \mapsto [w]_D$ for all words $w \in \Sigma^*$.

We call a language $L \subseteq \Sigma^*$ a *rational trace language*, if there exists a dependency relation D on Σ such that $L = \varphi_D^{-1}(S)$ for a rational subset S of the trace monoid $M(D)$ presented by (Σ, D), that is, if there exist a trace monoid $M(D)$ and a regular language $R \subseteq \Sigma^*$ such that $L = \varphi_D^{-1}(\varphi_D(R)) = \bigcup_{w \in R} [w]_D$. By $\mathcal{LRAT}$ we denote the class of all rational trace languages. In [10] it is shown that $\mathcal{LRAT} \subsetneq \mathcal{L}(\mathsf{stl\text{-}det\text{-}local\text{-}CD\text{-}R(1)})$.

Here we are interested in more general trace languages. A language $L \subseteq \Sigma^*$ is called a *one-counter trace language*, if there exist a dependency relation D on Σ and a one-counter language $R \subseteq \Sigma^*$ such that $L = \varphi_D^{-1}(\varphi_D(R)) = \bigcup_{w \in R} [w]_D$. Analogously, a language $L \subseteq \Sigma^*$ is called a *context-free trace language*, if there exist a dependency relation D on Σ and a context-free language $R \subseteq \Sigma^*$ such that $L = \varphi_D^{-1}(\varphi_D(R)) = \bigcup_{w \in R} [w]_D$ [1,5]. By $\mathcal{LOC}(D)$ we denote the set of one-counter trace languages obtained from (Σ, D), and $\mathcal{LOC}$ is the class of all one-counter trace languages. Further, by $\mathcal{LCF}(D)$ we denote the set of context-free trace languages obtained from (Σ, D), and $\mathcal{LCF}$ is the class of all context-free trace languages. The next theorem states that all context-free trace languages are accepted by $\mathsf{PD\text{-}CD\text{-}R(1)}$-systems.

Theorem 2. *If D is a dependency relation on a finite alphabet Σ, then $\mathcal{LCF}(D) \subseteq \mathcal{L}(\mathsf{PD\text{-}CD\text{-}R(1)})$.*

Proof. Let $L \subseteq \Sigma^*$ be a context-free trace language, that is, there exists a context-free language R over Σ such that $L = \varphi_D^{-1}(\varphi_D(R))$. As R is context-free, there exists a grammar $G = (V, \Sigma, S, P)$ in quadratic Greibach normal form for $R' = R \smallsetminus \{\varepsilon\}$. From G we construct a $\mathsf{PD\text{-}CD\text{-}R(1)}$-system $\mathcal{M} = (I, \Sigma, (M_i, \sigma_i)_{i \in I}, V, \bot, I_0, \delta)$ as follows (cf. the proof of Proposition 2):

– $I = \{ (A,a) \mid A \in V, a \in \Sigma, \exists \gamma \in V^{\leq 2} : (A \to a\gamma) \in P \} \cup \{+\}$,
– $I_0 = \{ (S,a) \mid \exists \gamma \in V^{\leq 2} : (S \to a\gamma) \in P \} \cup \{ + \mid \varepsilon \in L \}$,
– the stateless deterministic $\mathsf{R(1)}$-automata $M_{(A,a)}$ ($(A,a) \in I$) and M_+ are defined as follows:

$$\begin{aligned}
\delta_+(¢) &= \mathsf{MVR}, & \delta_{(A,a)}(¢) &= \mathsf{MVR},\\
\delta_+(\$) &= \mathsf{Accept}, & \delta_{(A,a)}(a) &= \varepsilon,\\
& & \delta_{(A,a)}(b) &= \mathsf{MVR} \quad \text{for all } b \in \Sigma \text{ satisfying } (b,a) \in I_D,
\end{aligned}$$

– the sets of successor indices are defined as $\sigma_{(A,a)} = \sigma_+ = I$ for all $(A,a) \in I$,
– and the successor relation δ is defined as follows, where $A \in V$ and $a \in \Sigma$:

$$\begin{aligned}
\delta((S,a),a,\bot) = {} & \{ (+,\bot) \mid (S \to a) \in P \}\\
\cup {} & \{ ((B,b),\bot B) \mid (S \to aB) \in P, (B,b) \in I \}\\
\cup {} & \{ ((B,b),\bot CB) \mid (S \to aBC) \in P, (B,b) \in I \},\\
\delta((A,a),a,A) = {} & \{ ((B,b),\varepsilon) \mid B \in V \smallsetminus \{S\}, (B,b) \in I, \text{ and } (A \to a) \in P \}\\
\cup {} & \{ (+,\varepsilon) \mid (A \to a) \in P \}\\
\cup {} & \{ ((B,b),B) \mid (A \to aB) \in P, (B,b) \in I \}\\
\cup {} & \{ ((B,b),CB) \mid (A \to aBC) \in P, (B,b) \in I \},
\end{aligned}$$

and δ yields the empty set for all other values.

It can now be shown that $L(\mathcal{M}) = \varphi_D^{-1}(\varphi_D(R)) = \bigcup_{u \in R}[u]_D$. □

Thus, we have the hierarchy of language classes depicted in the diagram in Figure 1. As $\mathcal{L}(\mathsf{stl\text{-}det\text{-}local\text{-}CD\text{-}R(1)})$ contains the non-context-free language $\{ w \in \{a,b,c\}^* \mid |w|_a = |w|_b = |w|_c \geq 0 \}$, and as this class does not contain the one-counter language $\{ a^n b^n \mid n \geq 0 \}$, we see that $\mathcal{L}(\mathsf{stl\text{-}det\text{-}local\text{-}CD\text{-}R(1)})$ is incomparable under inclusion to the language classes OCL and CFL. From Proposition 5 we see that the class $\mathcal{L}(\mathsf{OC\text{-}CD\text{-}R(1)})$ is incomparable under inclusion to the language class CFL.

Let $\Sigma = \{a,b,c\}$, and let $L' = \{ wa^m \mid |w|_a = |w|_b = |w|_c \geq 1, m \geq 1 \}$. As shown in Example 4 of [11] the language L' is accepted by a stl-det-local-CD-R(1)-system. However, the following result can be derived.

$$\begin{tikzcd}
 & & \mathcal{L}(\mathsf{PD\text{-}CD\text{-}R(1)}) \\
 & \mathcal{LCF} \arrow[ur] & \mathcal{L}(\mathsf{OC\text{-}CD\text{-}R(1)}) \arrow[u] \\
\mathsf{CFL} \arrow[ur] & \mathcal{LOC} \arrow[u] \arrow[ur] & \mathcal{L}(\mathsf{stl\text{-}det\text{-}local\text{-}CD\text{-}R(1)}) \arrow[u] \\
\mathsf{OCL} \arrow[u] \arrow[ur] & \mathcal{LRAT} \arrow[u] \arrow[ur] & \\
\mathsf{REG} \arrow[u] \arrow[ur] & &
\end{tikzcd}$$

Fig. 1. Hierarchy of language classes accepted by various types of CD-R(1)-systems. Each arrow represents a proper inclusion, and classes that are not connected by a sequence of arrows are incomparable under inclusion.

Proposition 6. *For each dependency relation D on Σ and each context-free language $R \subseteq \Sigma^*$, $L' \neq \bigcup_{w \in R} [w]_D$.*

Together with Proposition 5 this result shows that all the inclusions in the diagram in Figure 1 are proper, and that classes that are not connected in that diagram are incomparable under inclusion.

Next we present a restricted class of PD-CD-R(1)-systems that accept exactly the context-free trace languages.

Definition 2. *Let $\mathcal{M} = (I, \Sigma, (M_i, \sigma_i)_{i \in I}, V, \perp, I_0, \delta)$ be a* PD-CD-R(1)*-system in strong normal form that satisfies the following condition:*

$$(*) \qquad \forall i, j \in I : \Sigma_2^{(i)} = \Sigma_2^{(j)} \text{ implies that } \Sigma_1^{(i)} = \Sigma_1^{(j)},$$

that is, if two component automata erase the same letter, then they also read across the same subset of Σ. With $\mathcal{M}$ we associate the binary relation

$$I_{\mathcal{M}} = \bigcup_{i \in I} (\Sigma_1^{(i)} \times \Sigma_2^{(i)}),$$

that is, $(a, b) \in I_{\mathcal{M}}$ if and only if there exists a component automaton M_i such that $\delta_i(a) = \mathsf{MVR}$ and $\delta_i(b) = \varepsilon$. Further, let $D_{\mathcal{M}} = (\Sigma \times \Sigma) \smallsetminus I_{\mathcal{M}}$.

Observe that the relation $I_{\mathcal{M}}$ is necessarily irreflexive, but it will in general not be symmetric. Now the following characterization can be established.

Theorem 3. *Let $\mathcal{M}$ be a* PD-CD-R(1)*-system over Σ satisfying condition $(*)$ above. If the associated relation $I_{\mathcal{M}}$ is symmetric, then $D_{\mathcal{M}}$ is a dependency relation on Σ, and $L(\mathcal{M}) \in \mathcal{L}CF(D_{\mathcal{M}})$. In fact, from $\mathcal{M}$ one can construct a pushdown automaton B over Σ such that $L(\mathcal{M}) = \bigcup_{u \in L(B)} [u]_{D_{\mathcal{M}}}$.*

If the given PD-CD-R(1)-system $\mathcal{M}$ is a OC-CD-R(1)-system, then the pushdown automaton B constructed in the proof above is actually a one-counter automaton. Observe that the system $\mathcal{M}$ constructed in the proof of Theorem 2 is in strong normal form, that it satisfies property $(*)$, and that the associated relation $I_{\mathcal{M}}$ coincides with the relation I_D, and hence, it is symmetric. Thus, Theorems 2 and 3 together yield the following characterization.

Corollary 3. (a) *A language $L \subseteq \Sigma^*$ is a one-counter trace language if and only if there exists a* OC-CD-R(1)*-system $\mathcal{M}$ in strong normal form satisfying condition $(*)$ such that the relation $I_{\mathcal{M}}$ is symmetric and $L = L(\mathcal{M})$.*

(b) *A language $L \subseteq \Sigma^*$ is a context-free trace language if and only if there exists a* PD-CD-R(1)*-system $\mathcal{M}$ in strong normal form satisfying condition $(*)$ such that the relation $I_{\mathcal{M}}$ is symmetric and $L = L(\mathcal{M})$.*

4 Concluding Remarks

As seen above the language $L = \{ a^n v \mid v \in \{b, c\}^*, |v|_b = |v|_c = n, n \geq 0 \}$ is accepted by a OC-CD-R(1)-system, while $L \cap a^* \cdot b^* \cdot c^* = \{ a^n b^n c^n \mid n \geq 0 \}$ is not accepted by any PD-CD-R(1)-system (Prop. 4). Hence, the language classes $\mathcal{L}$(OC-CD-R(1)) and $\mathcal{L}$(PD-CD-R(1)) are not closed under intersection with regular languages. In fact, they are not closed under intersection or complementation, either, but they are closed under union and under the operation of taking the commutative closure. However, it remains open whether these classes are closed under product, Kleene star, ε-free morphisms, inverse morphisms or reversal.

References

1. Aalbersberg, I., Rozenberg, G.: Theory of traces. Theoretical Computer Science 60, 1–82 (1988)
2. Autebert, J., Berstel, J., Boasson, L.: Context-free languages and pushdown automata. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, Word, Language, Grammar, vol. 1, pp. 111–174. Springer, Berlin (1997)
3. Berstel, J.: Transductions and Context-Free Languages. In: Teubner Studienbücher: Informatik, Teubner, Stuttgart (1979)
4. Bordihn, H., Holzer, M., Kutrib, M.: Input reversals and iterated pushdown automata: a new characterization of Khabbaz geometric hierarchy of languages. In: Calude, C.S., Calude, E., Dinneen, M.J. (eds.) DLT 2004. LNCS, vol. 3340, pp. 102–113. Springer, Heidelberg (2004)
5. Bertoni, A., Mauri, G., Sabadini, N.: Membership problems for regular and context-free trace languages. Information and Computation 82, 135–150 (1989)
6. Diekert, V., Rozenberg, G.: The Book of Traces. World Scientific, Singapore (1995)
7. Jančar, P., Moller, F., Sawa, Z.: Simulation problems for one-counter machines. In: Bartosek, M., Tel, G., Pavelka, J. (eds.) SOFSEM 1999. LNCS, vol. 1725, pp. 404–413. Springer, Heidelberg (1999)
8. Kutrib, M., Messerschmidt, H., Otto, F.: On stateless two-pushdown automata and restarting automata. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) Proc. of Automata and Formal Languages, AFL 2008, pp. 257–268. Computer and Automation Research Institute, Hungarian Academy of Sciences (2008)
9. Messerschmidt, H., Otto, F.: Cooperating distributed systems of restarting automata. Intern. J. Found. Comput. Sci. 18, 1333–1342 (2007)
10. Nagy, B., Otto, F.: CD-systems of stateless deterministic R(1)-automata accept all rational trace languages. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 463–474. Springer, Heidelberg (2010)
11. Nagy, B., Otto, F.: On CD-systems of stateless deterministic R-automata with window size one. Kasseler Informatikschriften, Fachbereich Elektrotechnik/Informatik, Universität Kassel handle/urn:nbn:de:hebis:34-2010042732682 (February 2010), https://kobra.bibliothek.uni-kassel.de/

Unambiguous UML Composite Structures: The OMEGA2 Experience

Iulian Ober and Iulia Dragomir

Université de Toulouse - IRIT
118 Route de Narbonne, 31062 Toulouse, France
{iulian.ober,iulia.dragomir}@irit.fr

Abstract. Starting from version 2.0, UML introduced hierarchical composite structures, which are a very expressive way of defining complex software architectures, but which have a very loosely defined semantics in the standard. In this paper we propose a set of consistency rules that ensure UML composite structures are unambiguous and can be given a precise semantics. Our primary application of the static consistency rules defined in this paper is within the OMEGA UML profile [6], but these rules are general and applicable to other hierarchical component models based on the same concepts, such as MARTE GCM or SysML. The rule set has been formalized in OCL and is currently used in the OMEGA UML compiler.

1 Introduction

Composite structures are the language elements used in UML (since version 2.0) for modelling hierarchically structured components and they are instrumental in capturing the architecture of complex systems. In particular, real-time systems often exhibit static, albeit complex, hierarchical topologies. For this reason, functional modelling languages dedicated to the real-time domain, such as ROOM [19], SDL [8], or architecture description languages, such as AADL [18], allow for hierarchical components.

This paper deals with the introduction of composite structures in the OMEGA UML profile [6], an executable profile dedicated to the formal specification and validation of real-time systems. Our main goal was to have an expressive and unambiguous set of constructs for modelling hierarchical structures, with an operational semantics that integrates smoothly in the existing execution model of OMEGA. Due to limited space, this paper concentrates on the static typing and consistency rules that form the basis of the operational semantics. The rules formulated here may be applied, beyond the scope of the OMEGA profile, to other component-based models such as SysML [14] or the Generic Component Model of MARTE [15].

The typing and consistency rules presented in this paper have been formalized in OCL and are used in the current version of the OMEGA UML compiler. We are now working on a formalization within a proof assistant (Isabelle/HOL [17]) in order to prove the type safety of models observing this rule set.

Related Work. The idea of UML composite structures is rooted in previously existing languages, notably ROOM [19] and SDL [8]. However, UML adds complexity with respect to previous models, e.g., by allowing explicit port behaviour specifications, multiple interfaces per port, typing of connectors with associations, etc. Many problems identified in this paper stem from the added complexity. Potential problems and ambiguities in UML composite structures have previously been discussed by other authors [13, 3]. In [3], Cuccuru et al. proposed a set of additional rules meant to further clarify the semantics of UML composite structures. While we fully subscribe to the solutions they propose, some issues remain unsolved, and the present paper is complementary to their solutions.

Structure of the Paper. §2 introduces the original OMEGA profile. §3 presents the main elements of UML 2.x composite structures. §4 presents the principles and rules introduced for disambiguating UML composite structures in the context of the extended OMEGA profile (henceforth named OMEGA2). In §5 we briefly discuss the principles of the operational semantics used in the OMEGA2/IFx simulation and model-checking platform. Finally, in §6 we discuss the OCL formalization of the rules and their application on a large example model for a space application, before concluding.

2 Overview of OMEGA UML

OMEGA UML is an executable profile of UML used for formal specification and validation of real-time systems. As the semantics of standard UML [16] is intentionally underspecified in order to preserve the generality of the language, one of the strong points of OMEGA UML for practical usability is its precise operational semantics, defined in [4]. Thanks to this common semantics, different tools implementing different analysis techniques can be applied to OMEGA models as it has been shown in [6]; in particular, our IFx toolset [12] allows to simulate and to verify OMEGA models by applying timed-automata model checking techniques [2].

The original OMEGA profile is based on a subset of UML 1.5 containing the main constructs of class diagrams (classes and their relationships, basic types, signals), state machines and actions. State machines are attached to classes and, together with operations, they define the behaviour of class instances. Actions are used for describing transition effects and operation bodies. The profile defines a concrete textual syntax for actions, which is compatible with the UML action metamodel and implements its main elements: object creation and destruction, operation calls, expression evaluation, variable assignment, signal output, etc. The profile also extends standard UML with elements necessary for the specification and verification of real-time systems. For more information, the reader is referred to [6, 4, 12].

In answer to requests coming from industrial users of the IFx toolset, we have undertaken an upgrade to the profile and the toolset to UML version 2. The main challenge was to give an operational semantics to composite structures and to find the static conditions under which this is possible.

3 UML Composite Structures

Composite structures are introduced in UML 2.x for the specification of "structures of interconnected elements that are created within an instance of a containing classifier". Composite structures are a powerful mechanism to increase the expressiveness and the readability of UML class models. They are used for specifying the initialization of complex object structures. This is particularly useful for real-time embedded systems, which are often structured as hierarchical block models, with classes having a fixed number of instances with predefined roles. Writing the initialization code for such object structures has indeed proved to be one of the most tedious tasks with the previous version of OMEGA.

An example of a composite structure identifying the model elements involved is given in Figure 1. A composite structure defines the structure of (instances of) a class in terms of inner components, also called *parts* (`b` in Figure 1) and of communication *connectors*, also called *links* (`c,d,e,f`). *Connectors* exist either between inner components (`d,e`), or between inner components and the outside environment of the composite structure (`c,f`). A connector has two end points; an end point can be either an inner component or a *port*. A connector can be the realization of an *association*, although this is not mandatory. UML introduces the following terminology for connectors: *delegation* connectors are connectors between a (port of a) part and a port of its containing composite structure (e.g., `c,f`) and *assembly* connectors are links between (ports of) two parts of the same composite structure (e.g., `d,e`).

A *port* represents an interaction point between an object and its environment. The allowed inbound requests that can travel through the port are specified by the *provided interface(s)* (e.g., `g`), while the outbound requests are specified by the *required interface(s)* (e.g., `h`).

For further technical details, the reader is referred to the UML standard [16].

Composite structures are a big evolution of the object-oriented paradigm, which is the basis of UML. Their implications on the semantic level are huge and not completely defined by the UML standard (see for example the problems

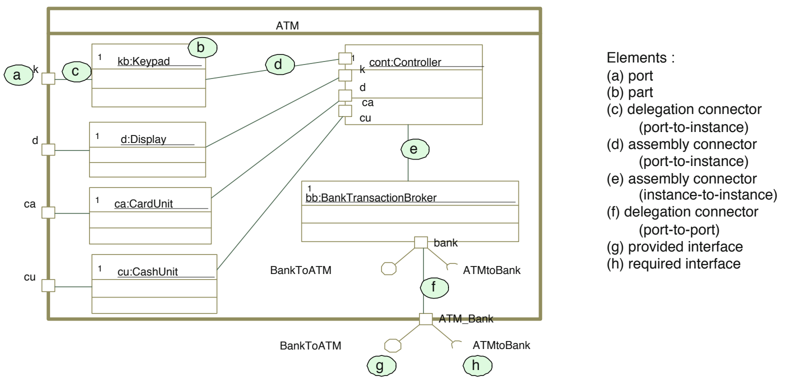


Fig. 1. Example of composite structure

outlined in [3]). For this reason, the OMEGA2 profile imposes some constraints on the usage of composite structures, described in the next section.

4 Unambiguous Composite Structures for OMEGA UML

In a composite structure, every connector links two entities, which can be either ports or components (parts). In both cases, the two entities are typed. In addition, the modeller can specify that the connector *realizes* an *association*. It is clear that, in general, connecting entities of arbitrary types does not make sense, and there should be clear compatibility rules (based on types, link direction, etc.) specifying what are the well formed structures. However, these type compatibility rules for connectors are not detailed in UML. The standard merely states that "the connectable elements attached to the ends of a connector must be compatible." and that "what makes connectable elements compatible is a semantic variation point." ([16] pp. 175-176). Various causes of ambiguity, such as the existence of several connectors starting from the same end-point are not even mentioned.

The principle of the OMEGA profile is to have a clear and coherent executable semantics for the chosen UML subset. Therefore, extending the profile to cover composite structures necessitates first to fully disambiguate the meaning of such structures, by setting well formedness constraints and by clarifying the run-time behaviour of the structures.

4.1 Typing Problems with Bidirectional Ports

In UML, ports are bidirectional, i.e. they can specify a set of allowed incoming requests (the *provided* interface) and a set of allowed outgoing requests (the *required* interface)[1]. This is represented in the model as follows: the port has a type, which is an arbitrary classifier (in practice, a class or an interface); all the interfaces that are directly or indirectly *realized* by the type of the port are considered to be *provided* interfaces. The *required* interfaces are those interfaces for which there exists a *Dependency* stereotyped with `<<Usage>>` between the port type (or one of its supertypes) and the respective interface(s). Figure 2-a shows a simple example of bidirectional port.

In Figure 2-a, the type of the port `port_0` is defined to be I (i.e., the provided interface, as said before). However, the fact that the port is bidirectional raises typing problems, as the port has to be treated as an entity of type I, and sometimes as an entity of type J. This is apparent in the following practical situations:

- When component A uses `port_0` to send out requests (e.g., by an action such as "`port_0.op2()`"), the type system has to ensure that the requests (here `op2`) conform to the required interface J. Thus, `port_0` has to be treated by the type system as an entity of type J, although its declared type is I.

[1] Here, incoming and outgoing are used from the point of view of the component owning the port.

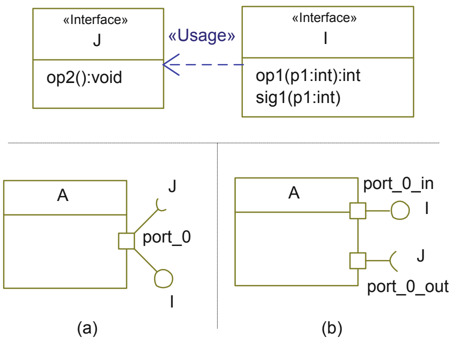


Fig. 2. Example of bidirectional port (a) and equivalent in OMEGA2 (b)

– When specifying the behaviour of `port_0` itself by a state machine[2], the state machine has to handle requests coming from both directions, i.e. requests conforming both to I and to J.

These typing inconsistencies are not addressed by the UML standard. When translating UML models to an implementation language (in our case, IF [1]) they raise homologous problems for the typing of the actual object that will represent the port.

While it is possible to give a general solution based on qualifying the types (I,J) with the corresponding directions (*in*, *out*) and on allowing the port entity to comply to multiple types, such a solution greatly complicates the type checking. For this reason, the solution we adopt here is to forbid bidirectional ports. This restriction is made without loss of expressive power, as any bidirectional port can be split in two unidirectional ports, like in the example in Figure 2-b, although it can be argued that it leads to less convenient models.

In OMEGA2 models, the convention used for modelling a port with a *required* interface J (such as `port_0_out` from Figure 2-b) is to declare J as the port's type and to stereotype the port with `<<reversed>>` (to distinguished it from a port *providing* J).[3]

4.2 Directionality Rules

For this section and the following ones, the discussion is based on the running example in Figure 3. The example is a composite A with two sub-components of types D and E, one using ports for communication (E) and one not (D). For both sub-components there are incoming links (links from port $pIJL$ of A) and outgoing links (links to ports rK and bak_rA_K of A).

Before discussing type compatibility issues for links, some simple directionality rules must be observed by well-formed structures:

[2] This is deemed possible by the UML 2.x standard [16], but without further detail.

[3] A similar convention is used by the IBM Rhapsody tool [7], which also supports the standard graphical symbol for *required interfaces* instead of the textual stereotype. For convenience, we use this representation in Figure 2-b and in the following.

Rule 1. *If a* delegation *link exists between two ports, the direction* (provided *or* required) *of the ports must be the same.*

Rule 2. *If an* assembly *link exists between two ports, one of the ports (the source) must be a* `<<reversed>>` *port (required) and the other (the destination) must be a provided port.*

Rule 3. *If a link is typed with an association, the direction of the association must be conform to the direction of the link (derived from the direction of the ports at the ends).*

The reason behind these rules can be easily seen on the model in Figure 3. Rule 1 forbids to put a connector for example between $pIJL$ and rK, since the direction of the connector would be ambiguous. Rule 3 forces the direction of a connector to be coherent with the direction of the realized association, like in the case of the link between d and rA_K (realizing association $itsK$).

4.3 Typing of Connectors

In programming or modelling languages, the purpose of the type system is to determine what are the operations that are applicable to an entity. By analogy, the purpose of a type system for composite structure connectors is to determine which requests (operation calls or signals) can be sent through each connector. In the following, we sketch the principles behind the OMEGA2 type system. Some supporting constructs (*interface groups*, *default delegation associations*) are defined first, followed by the definition of connector types and type-based consistency rules.

Interface groups. Let us note that it is sometimes necessary to declare several provided or required interfaces for one port (for example, $pIJL$ of A which provides interfaces I, J and L, see Figure 3). In UML, this is done by declaring a new interface that inherits from these interfaces and by using this new interface as the port type (IJL in Figure 3). However, such interfaces are artificial syntactic additions to the model, and they should not be taken in consideration by the link compatibility rules stated in the following. In our example, d and e only realize interfaces I and respectively J and L, so interface IJL is irrelevant for

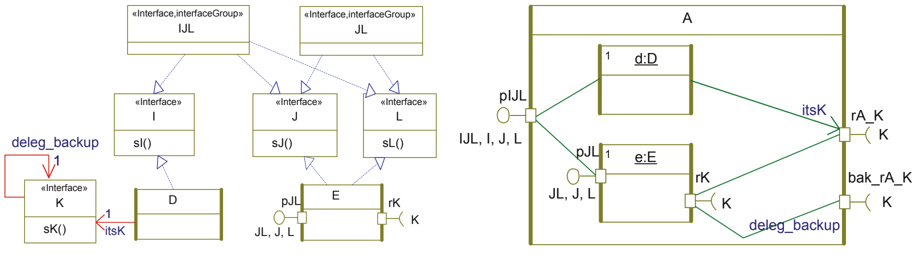


Fig. 3. Connection rules in composite structures (running example)

the semantics of the model. In OMEGA2, such interfaces must be stereotyped with `<<interfaceGroup>>` to distinguish them from meaningful ones, as shown in the upper part of Figure 3.

Default delegation associations. The default behaviour of a port is to forward requests from one side to the other, depending on its direction. The minimum information needed by the port is, for each provided/required interface, who the destination should be. For example in Figure 3, port $pIJL$ needs to know (and be able to refer to) the destination of requests belonging to interface I (here, d) and the destination of requests belonging to J or L (here, pJL of e). Similarly, rK needs to know that the destination of outgoing requests is by default rA_K.

It follows that, for each provided/required interface, the port has to possess an association designating to whom the port should forward requests belonging to that interface. In OMEGA2, every interface type I has by default an association called $deleg\_I$ pointing to itself, used for this purpose (for modelling convenience, the semantics considers they exist by default if they are omitted in the model). These associations are used to define the forwarding semantics of ports, described later on.

The dynamic type of a connector. The type of a connector determines what invocations (signals or operation calls) can travel through the connector and how do port behaviour descriptions refer to the connector. In general, in the case of a connector *originating*[4] from a port (i.e., not directly from a part), its type can be derived from the type of the entities situated at its two ends and does not necessarily need to be statically specified using an *association*. The following notion defines the dynamic type of the connector:

Definition 1 [**Set of transported interfaces**]. For a connector starting from a port, the *set of transported interfaces* is defined as the *intersection* between the two sets of interfaces provided/required at the two ends of the link.

As the ends of a link can be either ports or components, the meaning of provided/required interfaces is defined for each case:

– For a *Port*, the set of required/provided interfaces is the set containing the *Port*'s type and all its supertypes, minus all the interfaces stereotyped as `<<interfaceGroup>>`.
– For a component, the set of provided interfaces is the set of all interfaces directly or indirectly realized by the component's class. □

According to this definition, the set of transported interfaces for the links in Figure 3 are as follows[5]:

[4] According to link directionality.
[5] Link d to rA_K is not considered as it starts from a component and therefore must be statically typed (see Rule 5 later on).

– For link $pIJL$ to d the set is $\{I\}$.
– For link $pIJL$ to pJL the set is $\{J, L\}$.
– For link rK to rA_K the set is $\{K\}$.

4.4 Type Coherence Rules

Let us note that the link from $pIJL$ to d given as example above could have been statically typed with association *deleg_I*, because the set of transported interfaces $\{I\}$ contains only one element. However, in the general case when the derived set contains several interfaces (like for example the link between $pIJL$ and pJL which transports $\{J, L\}$), statically typing a link with an association is not necessary and may be restrictive.

If a static type is specified, it must be compatible with the dynamic type, as stated in the following rule:

Rule 4. *If a link outgoing from a port is statically typed with an association, then the association is necessarily directed (cf. rule 3) and the type pointed at by the association must belong to the* set of transported interfaces *for that link.*

On the example in Figure 3, rule 4 implies that, for example, if the link $pIJL$ to pJL is statically typed with an association then the association must point at either J or L.

While the type for a connector starting *from a port* does not need to be statically specified as it can be derived as shown before, if the connector starts directly *from a component* (and not from a port) then the static type *must* be specified:

Rule 5. *If a link originates in a component, then the link* must *be statically typed with an association, and the type of the entity at the other end of the link must be compatible with (i.e. be equal or a subtype of) the type at the corresponding end of the association.*

In Figure 3, only the link from d to rA_K is in this case; the link has indeed to be typed (here, with $itsK$) or otherwise the component would have no means to refer to it for communication.

Finally, a link is meaningful only if it can transport some requests:

Rule 6. *The set of transported interfaces of a link should not be void.*

4.5 Port Behaviour and Further Type-Based Rules

In OMEGA2, the default behaviour of a port is to forward requests from one side to the other, depending on the port's direction. Each request (signal or operation call) will be forwarded to a destination which depends on the interface to which the signal or operation belongs, using the default *deleg* associations described before. For example, the default forwarding behaviour of port $pIJL$ from Figure 3 can be described by the state machine in Figure 4-a[6].

[6] `deleg_I!sI` is the OMEGA2 syntax for the action of sending signal `sI` to the destination `deleg_I` (if the signal has formal parameters and no actual parameters are specified in the sending action, the actual values that will be sent are those received at the last *reception* – here the one that triggered the transition).

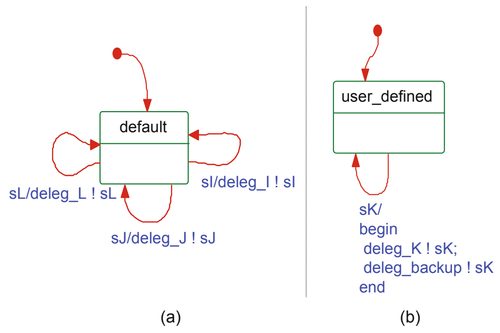


Fig. 4. (a) default state machine for port $pIJL$, (b) user-defined machine for port rK

The default behaviour is unambiguous only if for any interface the entity to which the corresponding *deleg* association points at is clear. Therefore, the following rule is necessary:

Rule 7. *If several* non-typed *connectors start from one port, then the sets of interfaces transported by each of these connectors have to be pairwise disjoint.*

Note that this rule does not forbid the case where a port is connected to n entities ($n > 1$) that provide or require the same interface: it merely says that in this case at least $n - 1$ connectors have to be explicitly typed with associations. In the example from Figure 3, port rK of e is in this situation: it has two links to two ports (rA_K and bak_rA_K), both typed with the same interface (K). According to Rule 7, one of the links has to be explicitly typed; here, the second one is statically typed with the association *deleg_backup*.

In addition, for completeness of the port behaviour, we require the following:

Rule 8. *The union of the sets of interfaces transported by each of the connectors* originating *from a port P must be equal to the set of interfaces provided/required by P.*

Applied for example to port $pIJL$ from Figure 3, this rule says that the two links originating from the port must transport, together, the entire set of interfaces provided by the port, i.e. $\{I, J, L\}$ (remember that IJL is an `<<interfaceGroup>>` and does not count in type checks).

The default port behaviour may be redefined by attaching a state machine to the port's type. In OMEGA2, this state machine may use the implicitly typed connectors (to which the behaviour may refer using the default *deleg* associations), as well as the explicitly typed connectors (via their defining association). In Figure 4-b we show an example of port behaviour for port rK (from Figure 3), which duplicates every sK signal on both the default connector (*deleg_K*, communicating with rA_K) and the secondary connector (*deleg_backup*, communicating with bak_rA_K).

5 Compiling Composite Structures to IFx

While the operational semantics of composite structures is outside the scope of this paper (which concentrates on structural well-formedness and typing), we briefly discuss here their translation into an executable model, since the existence of a feasible translation provides an empirical indication of the soundness of the rules stated before.

The OMEGA/IFx toolset [11] implements the OMEGA2 profile and provides simulation and model-checking functionality for OMEGA2 models. The toolset relies on the translation of models towards a simple language based on asynchronously communicating extended timed automata (IF [1]).

For the translation of composite structures, the main challenge was to fit the relatively complex, hierarchical UML modelling constructs into the simple and flat object space of IF automata. Our translation is based on the principle that the modelling elements involved in composite structures, namely ports and connectors, should be handled as first class language citizens. This means that we refrain from flattening the model during compilation and hard-wiring all the communication paths (something that is done in certain SDL compilers). Concretely, each port instance is implemented as an IF process instance (whose behaviour corresponds to the routing behaviour described in §4.5) and each connector is represented by attributes in the end-points (in ports or in components), corresponding to the association defining the connector (the default *deleg* association or the explicitly specified one). A UML composite structure diagram is thus used as an initialization scheme for instantiating components and ports and for creating links. A composite structure is therefore translated to a constructor (see [12] for a description of constructors in OMEGA).

6 Implementation and Evaluation

The rules presented in §4 have been formalised in OCL. While details are omitted here, the entire OCL rule set with explanations can be found in [5]. The OCL rule set can be evaluated over a UML model in standard XMI format using the Topcased OCL Environment [9], and is also used by the OMEGA UML compiler [11].

To validate the approach, we evaluated the rules on several complex models. The most complex example we used is a model of the solar wings deployment system of the ATV[7] provided by Astrium Space Transportation. The model features a 3-level hierarchical architecture with 37 classes (7 composite ones), 93 active objects at runtime and approximately 380 ports and 200 connectors.

The OCL formalization was applied on the model in order to test model compliance with the OMEGA2 profile and to search for modelling errors. Since the original model had not been built for simulation or verification, the first issue pointed out by the rules was that ports and connectors were untyped. The

[7] Automated Transfer Vehicle of the International Space Station, http://www.esa.int/atv

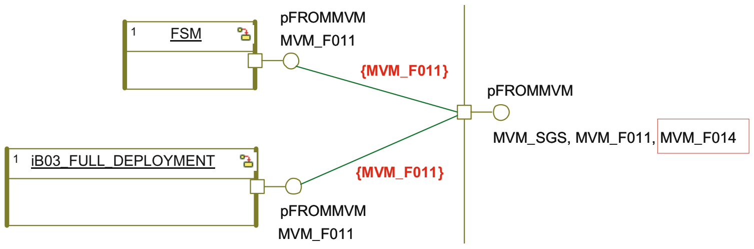


Fig. 5. Inconsistent port with respect to uniqueness and completeness rules

corrective action consisted in defining a total of 26 interfaces, and using them for specifying port contracts. Only a few ports in the original model were bidirectional and splitting them to unidirectional ports did not raise problems, resulting in a clearer model. The evaluation of the OCL rules yielded the inconsistent ports and connectors (cf. §4.1 and §4.2) which were either removed or redefined.

A second task was the verification of the uniqueness and completeness of ports, (cf. §4.3-§4.5). Approximately 20% of the evaluated ports were inconsistent with respect to rules 7 and 8. Figure 5 shows one such example, in which a port (of `iB03_FULL_DEPLOYMENT`) was incorrectly typed with interface `MVM_F011` instead of `MVM_F014`, resulting in routing in the outer port that is both ambiguous (for `MVM_F011`) and incomplete (for `MVM_F014`). This was detected using the OCL rules.

Finally, the corrected model was given as input to the OMEGA2 compiler and was simulated with the IFx toolset [11]. During simulation, deadlocks due to missing connectors or unhandled requests by ports were not found. This provides strong empirical evidence that, under the constraints of the rule set, the OMEGA2 type system is safe.

7 Conclusions

We presented an approach for defining an operational model of UML composite structures, by defining structural well-formedness and typing rules which are significantly more precise than the current UML standard [16], and by closing the relevant semantic variation points left open in the standard. The approach is based on :

– a set of static well-formedness rules, including type checking rules, which go beyond the rule sets that may be found in the literature, such as [3],
– dynamic typing of connectors based on a derived set of *transported interfaces*,
– a full definition of the default behaviour of *Ports*, and the means for defining port behaviour differing from the default.

The rule set is used by the type checker of the OMEGA UML compiler. In addition, the compiler goes all the way down to an operational implementation of composite structures, by translating OMEGA UML models (edited with any XMI 2.0 compatible UML editor) into IF models, for which a simulation and model-checking platform exists.

Experiments with realistic models (see §6) show that the rules are able to catch the most frequent modelling errors in composite structures. After correcting such errors, simulation and exhaustive state-space search with the IF model-checker show no cases of routing problems (deadlocks in ports due to missing links, unexpected requests not conforming to object interfaces, etc.).

Future work – towards type safety. The purpose of the rules defined in this paper is to ensure the type safety of OMEGA UML composite structures. We are currently working on a formalization of the structure of the OMEGA profile and of the rules described above within the Isabelle/HOL proof assistant [17], as a first step towards proving type safety. Within this context, type safety means that in composite structures obeying to the rules above, the following two properties hold: (1) a request (signal or method invocation) can always be deterministically routed by the intermediate ports up to a final destination object, and (2) from routing, a destination object can only receive requests that belong to its interface.

References

[1] Bozga, M., Graf, S., Ober, I., Ober, I., Sifakis, J.: The IF toolset. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 237–267. Springer, Heidelberg (2004)

[2] Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)

[3] Cuccuru, A., Gérard, S., Radermacher, A.: Meaningful composite structures. In: Busch, C., Ober, I., Bruel, J.-M., Uhl, A., Völter, M. (eds.) MODELS 2008. LNCS, vol. 5301, pp. 828–842. Springer, Heidelberg (2008)

[4] Damm, W., Josko, B., Pnueli, A., Votintseva, A.: A discrete-time UML semantics for concurrency and communication in safety-critical applications. Sci. Comput. Program. 55(1-3), 81–115 (2005)

[5] Dragomir, I., Ober, I.: Well-formedness and typing rules for UML composite structures (November 2010), `http://arxiv.org/abs/1010.6155`

[6] Hooman, J., Kugler, H., Ober, I., Votintseva, A., Yushtein, Y.: Supporting UML-based development of embedded systems by formal techniques. Software and System Modeling 7(2), 131–155 (2008)

[7] IBM. Rational rhapsody v7.5. reference manuals, `http://www.ibm.com/developerworks/rational/`

[8] ITU-T. Languages for telecommunications applications – Specification and Description Language (SDL). ITU-T Revised Recommendation Z.100 (1999)

[9] Topcased. Topcased toolset, `http://www.topcased.org`

[10] Nipkow, T., von Oheimb, D.: Java$_{light}$ is type-safe - definitely. In: POPL, pp. 161–170 (1998)

[11] Ober, I., Dragomir, I.: OMEGA2: A new version of the profile and the tools. In: 14th IEEE International Conference on Engineering of Complex Computer Systems, UML & AADL track, pp. 373–378. IEEE, Los Alamitos (2010)

[12] Ober, I., Graf, S., Ober, I.: Validating timed UML models by simulation and verification. STTT 8(2), 128–145 (2006)

[13] Oliver, I., Luukkala, V.: On UML's Composite Structure Diagram. In: 5th Workshop on System Analysis and Modelling (SAM) (June 2006)

[14] Object Management Group – Systems Modeling Language (SysML), v1.1.(2008), http://www.omg.org/spec/SysML/1.1/
[15] Object Management Group – UML profile for modeling and analysis of real-time and embedded systems (MARTE) (June 2008), http://www.omgmarte.org/Documents/Specifications/08-06-09.pdf
[16] Object Management Group – Unified Modeling Language (version 2.2) (February 2009), http://www.omg.org/spec/UML/2.2
[17] Paulson, L.C.: Isabelle - A Generic Theorem Prover (with a contribution by T. Nipkow). In: Isabelle. LNCS, vol. 828, Springer, Heidelberg (1994)
[18] SAE. Architecture analysis and design language (AADL) (November 2004), http://www.sae.org/technical/standards/AS5506/1
[19] Selic, B., Gullekson, G., Ward, P.T.: Real-Time Object-Oriented Modeling. Wiley Professional Computing, John Wiley (1994)

Comparing Linear Conjunctive Languages to Subfamilies of the Context-Free Languages

Alexander Okhotin[⋆]

Department of Mathematics, University of Turku, Turku FI–20014, Finland
Academy of Finland
alexander.okhotin@utu.fi

Abstract. Linear conjunctive grammars define the same family of languages as one-way real-time cellular automata (Okhotin, "On the equivalence of linear conjunctive grammars to trellis automata", *RAIRO ITA*, 2004), and this family is known to be incomparable to the context-free languages (Terrier, "On real-time one-way cellular array", *Theoret. Comput. Sci.*, 1995). This paper investigates subclasses of the context-free languages for possible containment in this class. It is shown that every visibly pushdown automaton (Alur, Madhusudan, "Visibly pushdown languages", *STOC 2004*) can be simulated by a one-way real-time cellular automaton, but already for LL(1) context-free languages and for one-counter DPDAs no simulation is possible.

1 Introduction

This paper contributes to the study of a family of formal languages notable for having two equivalent definitions coming from different lines of research. By the first definition, this is the family recognized by one of the simplest types of cellular automata: the *one-way real-time cellular automata*, also known as *trellis automata*, studied by Dyer [5], Čulík, Gruska and Salomaa [4], Ibarra and Kim [6], Terrier [17], and others. These automata work in real time (that is, make $n-1$ steps on an input of length n), and the next value of each cell is determined only by its own value and the value of its right neighbour.

The second definition is made in terms of formal grammars. Consider first the context-free grammars augmented with a conjunction operation in the rules. This model is called a *conjunctive grammar* [11], and, along with even more general *Boolean grammars* [13], they are known to preserve the main practical properties of the context-free grammars [15], have a greater expressive power, and offer a new field for theoretical studies [8,9,16]. Consider their restriction, in which concatenation can be taken only with terminal strings: such grammars are known as *linear conjunctive grammars* [11], and they are computationally equivalent to the above class of cellular automata [12].

An important progress in the study of this language family was made by Terrier [17], who proved a general lemma asserting non-representability of some

⋆ Supported by the Academy of Finland under grants 134860 and 218315.

languages, and used it to show that the concatenation of a certain linear context-free language L_0 with itself—obviously a context-free language—is not recognized by any trellis automaton. This showed that the family of linear conjunctive languages is not closed under concatenation and is incomparable with the context-free languages. However, the language $L_0 \cdot L_0$ constructed by Terrier is inherently ambiguous [14], and this still leaves open a possibility that some subclasses of the context-free languages can be simulated by trellis automata.

This paper continues the comparison between context-free grammars and trellis automata, by considering the following most important subfamilies of the context-free languages. These are the families defined by

– unambiguous context-free grammars;
– LR(1) context-free grammars (deterministic pushdown automata);
– LL(k) context-free grammars,
– deterministic counter automata,
– parenthesis grammars [10], balanced context-free grammars [2], as well as their recently introduced generalization: visibly pushdown automata [1].

In Section 3 it is demonstrated that every visibly pushdown automaton (VPDA) can be simulated by a trellis automaton. Note that VPDAs are deterministic only when executed sequentially, as per their definition, while, on the other hand, a trellis automaton attains its full power on parallel computations. Hence the deterministic computation of a VPDA has to be simulated nondeterministically: the trellis automaton does not know the state, in which the VPDA begins processing each substring, and so it calculates the result of the VPDA's computation for every possible initial state. These computed behaviours on substrings are gradually combined, until the behaviour on the entire string is obtained.

The second result, established in Section 4, is that the more potent known subfamilies of the context-free languages are already not contained in the linear conjunctive languages. This is done by constructing an LL(1) context-free language (which is at the same time recognized by a deterministic counter automaton), and proving that it is not linear conjunctive. The proof is carried out using the lemma of Terrier [17].

2 Linear Conjunctive Grammars and Trellis Automata

All families of formal languages considered in this paper are subsets of the family generated by *conjunctive grammars*, which directly extend the context-free grammars by allowing a conjunction of syntactical conditions in any rule.

Definition 1 (Okhotin [11]). *A conjunctive grammar is a quadruple $G = (\Sigma, N, P, S)$, in which: Σ and N are disjoint finite nonempty sets of terminal and nonterminal symbols, respectively; P is a finite set of grammar rules, each of the form*

$$A \to \alpha_1 \& \ldots \& \alpha_n, \tag{1}$$

with $A \in N$, $n \geqslant 1$ and $\alpha_1, \ldots, \alpha_n \in (\Sigma \cup N)^*$; $S \in N$ is a nonterminal designated as the start symbol.

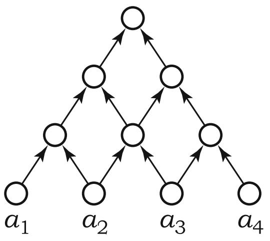
A rule (1) expresses that every terminal string that is generated by each α_i is hence generated by A. This understanding can be formalized in two ways. One definition uses language equations with nonterminal symbols as unknown languages, and using the least solution of the system

$$A = \bigcup_{A \to \alpha_1 \& \ldots \& \alpha_m \in P} \bigcap_{i=1}^{m} \alpha_i \quad (\text{for all } A \in N)$$

to define the languages generated by these nonterminals. The other definition employs term rewriting that generalizes Chomsky's string rewriting: according to it, a rule (1) allows rewriting A with a term $(\alpha_1 \& \ldots \& \alpha_n)$ over concatenation and conjunction, and furthermore it is allowed to rewrite a conjunction $(w \& \ldots \& w)$ of identical terminal strings with a single such string w.

The main facts about the conjunctive grammars (and the more general Boolean grammars [13] that further allow negation) are that, on the one hand, they can define some nontrivial non-context-free languages, such as $\{wcw \mid w \in \{a,b\}^*\}$ [11] and $\{a^{4^n} \mid n \geqslant 0\}$ [7], and on the other hand, they retain the key practical properties of the context-free grammars: in particular, parse trees and efficient parsing algorithms [14,15].

The family of conjunctive grammars has two important special cases. One of these cases are the *context-free grammars*, in which there is a unique conjunct in every rule, that is, every rule (1) has $n = 1$. The other case are the *linear conjunctive grammars*, in which every conjunct α in every rule (1) may contain at most one nonterminal symbol. Grammars obeying both restrictions are known as *linear context-free*.



The family of languages defined by linear conjunctive grammars is remarkable for being the same as the family recognized by one of the basic kinds of cellular automata. These are the *one-way real-time cellular automata* [5], also known as *trellis automata* [4]. Such an automaton is defined as a quintuple $M = (\Sigma, Q, I, \Delta, F)$, and processes an input string of length $n \geqslant 1$ using a uniform triangular array of $\frac{n(n+1)}{2}$ processor nodes, connected as in the figure. Each node computes a value from a fixed finite set Q. The nodes in the bottom row obtain their values directly from the input symbols, using a function $I : \Sigma \to Q$. The rest of the nodes compute the function $\Delta : Q \times Q \to Q$ of the values in their predecessors. The string is accepted if and only if the value computed by the topmost node belongs to the set of accepting states $F \subseteq Q$.

Formally, I is extended to a letter-to-letter homomorphism $I : \Sigma^* \to Q^*$, while the function Δ is inductively extended to the domain Q^+, with $\Delta(q\alpha q') = \Delta(\Delta(q\alpha), \Delta(\alpha q'))$. Then define $L(M) = \{w \mid \Delta(I(w)) \in F\}$.

Proposition 1 (Okhotin [12]). *A language $L \subseteq \Sigma^+$ is generated by a linear conjunctive grammar if and only if L is recognized by a trellis automaton.*

Proposition 2 (Čulík et al. [4]). *For every linear conjunctive language $L \subseteq \Sigma^*$ and for every letter $a \in \Sigma$, the languages $a^{-1} \cdot L = \{w \mid aw \in L\}$ and $L \cdot a^{-1} = \{w \mid wa \in L\}$ are linear conjunctive as well.*

3 Visibly Pushdown Languages Are Linear Conjunctive

This section demonstrates the containment of a noteworthy subclass of the deterministic context-free languages in the linear conjunctive languages. The subclass in question is the most general model in the study of structured context-free languages, in which the string itself contains a description of the structure of its own parse tree. The basic such formalism are the *parenthesis grammars* of McNaughton [10], which were extended by Berstel and Boasson [2] to the more general *balanced grammars*. The latest model are the *visibly pushdown automata*, defined by Alur and Madhusudan [1].

A visibly pushdown automaton (VPDA) is a special case of a deterministic pushdown automaton, in which the input alphabet Σ is split into three disjoint subsets Σ_c, Σ_r and Σ_i, and the type of the input symbol determines the type of the operation with the stack. For an input symbol in Σ_c, the automaton always pushes one symbol onto the stack. If the input symbol is in Σ_r, the automaton pops one symbol. Finally, for a symbol in Σ_i, the automaton may not use the stack (that is, neither modify it, nor even examine its contents).

Let Q denote the set of states of the automaton, with a subset of accepting states $F \subseteq Q$, let Γ be its pushdown alphabet, and let $\perp \in \Gamma$ be the initial pushdown symbol. For each input symbol $a \in \Sigma_c$, the behaviour of the automaton is described by partial functions $\delta_a : Q \to Q$ and $\gamma_a : Q \to (\Gamma \setminus \{\perp\})$, which provide the next state and the symbol to be pushed onto the stack, respectively. For every $b \in \Sigma_r$, there is a partial function $\delta_b : Q \times \Gamma \to Q$ specifying the next state, assuming that the given stack symbol is popped from the stack. For $c \in \Sigma_i$, the state change is described by a partial function $\delta_c : Q \to Q$. There is an additional condition that whenever the stack contains only one symbol (which will be $\perp$), any attempts to pop this symbol will result in checking that it is there, but not actually removing it. The acceptance is by reaching a state in F after reading the last symbol of the input.

Before approaching the construction of a trellis automaton simulating any given VPDA, its important elements will be illustrated on a simpler example. This is a single trellis automaton, which recognizes the nested structure of strings over a three-letter alphabet $\Sigma = \{a, b, c\}$, partitioned as $\Sigma_c = \{a\}$, $\Sigma_r = \{b\}$ and $\Sigma_i = \{c\}$. This nested structure is presented as the following variant of the Dyck language:

Example 1. Let $L_0 \subseteq \{a, b, c\}$ be the language defined by the context-free grammar $S \to aAb \mid c$, $A \to SA \mid \varepsilon$. Then the trellis automaton with the set of states $\{\nearrow, \nwarrow, -, /\!\!/\}$, illustrated in Figure 1, carries out computations sufficient to parse L_0. In particular, adding an extra accepting state $*$ with the extra transitions shown outside of the table yields a trellis automaton recognizing $(L_0 \cup \{b\})^+\$$.

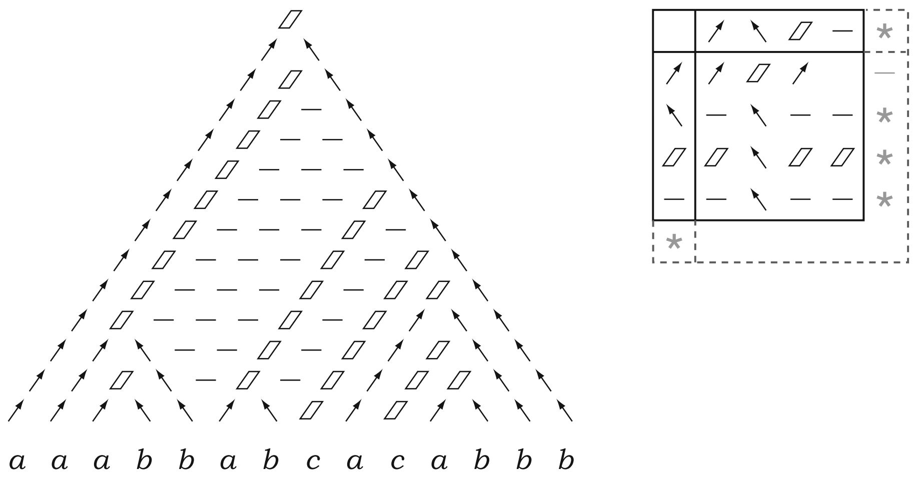


Fig. 1. A trellis automaton processing L_0

The state $\nearrow$ represents a letter a looking for a matching b to the right, and similarly, $\nwarrow$ represents b looking for a matching a to the left. Once these arrows meet, they produce a state ▱, and a single letter c produces the same state ▱. The fourth state "–" is used to fill the areas, where no computations are done.

The initial function of the trellis automaton is defined by $I(a) = \nearrow$, $I(b) = \nwarrow$, $I(c) =$ ▱. Now the goal is to compute ▱ on each element of L_0, as well as to compute $\nearrow$ on each proper prefix of a string in L_0, and $\nwarrow$ on each proper suffix. The computation is defined and proved inductively on the structure of the strings in L_0.

The base case is a one-symbol string $c \in L_0$, on which the trellis automaton computes the intended result. For a string of the form $aw_k \ldots w_1 b$ with $w_i \in L_0$, the general plan is that the automaton computes the state ▱ on each w_i (which holds by the induction hypothesis), and then these states are propagated to the right, until they meet the arrows $\nwarrow$ spawned from the last b. The diagonals, in which no states ▱ are propagated, are filled with the states "–".

The computations carried out between every subsequent substrings w_{i+1} and w_i depend on the types of both strings. If $w_{i+1} = c$ and $w_i = c$, then the propagation is done by

$$\Delta(▱, ▱) = ▱. \tag{2a}$$

For $w_{i+1} = awb$ and $w_i = aw'b$, the middle zone between them is filled by the transitions

$$\Delta(\nwarrow, \nearrow) = -, \tag{2b}$$

$$\Delta(\nwarrow, -) = -, \tag{2c}$$

$$\Delta(-, \nearrow) = -, \tag{2d}$$

$$\Delta(-, -) = -, \tag{2e}$$

and then the state $/\!\!/$ computed for w_{i+1} is propagated by the transition

$$\Delta(/\!\!/, -) = /\!\!/. \tag{2f}$$

If $w_{i+1} = c$ and $w_i = awb$, the required transitions are

$$\Delta(/\!\!/, \nearrow) = /\!\!/ \tag{2g}$$

and (2a). The last case of $w_{i+1} = awb$ and $w_i = c$ uses the transitions

$$\Delta(\nwarrow, /\!\!/) = -, \tag{2h}$$
$$\Delta(-, /\!\!/) = -, \tag{2i}$$

This concludes the propagation of signals from all w_i.

On the right border, the diagonal spawned from b consumes the signals coming from w_i using the transitions

$$\Delta(/\!\!/, \nwarrow) = \nwarrow, \tag{2j}$$
$$\Delta(-, \nwarrow) = \nwarrow, \tag{2k}$$

as well as one extra transition

$$\Delta(\nwarrow, \nwarrow) = \nwarrow \tag{2l}$$

used if $w_1 = awb$. The left border is maintained by the transitions

$$\Delta(\nearrow, /\!\!/) = \nearrow, \tag{2m}$$
$$\Delta(\nearrow, \nearrow) = \nearrow, \tag{2n}$$

of which the latter is used only for $w_k = awb$.

Finally, once the left diagonal meets the right diagonal, the following transition is employed:

$$\Delta(\nearrow, \nwarrow) = /\!\!/. \tag{2o}$$

The above trellis automaton shall now be generalized to establish the following result:

Theorem 1. *Let* $(\Sigma_c, \Sigma_r, \Sigma_i, Q, \{(\delta_a, \gamma_a)\}_{a \in \Sigma_c}, \{\delta_b\}_{b \in \Sigma_r}, \{\delta_c\}_{c \in \Sigma_i}, F)$ *be a VPDA recognizing a language* $L \subseteq (\Sigma_c \cup \Sigma_r \cup \Sigma_i)^*$. *Then there exists a trellis automaton that recognizes the same language.*

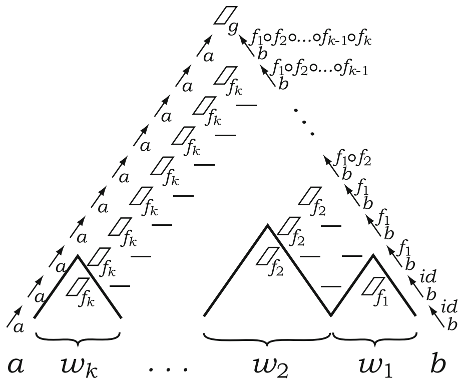
Proof. The main construction produces a trellis automaton recognizing the language ¢L\$, where ¢ and \$ are two new symbols. This automaton operates by the same principle as the one in Example 1, and each of its states represents one of the states $\nearrow, \nwarrow, /\!\!/, -$, with some attached data related to the input symbols and internal states of a VPDA.

Though the state of a VPDA at each point of its computation is uniquely determined, it depends on all symbols of the input up to the current position. However, a trellis automaton will not have all this data available (except on the left diagonal of its computation, that is, almost never). Because of this, the deterministic operation of a VPDA is simulated nondeterministically: for each substring of the input with balanced parentheses, the constructed trellis automaton shall trace the computation of a VPDA beginning with all possible states.

In accordance with this plan, a state $/\!\!/$ from the automaton in Example 1 now has to remember a function $f : Q \to Q$, which maps a state, on which the VPDA begins its computation on the current substring, to the resulting state after reading this substring. Each arrow $\nearrow$ remembers the letter $a \in \Sigma_c$, from each it has been spawned. Arrows $\nwarrow$ also remember their original symbol $b \in \Sigma_r$, and besides it they calculate a function $f : Q \to Q$ representing the behaviour of the VPDA on the respective substrings. In overall, the constructed automaton has the set of states

$$\{\nearrow_a \mid a \in \Sigma_c\} \cup \{\nwarrow_b^f \mid b \in \Sigma_r,\ f : Q \to Q\} \cup \{/\!\!/_f \mid f : Q \to Q\} \cup \{-\},$$

and its computations extend those in Example 1 by maintaining the following information flow.


$/\!\!/_g$
$\nwarrow_b^{f_1 \circ f_2 \circ \ldots \circ f_{k-1} \circ f_k}$
$\nwarrow_b^{f_1 \circ f_2 \circ \ldots \circ f_{k-1}}$
$\nwarrow_b^{f_1 \circ f_2}$
$\nwarrow_b^{f_1}$
$\nwarrow_b^{id}$
$\nearrow_a$
$/\!\!/_{f_k}$
$/\!\!/_{f_2}$
$/\!\!/_{f_1}$
a
w_k
$\ldots$
w_2
w_1
b

The initial function sets the values $I(a) = \nearrow_a$ for $a \in \Sigma_c$; $I(b) = \nwarrow_b^{id}$ for $b \in \Sigma_r$, where $id : Q \to Q$ is the identity function; $I(c) = /\!\!/_{\delta_c}$ for $c \in \Sigma_i$. The transitions belong to the same fifteen groups as in Example 1, but now they have data to communicate and transform. The analogues of transitions (2a)–(2i), which simply propagate the data, are omitted for brevity. The first nontrivial transition is (2j), which now computes the composition of behaviours of the VPDA on neighbouring substrings, thus producing the behaviour on their concatenation:

$$\Delta(/\!\!/_g, \nwarrow_b^f) = \nwarrow_b^{f \circ g}.$$

The second operation is calculating the behaviour of the VPDA on the entire substring $aw_k \ldots w_1 b$: the transition (2o) is expanded into

$$\Delta(\nearrow_a, \nwarrow_b^f) = /\!\!/_g,$$

where the new function $g(q) = \delta_b\big(f(\delta_a(q)), \gamma_a(q)\big)$ represents the behaviour of the VPDA on this substring as follows. If the VPDA begins reading the substring in the state q, it first reads a, pushes $\gamma_a(q)$ and enters the state $\delta_a(q)$, then processes $w_k \ldots w_1$, finishing in the state $f(\delta_a(q))$, and finally pops the symbol $\gamma_a(q)$ from the stack and uses it in its transition by b.

The last step of the construction is to add new symbols ¢ and \$ and recognize the language ¢L\$. For all $f : Q \to Q$, let $\Rsh_f$ and $\Lsh_f$ be new states, and define $I(¢) = \Rsh_{id}$ and $I(\$) = \Lsh_{id}$. The purpose of the states $\Rsh_f$ is to collect returns without calls, which are allowed as per the definition of VPDAs. This is done by the transitions

$$\begin{aligned} \Delta(\Rsh_f, \nearrow_a) &= \Rsh_f, \\ \Delta(\Rsh_f, /\!\!/_g) &= \Rsh_f, \\ \Delta(\Rsh_f, \nwarrow^g_b) &= \Rsh_h, \quad \text{with } h(q) = \delta_b(f(q), \perp). \end{aligned}$$

The states $\Lsh_f$ similarly compute the behaviour of the VPDA on the suffix beginning after the last unmatched return:

$$\begin{aligned} \Delta(\nwarrow^g_b, \Lsh_f) &= \Lsh_f, \\ \Delta(-, \Lsh_f) &= \Lsh_f, \\ \Delta(/\!\!/_g, \Lsh_f) &= \Lsh_{f \circ g} . \end{aligned}$$

These two parts of the string are connected by the following transition leading to a new accepting state $*$:

$$\Delta(\Rsh_f, \Lsh_g) = *, \quad \text{if } g(f(q_0)) \in F.$$

Having constructed a trellis automaton recognizing ¢L\$, the desired trellis automaton for L is obtained from it by applying Proposition 2 twice. □

4 Languages That Are Not Linear Conjunctive

Having found a subclass of deterministic context-free languages that can be simulated by trellis automata, one naturally ponders on whether any larger subclasses, such as LL(k) context-free languages, deterministic context-free languages, or even unambiguous context-free languages, can be simulated as well. The answer, which turns out to be negative, shall now be obtained by using the method of Terrier [17] to prove that there are no trellis automata for some languages that are simpler than the previously known examples.

The method of Terrier is based upon a special complexity function of a language, which reflects the necessary amount of calculations in the last few steps of the computation of a trellis automaton. This complexity function is defined for an arbitrary language as follows:

Definition 2. *Let $L \subseteq \Sigma^*$ be a language, let $k \geqslant 1$ and let $w = a_1 \ldots a_n$ be a string with $n \geqslant k$. Define a set*

$$S_{L,k,w} = \{(i,j) \mid i, j \geqslant 0,\ i + j < k,\ a_{i+1} \ldots a_{n-j} \in L\},$$

which represents the membership in L of all substrings of w longer than $|w| - k$ symbols. Next, define the set of all sets $S_{L,k,w}$ for all strings w:

$$\widehat{S}_{L,k} = \{S_{L,k,w} \mid w \in \Sigma^*,\ |w| \geqslant k\}.$$

Let $f_L(k) = |\widehat{S}_{L,k}|$.

Each set $S_{L,k,w}$ has between 0 and $\frac{k(k+1)}{2}$ elements, and accordingly, the cardinality of the set $\widehat{S}_{L,k}$ is between 1 (if $L \cap \Sigma^+$ is $\varnothing$ or Σ^+) and $2^{\frac{k(k+1)}{2}}$.

This measure exposes the following limitation of linear conjunctive languages: their growth rate cannot get as high as $2^{\Theta(k^2)}$, and is limited by $2^{O(k)}$:

Lemma 1 (Terrier [17]). *If $L \in \Sigma^*$ is linear conjunctive, then its complexity measure $f_L(k)$ is bounded by an exponential function, that is,*

$$f_L(k) \leqslant p^k, \quad \textit{for some } p \geqslant 1.$$

The reason for this limitation is that for every string $w = a_1 \dots a_n$ with $n \geqslant k$, a trellis automaton has to determine the membership in L of all substrings $a_{i+1} \dots a_{n-j}$ with $i, j \geqslant 0$ and $i+j < k$, based only on the states it has computed on the substrings with $i + j = k - 1$. The number p in the lemma reflects the number of states in a trellis automaton recognizing L, and thus the automaton can distinguish between p^k possible situations.

Lemma 1 was accompanied with an example of a context-free language that maximizes this complexity measure, and hence is recognized by no trellis automaton:

Example 2 (Terrier [17]). Consider the linear context-free language

$$L_0 = \{a^n w b^n \mid n \geqslant 1;\ w \in b\{a,b\}^*a \text{ or } w = \varepsilon\}.$$

Then the language

$$L = L_0 \cdot L_0 = \{a^{i_1} b^{j_1} \dots a^{i_m} b^{j_m} \mid m \geqslant 2;\ i_t, j_t \geqslant 1,\ \exists \ell :\ i_1 = j_\ell \text{ and } i_{\ell+1} = j_m\}$$

has $f_L(k) = 2^{\frac{k(k+1)}{2}}$ and therefore is not linear conjunctive.

However, the language in Example 2 is inherently ambiguous [14, Prop. 8], and in order to separate linear conjunctive languages from unambiguous context-free languages and their subclasses, new examples are needed.

Lemma 2. *The language*

$$L = \{c^m a^{\ell_0} b \dots a^{\ell_{m-1}} b a^{\ell_m} b \dots a^{\ell_z} b d^n \mid m, n,\ \ell_i \geqslant 0,\ z \geqslant 1,\ \ell_m = n\}$$

is generated by an LL(1) context-free grammar, as well as recognized by a deterministic one-counter automaton. However, $f_L(k) \geqslant (k+1)! = 2^{\Theta(k \log k)}$, and therefore L is not linear conjunctive.

Proof. From the outlook of pushdown automata, the membership of a string $c^m w d^n$ with $w \in \{a,b\}^*$ in L can be determined as follows. First, the prefix c^m is read, and its length is stored in the stack. Then the automaton uses these stack contents to count the first m instances of b in w, skipping any as encountered, Right after the m-th b, the automaton reads the next block of as, storing its length in the stack, and then skips all remaining as and bs. Finally, the stored length of the block of as is compared against the number of ds. This is a deterministic three-turn one-counter automaton.

In order to construct an LL(1) context-free grammar for L, consider that this language is representable as a concatenation of the following two languages:

$$L_1 = \{c^m a^{\ell_1} b \dots a^{\ell_m} b \mid m \geqslant 0,\ \ell_i \geqslant 0\},$$
$$L_2 = \{a^n b w d^n \mid n \geqslant 0,\ w \in \{a,b\}^*\}.$$

This leads to the following context-free grammar, which turns out to be LL(1):

$$\begin{aligned} S &\to CD \\ C &\to cCAb \mid \varepsilon \\ A &\to aA \mid \varepsilon \\ D &\to aDd \mid bB \\ B &\to aB \mid bB \mid \varepsilon \end{aligned}$$

The last claim about this language is that $f_L(k) \geqslant (k+1)!$ for each $k \geqslant 1$. For every k-tuple of integers $(\ell_0, \dots, \ell_{k-1})$ with $k-1-i \leqslant \ell_i \leqslant k$ for each i, define the corresponding string

$$w_{\ell_0,\dots,\ell_{k-1}} = c^{k-1} a^{\ell_0} b \dots a^{\ell_{k-1}} b d^{k-1}.$$

Then the set $S_{L,k,w_{\ell_0,\dots,\ell_{k-1}}}$ for this string contains all pairs (i,j) with $i,j \geqslant 0$ and $i+j<k$, which satisfy $c^{k-1-i} a^{\ell_0} b \dots a^{\ell_{k-1}} b d^{k-1-j} \in L$, that is, $\ell_{k-1-i} = k-1-j$. Thus different possible values of ℓ_{k-1-i} are indicated by the membership of the pairs $(i,0)$, $(i,1)$, …, $(i,k-1-i)$ in $S_{L,k,w_{\ell_0,\dots,\ell_{k-1}}}$: if ℓ_{k-1-i} is between $k-1-i$ and $k-1$, then exactly one of these pairs is in the set, and if $\ell_{k-1-i} = k$, then none of them are present.

Accordingly, the sets $S_{L,k,w_{\ell_0,\dots,\ell_{k-1}}}$ corresponding to different tuples $(\ell_0, \dots, \ell_{k-1})$ are pairwise distinct, and since there are $(k+1)!$ such tuples, $f_L(k) = |\{S_{L,k,w} \mid w \in \Sigma^*,\ |w| \geqslant k\}| \geqslant (k+1)!$, as claimed. □

Theorem 2. *The family of linear conjunctive languages is incomparable with unambiguous context-free languages, deterministic context-free languages and LL(k) context-free languages.*

5 LL(k) Context-Free vs. Visibly Pushdown Languages

Lemma 2 suggests a further question of comparing LL(k) context-free languages to visibly pushdown languages. An LL(1) context-free language without a VPDA is well-known: that is the language $\{a^n b^n \mid n \geqslant 0\} \cup \{b^n a^n \mid n \geqslant 0\}$ [1]. The next example shows that there is no converse inclusion either.

Lemma 3. *The language $\{a^n b^n \mid n \geqslant 0\} \cup \{a^n c^n \mid n \geqslant 0\}$ is recognized by a VPDA with $\Sigma_c = \{a\}$, $\Sigma_r = \{b, c\}$, $\Sigma_i = \varnothing$. On the other hand, there is no LL(k) context-free grammar for this language.*

Proof. The construction of a VPDA for this language requires no explanation.

To see that this language is not LL context-free, suppose that it is generated by some LL(k) context-free grammar $G = (\Sigma, N, P, S)$. Assume, without loss of generality, that $\varepsilon \notin L_G(A)$ for each $A \in N$: every LL context-free grammar can be transformed to this form by increasing the lookahead length by one. For each nonterminal A, let ℓ_A be the length of the shortest string generated by A, and let $\ell = \max_{A \in N} \ell_A$.

For each $n \geqslant k-1$, consider the computation of the LL(k) parser on the strings $a^n b^n$ and $a^n c^n$, and let $\alpha_n \in (\Sigma \cup N)^*$ be the stack contents at the point when the prefix $a^{n-(k-1)}$ has been read and $a^{k-1} b^n$ or $a^{k-1} c^n$ remains unread. Up to this point, the computation does not depend upon whether there are bs or cs ahead, because a parser with k symbols of lookahead could not yet see them. So it must be ready to recognize these two possible continuations, that is, $L_G(\alpha_n)$ must be exactly $\{a^{k-1} b^n, a^{k-1} c^n\}$. Therefore, the strings α_n for different n are pairwise distinct, and their length is accordingly unbounded. Fix n as the smallest number with $|\alpha_n| \geqslant k$ and $n > \ell$.

Since both $a^{k-1} b^n$ and $a^{k-1} c^n$ are in $L_G(\alpha_n)$, the last symbol of α_n must be a nonterminal. Denote $\alpha_n = \beta A$ with $A \in N$, and consider the factorizations $a^{k-1} b^n = u_1 u_2$ and $a^{k-1} c^n = v_1 v_2$ with $u_1, v_1 \in L_G(\beta)$ and $u_2, v_2 \in L_G(A)$. Since $|\beta| \geqslant k-1$, the string u_1 contains at least the prefix a^{k-1}, and similarly $v_1 \in a^{k-1} c^*$. Then the factorization takes the form $a^{k-1} b^{n-i}, a^{k-1} c^{n-j} \in L_G(\beta)$ and $b^i, c^j \in L_G(A)$ for some $i, j \in \{1, \ldots, n\}$.

If $i > \ell$, then there is a shorter string $x \in L_G(A)$ with $|x| < i$, and hence $a^{k-1} b^{n-i} x \in L_G(\alpha)$, which further implies $a^n b^{n-i} x \in L(G)$, where $|b^{n-i} x| < n$, and this is a contradiction. Therefore, $i \leqslant \ell < n$, and for the same reason $j \leqslant \ell < n$. Now the string $a^{k-1} b^{n-i} c^j$ is in $L_G(\alpha)$, and $a^n b^{n-i} c^j$ is in $L(G)$, where $n - i > 0$ and $j > 0$, which is a contradiction as well. □

Theorem 3. *The family of LL context-free languages is incomparable with visibly pushdown languages.*

6 Conclusion

The relations between the language families studied in this paper is illustrated in Figure 2. All inclusions are proper, except the ones marked with a question mark, and any two families not connected by a directed path are now known to be incomparable. This increases our knowledge of the basic families of formal languages, and suggests further research on extending structured models in formal language theory towards Boolean grammars.

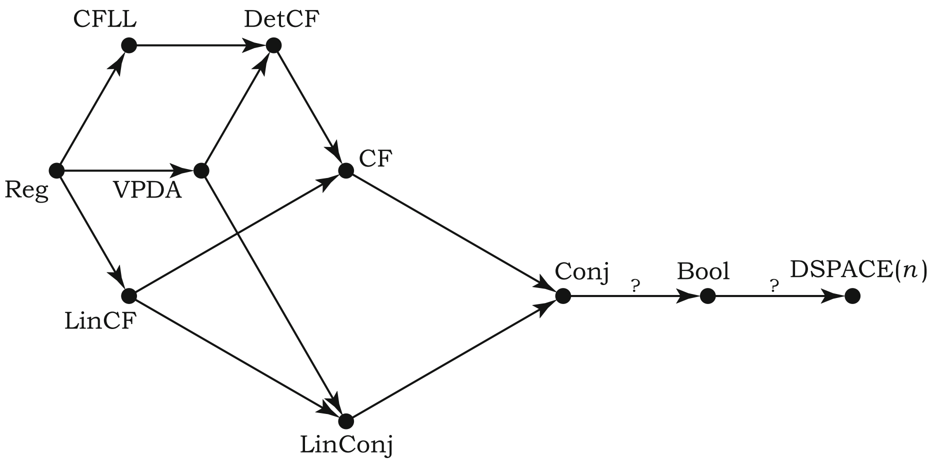


Fig. 2. The hierarchy of language families

References

1. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: ACM Symposium on Theory of Computing, STOC 2004, Chicago, USA, June 13–16, pp. 202–211 (2004)
2. 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)
3. Crespi-Reghizzi, S., Mandrioli, D.: Algebraic properties of structured context-free languages: old approaches and novel developments. In: WORDS (2009)
4. Čulík II, K., Gruska, J., Salomaa, A.: Systolic trellis automata. International Journal of Computer Mathematics 15, 195–212 (1984); 16, 3–22 (1984)
5. Dyer, C.: One-way bounded cellular automata. Information and Control 44, 261–281 (1980)
6. Ibarra, O.H., Kim, S.M.: Characterizations and computational complexity of systolic trellis automata. Theoretical Computer Science 29, 123–153 (1984)
7. Jeż, A.: Conjunctive grammars can generate non-regular unary languages. International Journal of Foundations of Computer Science 19(3), 597–615 (2008)
8. Jeż, A., Okhotin, A.: Conjunctive grammars over a unary alphabet: undecidability and unbounded growth. Theory of Computing Systems 46(1), 27–58 (2010)
9. Kountouriotis, V., Nomikos, C., Rondogiannis, P.: A game-theoretic characterization of Boolean grammars. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 334–347. Springer, Heidelberg (2009)
10. McNaughton, R.: Parenthesis grammars. Journal of the ACM 14(3), 490–500 (1967)
11. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics 6(4), 519–535 (2001)
12. Okhotin, A.: On the equivalence of linear conjunctive grammars to trellis automata. Informatique Théorique et Applications 38(1), 69–88 (2004)
13. Okhotin, A.: Boolean grammars. Information and Computation 194(1), 19–48 (2004)

14. Okhotin, A.: Unambiguous Boolean grammars. Information and Computation 206, 1234–1247 (2008)
15. Okhotin, A.: Fast parsing for Boolean grammars: a generalization of Valiant's algorithm. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 340–351. Springer, Heidelberg (2010)
16. Okhotin, A., Reitwießner, C.: Conjunctive grammars with restricted disjunction. Theoretical Computer Science 411(26-28), 2559–2571 (2010)
17. Terrier, V.: On real-time one-way cellular array. Theoretical Computer Science 141, 331–335 (1995)

A Local Search Algorithm for Branchwidth

Arnold Overwijk[1], Eelko Penninkx[2], and Hans L. Bodlaender[2]

[1] School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA
[2] Department of Computing Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
aoverwij@cs.cmu.edu, penninkx@cs.uu.nl, hansb@cs.uu.nl

Abstract. As many problems can be solved in polynomial time on graphs of bounded branchwidth, there is a need for efficient and practical algorithms that find branch decompositions of small width of given graphs. In this paper, we give the first local search algorithm for this problem. Using non-trivial combinatorial properties of the neighbourhood space, significant reductions in the search space can be obtained. The fitness function can be easily tuned to express the expected cost in the dynamic programming step for specific problems. The algorithm is tested on a number of planar and non-planar graphs. For all tested planar graphs the algorithm found an optimal branch decomposition. For most non-planar graphs, we obtained branch decompositions of width equal to or smaller than the treewidth. Our experiments indicate that local search is a very suitable technique for finding branch decompositions of small width.

1 Introduction

It is well known that many problems that are intractable (e.g., NP-hard) on arbitrary graphs become polynomial or linear time solvable when restricted to graphs of bounded branchwidth or treewidth. This technique has been used in several theoretical as well as practical settings. We refer the reader to the surveys [6,8,23,24] for more background. In many cases, a combinatorial problem is solved in the following way: first a branch (or tree) decomposition of small width is found, and then a dynamic programming algorithm is applied on it. As the running time of the second step is usually exponential (or worse) in the width of the decomposition obtained in the first step, there is the need of 'good' algorithms that find branch (or tree) decompositions of small width.

Branchwidth and treewidth were first introduced by Robertson and Seymour in their work on the graph minors theorem [26,27]. The branchwidth $\beta(G)$ and treewidth $\tau(G)$ of a graph G are related by $\beta(G) \leq \tau(G) \leq \lfloor \frac{3}{2}\beta(G) \rfloor$ (see [27,5].) In many cases, using tree decompositions instead of branch decompositions would give algorithms with the same asymptotic running time; however, there can be significant differences in the constant factor, and thus in the actual running time of programs using these decompositions. (See also [15].)

It should be noted that some of the theoretical algorithmic results for finding tree or branch decompositions, e.g., the linear time algorithms for fixed values of k are not useful in practice [4,10,28], as the constant factors, even for small values of k, are much too large. Thus, there is the need for practical algorithms that determine the branchwidth or treewidth of a graph, and find corresponding small width decompositions. While much work has been done on finding tree decompositions in practical settings (see e.g., the overviews in [7,9]), relatively few such work has been carried out on finding branch decompositions. This is probably due to the relative strong focus in the earlier theoretical results on treewidth and the fact that tree decompositions have several equivalent formulations that are easier to work with, e.g., with a representation of a linear ordering of the vertices (see [5,11]), where such formulations are not known for branchwidth.

Branch decomposition based algorithms with practical importance for problems in planar graphs have received a great deal of attention in the past decade [14,17]. Seymour and Thomas gave an $O(n^2)$ time algorithm [29] for deciding if the branchwidth of a planar graph is at most a given value β. Versions that also compute minimum width branch decompositions use more time ($O(n^3)$ by Gu and Tamaki [19]). Experimental studies show that these algorithms are efficient in practice [21,22,3]. Hicks also gave a divide and conquer heuristic to make the algorithm even more efficient in practice [22].

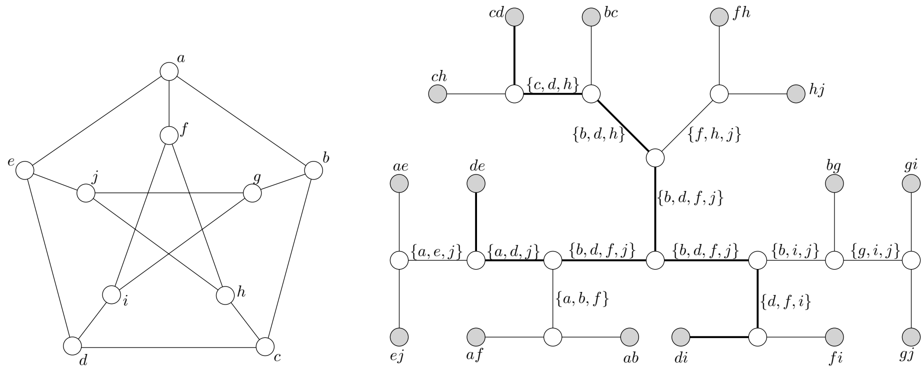
Branchwidth is $\mathcal{NP}$-complete on general graphs [29]. Therefore we have to rely on heuristics (or slow exact methods like [16]) to find branch decompositions for general graphs. Cook and Seymour [12,13] gave a heuristic, based on spectral graph theory and the work of Alon [1], to produce branch decompositions. Hicks [20] also presented a heuristic that was comparable with the heuristic of Cook and Seymour. The algorithm finds separations by minimal vertex separators between diameter pairs.

In this paper we present a local search based algorithm for constructing branch decompositions on general graphs. Although a significant amount of research has been performed on local search for the related problem of finding tree decompositions [8,11,25], our algorithm is the first local search based heuristic for constructing branch decompositions. Our heuristic is an anytime algorithm that is tested on a number of planar graphs for which the branchwidth is already known. Moreover, we have tested our heuristic on a number of non planar graphs, some taken in order to compare our results to heuristics by Hicks [20], and some taken from the graphs that are nowadays used as benchmark for testing algorithms for treewidth.

This paper is organised as follows. In section 2, we give some preliminary definitions. We present our upperbound heuristic and prove its correctness in Section 3. Section 4 describes our implementations. Computational results are presented in Section 5. Conclusions and suggestions for future work can be found in Section 6.

2 Preliminaries

Throughout this paper we use n for the number of vertices and m for the number of edges in the input graph G. All graphs in this paper are undirected. We also assume that all graphs are biconnected as, the branchwidth of a graph equals the branchwidth of its biconnected components[1]. A *ternary tree* is a tree where every non-leaf node has degree 3.



(a) Petersen graph (b) Optimal branch decomposition

Fig. 1. An optimal branch decomposition for the Petersen graph

Consider a graph G with vertex set $V(G)$ and edge set $E(G)$. A *branch decomposition* of G is a ternary tree T such that there is a bijective mapping between the leaves of T and $E(G)$. Removing an edge e from T separates T into two subtrees T' and T''. Let E' and E'' be the edges of G contained in the leaves of the two subtrees. The *middle set* of e, denoted mid(e), is the set of vertices of G incident to some edge in both E' and E''. The width of an edge e is the size of mid(e). The *width* of T is the maximum width over all edges in T. The *branchwidth* of G, denoted by $\beta(G)$, is the minimum width over all branch decompositions of G. An *optimal branch decomposition* of G has branchwidth $\beta(G)$.

The following fact is a consequence of the definitions:

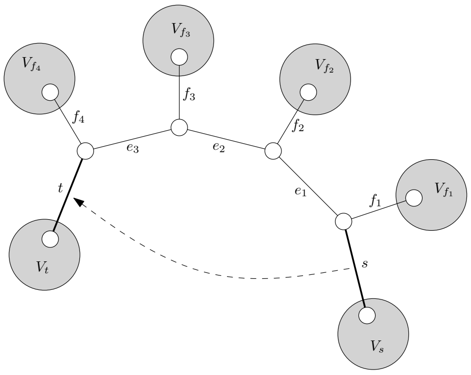
Lemma 1. *Let G be graph, and $\mathcal{B}$ be some branch decomposition of G. Then for all $v \in V(G)$ the edges $e \in E(\mathcal{B}) : v \in mid(e)$ induce a connected subgraph of $\mathcal{B}$.*

Figure 1 illustrates an optimal branch decomposition for the Petersen graph. The middle sets containing vertex d are highlighted.

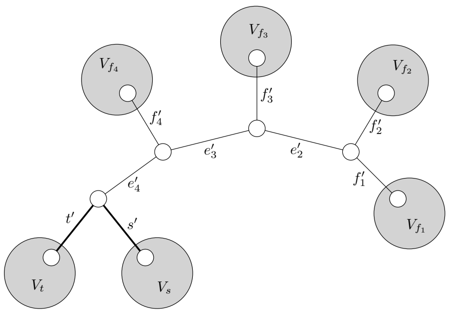
[1] This folklore fact can be shown by repeating the following step: suppose $W \subseteq V$ forms a biconnected component of G, with cutvertex v. If we have branch decompositions of $G[W]$ and $G[(V-W) \cup \{v\}]$ of width k' and k'', respectively, then we can build a branch decomposition of G of width $\max\{k', k''\}$ by subdividing in each branch decomposition an edge that contains v in the middle set, and adding an edge to the branch decomposition that connects the two new nodes.

3 Local Search

Consider a biconnected graph G and a branch decomposition $\mathcal{B}$ of G. Our algorithm repeatedly tries to move subtrees of $\mathcal{B}$ to another location to improve its quality. For distinct edges s and t in $E(\mathcal{B})$ we define the *move* operator, denoted move(s,t), as follows. Let e_i denote the edges on the simple path $P = (s, e_1, \ldots, e_k, t)$, where $1 \leq i \leq k$. Note that $k \geq 1$ if s and t are not connected. Let $f_1, \ldots, f_{k+1}$ denote the edges incident to the internal vertices on P. Let $V_{f_i}, V_s, V_t \subseteq V(G)$ be the vertices of G incident to some edge in the leaves of the corresponding subtree. A move is performed by contracting e_1, subdividing t by inserting a new vertex v, connecting s to v, and updating the middle sets. See Figure 2 for an illustration. Observe that if s and t are adjacent the move operator does not change the tree.



(a) Before



(b) After

Fig. 2. Example for move(s,t)

We will now show that for all branch decompositions $\mathcal{B}, \mathcal{B}'$ there is a series of moves that transforms $\mathcal{B}$ to $\mathcal{B}'$. We define a *linear branch decomposition* as a branch decomposition whose non-leaf nodes induce a path. We need the following lemma:

Lemma 2. *Let $\mathcal{P}$ denote a linear branch decomposition. Then every branch decomposition $\mathcal{B}$ can be transformed to $\mathcal{P}$ by a series of move operations.*

Proof. We prove this by construction. Let $d_i \in E(\mathcal{P}), i = 1, \ldots, m$ denote the edges of $\mathcal{P}$ connected to the leaves of $\mathcal{P}$, and let b_i be the edges incident to the leaves of $\mathcal{B}$ such that b_i and d_i both connect to the leaf corresponding to the same edge in G. Then a series of moves move(b_2, b_1), move$(b_3, b_2), \ldots,$ move(b_m, b_{m-1}) will transform $\mathcal{B}$ to $\mathcal{P}$. $\square$

Theorem 1. *The search space of the move operator is complete: for all branch decompositions $\mathcal{B}, \mathcal{B}'$ there is a series of moves that transforms $\mathcal{B}$ to $\mathcal{B}'$.*

Proof. We first observe that the move operator is invertible, as we can undo move(s, t) by applying move(s', f_1'). Together with Lemma 2 this implies that we can transform $\mathcal{B}$ to any linear branch decomposition $\mathcal{P}$, and $\mathcal{P}$ to $\mathcal{B}'$. □

After performing some move(s, t) we need to update the middle sets of the branch decomposition. We use the same notation as before, see Figure 2. For ease of notation we notate $\text{mid}(e_1') = \text{mid}(e_{k+1}) = \emptyset$ for the non-existing edges e_1' and e_{k+1}. From the definition of the middle sets it immediately follows that $\text{mid}(s') = \text{mid}(s), \text{mid}(t') = \text{mid}(t)$ and $\text{mid}(f_i') = \text{mid}(f_i)$. Hence the only middle sets that need to be updated are $\text{mid}(e_i')$. Let $A_i = \text{mid}(e_i') \setminus \text{mid}(e_i)$ denote the vertices added to $\text{mid}(e_i)$, and $B_i = \text{mid}(e_i) \setminus \text{mid}(e_i')$ denote the removed vertices. Note that $A_i, B_i \subseteq \text{mid}(s)$. We have the following:

Lemma 3. *For all A_i, A_j with $1 \leq i < j \leq k+1$, it holds that $A_i \subseteq A_j$.*

Proof. Suppose for a contradiction that $A_i \not\subseteq A_j$ for some $i < j$. Let v be a vertex in A_i that is not in A_j. Then $v \in \text{mid}(e_i')$ and $v \notin \text{mid}(e_j')$. As $v \in \text{mid}(s) = \text{mid}(s')$ this means that the edges with middle sets containing v are not connected, contradicting Lemma 1. □

Lemma 4. *For all B_i, B_j with $1 \leq i < j \leq k+1$, it holds that $B_j \subseteq B_i$.*

Proof. We will show that $v \in B_j$ implies that $v \in B_i$, from which the lemma directly follows. If $v \in B_j$ obviously $v \in \text{mid}(e_j)$, and also $v \in \text{mid}(s) = \text{mid}(s')$. Lemma 1 directly gives $v \in \text{mid}(e_i)$ for all $i < j$. Because $v \notin \text{mid}(e_j')$ then the same theorem also implies that $v \notin \text{mid}(e_i')$ for all $i < j$, and hence $v \in B_i$, proving the lemma. □

These lemmata show that the set of vertices we remove from the middle sets on the path from s to t only shrinks, while the set of vertices added only increases. We directly obtain the main theorem of this section:

Theorem 2. *For all edges e_i, e_j with $1 \leq i < j \leq k+1$, it holds that $|mid(e_i)| - |mid(e_i')| \geq |mid(e_j)| - |mid(e_j')|$.*

3.1 Search Strategy

We will now show how the results from the previous section can be applied to our local search strategy. Note that a branch decomposition has m leaves, $m-2$ internal nodes and $2m-3$ edges. A naive search strategy would try all $O(m^2)$ possible edges s and t to find an improvement. We will show how the results from the previous section can be used to improve this.

Suppose we want to move edge s to the best possible location in the tree. We perform a depth first search on the tree edges, starting at s. Along the way we construct A_t and B_t for all visited edges t. This can be done very efficiently because of Lemmata 3 and 4. As soon as $|B_t| < |A_t|$ for some visited edge t we prune the search tree at t by Theorem 2.

3.2 Fitness Function

A fitness function quantifies the optimality of a solution, which ideally correlates closely with the algorithm's goal, and yet may be computed quickly. For a branch decomposition $\mathcal{B}$ the first idea would be to use branchwidth, but this would create large parts of the search space with the same fitness. Instead, we take the width of all edges into account as follows. Let $E_i = \{e \in E(\mathcal{B}) : |\text{mid}(e) = i|\}$ contain all edges of $\mathcal{B}$ of width i. The fitness of $\mathcal{B}$ is defined as $(|E_n|, |E_{n-1}|, \ldots, |E_1|)$, using the lexicographical ordering for comparison (a lower value is better).

We can use a different fitness function if the branch decomposition will be used to solve some specific problem modelled on a graph:

$$\text{Fitness} = \sum_{i=1}^{n} |E_i| f(i)$$

Usually $f(i)$ should correspond to the amount of work necessary to process middle sets of size i in the dynamic programming step. For most problems $f(i) \sim c^i$, as this corresponds to the maximum size of the tables during dynamic programming.

4 Implementation

Let G be a graph with n vertices and m edges. Our local search is started with a *linear branch decomposition* $\mathcal{B}_0$ (remember that the leaves of $\mathcal{B}_0$ induce a path) with the edges of G randomly assigned to its leaves. Starting with this (very poor) initial solution we perform moves that lead us to neighbour solutions.

For each edge in the branch decomposition, we keep track of the nodes present on both sides. We use a Boolean array to store on which side a node in the branch decomposition is located. Furthermore we also keep track of the middle sets by using two bits for each vertex of the original graph storing for each side if the vertex is present at that side; for faster processing, we store these in an array of bytes. When a move is performed, this allows us to update the middle set of the edges in $O(k * n)$ time, where k is the length of the path. Notice that there is room for improvement by using better data structures (e.g. sparse vectors).

Iterated Hill Climbing. The main part of the algorithm consists of a straightforward *hill climbing* approach. The function *findBestTarget* finds the target edge t that optimises the fitness when moving edge s. If no such edge exists it returns *nil*. We use the following algorithm:

```
$i = 0$; $f_0 = \text{fitness}(\mathcal{B}_0)$;
**repeat**
  i++; $\mathcal{B}_i \Leftarrow \mathcal{B}_{i-1}$; $f_i = f_{i-1}$;
  **for all** $s \in E(\mathcal{B}_i)$ **do**
    $t \Leftarrow$findBestTarget$(\mathcal{B}_i, s)$;
    **if** $t \neq nil$ **then** move$(s, t)$; $f_i = \text{fitness}(\mathcal{B}_i)$;
  **end for**
**until** $f_i = f_{i-1}$;
```

This algorithm terminates in a local optimum, which might not be the global optimum. To escape from the attraction of the local optimum, we can perform a number of random moves and start hill climbing again. We have carried out a few experiments, and it appears that in most cases, just two or three such moves are sufficient to obtain the desired effect. A further experimental study seems interesting.

5 Results

All experiments are run on a Intel(R) Core(TM)2 Duo CPU P8400 @ 2.26 GHz running a 32bit version of Windows XP. The algorithms are single threaded and implemented in the C#.Net language. We noticed that C#.Net is not the most efficient language, however our aim was not to set fast benchmark results, but to show that local search techniques are a suitable approach for finding branch decompositions on general graphs.

Table 1. Results for planar graphs

Graph	Vertices	Edges	$\beta(G)$	Result	Heuristic	Time (s)
a280	280	788	13	13	13	2138
bier127	127	368	14	14	14	177
ch130	130	377	10	10	10	142
ch150	150	432	12	12	12	266
d198	198	571	12	12	12	678
eil51	51	140	8	8	8	8
eil76	76	215	10	10	10	24
eil101	101	290	10	10/11	10	69
kroA100	100	285	9	9	9	64
kroA150	150	432	11	11	11	253
kroA200	200	586	11	11	11	707
kroB100	100	284	9	9	9	64
kroB150	150	436	10	10	10	227
kroB200	200	580	12	12	12	684
kroC100	100	286	9	9	9	73
kroE100	100	283	8	8	8	65
lin105	105	292	8	8/9	8	59
pr107	107	283	6	6/7	6	64
pr124	124	318	8	8	8	85
pr136	136	377	10	10	10	157
pr144	144	393	9	9	9	183
pr152	152	428	8	8	8	251
pr226	226	586	7	7	7	680
rat99	99	279	9	9	9	58
rd100	100	286	9	9	10	73
rd400	400	1183	17	17	18	9508
tsp225	225	622	12	12	12	862
u159	159	431	10	10/11	10	236

Table 2. Results for non-planar graphs

Graph	Vertices	Edges	$\tau(G)$	Result	Heuristic	Time (s)
alarm	37	65	4	4	5*	1
barley	48	126	7	6/7	7*	4
bcs01	48	176	13	12	12	17
bcs03	56	132	3	3	3	6
bcs04	132	1758	28-34	30	32	67644
bcs05	153	1135	20	18/19	20	10173
mildew	35	80	4	4	5*	1
myciel3	12	20	5	5	5*	1
myciel4	24	71	10	9/10	11*	1
myciel5	47	236	19	19/20	21*	72
myciel6	96	755	35	36/38	45*	4492
oesoca+	67	208	11	9	11*	23
oow-bas	27	54	4	4	5*	1
oow-solo	40	87	6	5	7*	2
oow-trad	33	72	6	5	6*	1
petersen	10	15	4	4	5*	1
queen5_5	25	320	18	16/17	18*	89
queen6_6	36	580	25	24	26*	898
ship-ship	50	114	8	7	9*	5
vsd-hugin	38	62	4	4	5*	1
water	32	123	9	9	9*	5
wilson-hugin	21	27	3	3	3*	1

We have applied our local search to a number of planar graphs for which the branchwidth can be found in [20,21]. The averaged results can be found in Table 1. In this table $\beta(G)$ is the branchwidth of the graph, *Heuristic* is the branchwidth obtained by heuristics of Hicks, and *Result* contains the branchwidth found by our local search. The first number is the result obtained by iterative hill climbing (performing random moves when stuck), and the second number is the result obtained without random moves. If only one result is mentioned then both methods yielded the same result. *Time* gives the average amount of time spent before reaching a local optimum. Our algorithm was able to find an optimal branch decomposition for all tested planar graphs. The heuristics of Hicks did not find an optimal branch decomposition for graphs rd100 and rd400. Most of our graphs can be obtained from the TreewidthLIB website: http://people.cs.uu.nl/hansb/treewidthlib.

We also tested our algorithm on a number of non-planar graphs. We used graphs for which the treewidth is known from [18,2]. This implies that we have a lower and upper bound for the branchwidth: a treewidth of τ ensures that the branchwidth is between $\lceil \frac{2}{3}\tau \rceil$ and τ. We also used the non-planar fill-in graphs from Hicks [20]. Table 2 gives the results of our algorithm. We only have the results of Hicks' heuristics for bcs01, bcs03, bcs04 and bcs05. For all other graphs we used our own implementation of the eigenvector heuristic of Cook and Seymour [12,13], those results are marked with a $(*)$. If the resulting branchwidth is larger than the treewidth $\tau(G)$ we know that the result is not optimal. For only one non-planar graph (myciel6) this is observed.

For all cases our algorithm performs at least as well as the heuristics used for comparison. Also note that not only the maximum width is minimised, but also the width of all other edges in the branch decomposition. This implies that the result is very suitable for use in a dynamic programming approach.

Scalability. The runtime of our algorithm highly depends on the number of edges in the input graph. To analyse the scalability of our algorithm, we created random graphs with 50 nodes and the number of edges varying between 50 and 1000, inclusive; each edge chosen uniformly over all not yet chosen pairs. Figure 3 illustrates the runtime of our implementation on these graphs. The drop at the end can be explained by the fact that a graph with 50 nodes and 1000 edges is almost a complete graph. The memory usage of our data structure is in the order $O(m^2)$.

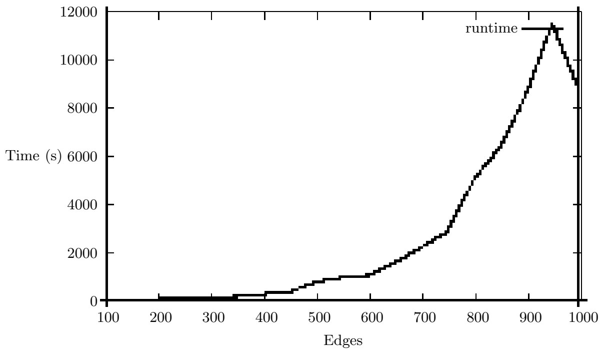


Fig. 3. Scalability of our algorithm

Note that our local search algorithm can be parallelised in a straightforward matter. One possible way to do this is to have several instances of *findBestTarget* running at the same time, and distribute the edges of the branch decomposition. As soon as an improvement is found the work of all other threads is discarded, and then process is repeated.

6 Conclusions and Future Work

We presented the first local search based algorithm for finding branch decompositions on general graphs. Our heuristic found an optimal branch decomposition for each planar graph used in our experiments. Moreover the heuristic also found branch decompositions of width lower than or equal to the treewidth for non-planar graphs with only one exception. Notice that the branchwidth of these graphs is still unknown and we therefore do not know whether these results are optimal or not. Furthermore our heuristic can be used to construct branch decompositions that are optimised for dynamic programming to solve a specific $\mathcal{NP}$-hard problem.

In this paper we have shown that local search based techniques are a promising approach to deal with the construction of branch decompositions. The behaviour of our algorithm indicates that improvements can be easily found for branch decompositions that are not in a local optimum. Moreover most local optima are near optimal (i.e. their width is close to the branchwidth). Finally our algorithm is capable of escaping the attraction of a local optimum, allowing iterated local search.

Possible improvements to the algorithm can consist of using advanced local search techniques, like simulated annealing or tabu search, or more refined selection criteria for which edges are moved. Another improvement could be to start with a better initial solution, as we currently start with a random initial solution, which hence can be expected to be of poor quality. An alternative would be to start with a solution generated by a heuristic, like one generated by the eigenvector method of Cook and Seymour [12,13].

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. IIS-0917072 and CNS-0751185. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

1. Alon, N.: Eigen values and expanders. Combinatorica 6, 83–96 (1986)
2. Bachoore, E.H., Bodlaender, H.L.: A branch and bound algorithm for exact, upper, and lower bounds on treewidth. In: Cheng, S.-W., Poon, C.K. (eds.) AAIM 2006. LNCS, vol. 4041, pp. 255–266. Springer, Heidelberg (2006)
3. Bian, Z., Gu, Q.-P.: Computing branch decompositions of large planar graphs. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 87–100. Springer, Heidelberg (2008)
4. Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small treewidth. SIAM Journal on Computing 25, 1305–1317 (1996)
5. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theoretical Computer Science 209, 1–45 (1998)
6. Bodlaender, H.L.: Discovering treewidth. In: Vojtáš, P., Bieliková, M., Charron-Bost, B., Sýkora, O. (eds.) SOFSEM 2005. LNCS, vol. 3381, pp. 1–16. Springer, Heidelberg (2005)
7. Bodlaender, H.L.: Treewidth: Characterizations, applications, and computations. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 1–14. Springer, Heidelberg (2006)
8. Bodlaender, H.L., Koster, A.M.C.A.: Combinatorial optimization on graphs of bounded treewidth. The Computer Journal 51(3), 255–269 (2008)
9. Bodlaender, H.L., Koster, A.M.C.A.: Treewidth computations I. Upper bounds. Information and Computation 208, 259–275 (2010)

10. Bodlaender, H.L., Thilikos, D.M.: Constructive linear time algorithms for branchwidth. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) ICALP 1997. LNCS, vol. 1256, pp. 627–637. Springer, Heidelberg (1997)
11. Clautiaux, F., Moukrim, A., Négre, S., Carlier, J.: Heuristic and meta-heuristic methods for computing graph treewidth. RAIRO Operations Research 38, 13–26 (2004)
12. Cook, W., Seymour, P.D.: An algorithm for the ring-routing problem. Bellcore technical memorandum, Bellcore (1993)
13. Cook, W., Seymour, P.D.: Tour merging via branch-decomposition. INFORMS Journal on Computing 15(3), 233–248 (2003)
14. Dorn, F., Penninkx, E., Bodlaender, H.L., Fomin, F.V.: Efficient exact algorithms on planar graphs: Exploiting sphere cut branch decompositions. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 95–106. Springer, Heidelberg (2005)
15. Dorn, F., Telle, J.A.: Semi-nice tree-decompositions: The best of branchwidth, treewidth and pathwidth with one algorithm. Discrete Applied Mathematics 157, 2737–2746 (2009)
16. Fomin, F.V., Mazoit, F., Todinca, I.: Computing branchwidth via efficient triangulations and blocks. Discrete Applied Mathematics 157, 2726–2736 (2009)
17. Fomin, F.V., Thilikos, D.M.: New upper bounds on the decomposability of planar graphs. Journal of Graph Theory 51, 53–81 (2006)
18. Gogate, V., Dechter, R.: A complete anytime algorithm for treewidth. In: Chickering, D.M., Halpern, J.Y. (eds.) Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence, UAI 2004, pp. 201–208. AUAI Press (2004)
19. Gu, Q.-P., Tamaki, H.: Optimal branch-decomposition of planar graphs in $O(n^3)$ time. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 373–384. Springer, Heidelberg (2005)
20. Hicks, I.V.: Branchwidth heuristics. Congressus Numerantium 159, 31–50 (2002)
21. Hicks, I.V.: Planar branch decompositions I: The ratcatcher. INFORMS Journal on Computing 17, 402–412 (2005)
22. Hicks, I.V.: Planar branch decompositions II: The cycle method. INFORMS Journal on Computing 17, 413–421 (2005)
23. Hicks, I.V., Koster, A.M.C.A., Kolotoğlu, E.: Branch and tree decomposition techniques for discrete optimization. In: Smith, J.C. (ed.) TutORials 2005. INFORMS Tutorials in Operations Research Series, INFORMS Annual Meeting, ch.1, pp. 1–29 (2005)
24. Hliněný, P., Oum, S., Seese, D., Gottlob, G.: Width parameters beyond tree-width and their applications. The Computer Journal 51, 326–362 (2008)
25. Kjærulff, U.: Optimal decomposition of probabilistic networks by simulated annealing. Statistics and Computing 2, 2–17 (1992)
26. Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. Journal of Algorithms 7, 309–322 (1986)
27. Robertson, N., Seymour, P.D.: Graph minors. X. Obstructions to tree-decomposition. Journal of Combinatorial Theory, Series B 52, 153–190 (1991)
28. Röhrig, H.: Tree decomposition: A feasibility study. Master's thesis, Max-Planck-Institut für Informatik, Saarbrücken, Germany (1998)
29. Seymour, P.D., Thomas, R.: Call routing and the ratcatcher. Combinatorica 14(2), 217–241 (1994)

Finding the Description of Structure by Counting Method: A Case Study

Ahti Peder and Mati Tombak

University of Tartu, Institute of Computer Science,
Tartu, Liivi 2, 50409, Estonia
ahti.peder@ut.ee, mati.tombak@ut.ee

Abstract. This paper presents a method to generate characterizing definitions for finite and parameterized structures. In particular, the method is applied to generate a conjecture for the properties characterizing a special class of graphs, called superpositional graphs.

The method can be used if the exact set of properties that describes a given finite structure cannot be found by pure thought but we can find the number of objects for small values of the parameter.

The next step is to codify the objects as assignments to a set of propositional variables, and the candidate properties as propositional formulae, in such a way that an object satisfies the property if and only if the assignment satisfies the formula. The main idea of this method is to find models that do not fit with the current approximation of the description of the structure and stepwise refine the logical description.

Finally, we "translate" the logical description into a mathematical one and prove it.

Keywords: superpositional graph, integer sequence, family of propositional formulae, $\#SAT$.

1 Introduction

The development of the method presented in this paper was motivated by the fact that the description of a special class of graphs, superpositional graphs with *classical properties* could not be found. The amount of properties found turned out to be incomplete.

Binary Decision Diagrams are graph representation of Boolean functions. They were first introduced by C. Lee [3] as a data structure for representing Boolean functions and further popularized by R. Bryant [1].

A good data structure is key to efficient Boolean function manipulation. In 1976, R. Ubar proposed in [10] a new data structure called *Structurally Synthesized Binary Decision Diagrams* (SSBDDs). However, for the first time they were introduced as *Structural Alternative Graphs*. The algorithms based on SSBDDs are used in the programs of digital diagnostics; SSBDDs provide an efficient opportunity for modeling digital systems for simulation purposes. One of the fastest fault simulator in the world is based on SSBDDs [11].

SSBDDs are based on *Superpositional Graphs* [6], akin to the binary decision diagrams that are based on binary graphs. Many properties of SSBDD depend on the properties of superpositional graphs [2]. An important problem for many applications of simulation purposes is: *given a binary graph, is it a superpositional graph*?

The purpose is to develop necessary and sufficient properties for describing superpositional graphs without using the superposition. The necessary decision problem for decomposing superpositional graphs – whether a binary graph is a superpositional graph – would be easy to solve if we knew the sequence of superpositions that was used for generating the graph. However, we do not have this sequence.

To find similar structures we tried to count n-node superpositional graphs and found from [7] that the resulting sequence coincides with the beginning of the sequence of large Schröder numbers. Unfortunately, this observation is not very helpful – we could not translate any of the problems, described by large Schröder numbers, into the theory of superpositional graphs. We can use only the integer sequence as a starting point of our search for a solution to the problem.

Our next goal is to describe a characteristic set of properties of the class SPG, knowing that this set must generate the detected integer sequence. One possibility to do it would be by means of propositional formulae [8]. We try to find a family of propositional formulae $\mathcal{F}_n$, which depends on parameter n so that $\#SAT(\mathcal{F}_n)$ is the number of n-node superpositional graphs. Choosing the propositional variables so that every assignment corresponds to some binary graph, we can interprete the formulae as the properties of the binary graphs. Using a special translator and a counter of satisfying assignments [5] we find the successive approximations until the corresponding integer sequence coincides with the target sequence.

2 Superpositional Graphs. Definitions and Properties

Definition 1. *A* binary graph *is an oriented acyclic connected graph with root and two terminals (sinks) –* 0 *and* 1*. Every internal node v has two sucessors: $high(v)$ and $low(v)$. Therefore, an edge $a \to b$ is a* 0-edge *(*1-edge*) if $low(a) = b$ ($high(a) = b$).*

Definition 2. *Let G and E be two binary graphs. A* superposition *of E into G instead of internal node v (denoted by $G_{v \leftarrow E}$) is a graph, which we obtain by deleting v from G and redirecting all edges, pointing to v, to the root of E, all edges of E, pointing to the terminal node* 1*, to the node $high(v)$ and all edges, pointing to the terminal node* 0*, to the node $low(v)$.*

Let A, C, and D be binary graphs, depicted in Fig. 1.

Definition 3. *A* class of superpositional graphs *(SPG) is defined inductively as follows:*

1° *graph $A \in SPG$.*

2° *if $G \in SPG$ and $v \in V(G) \setminus \{0, 1\}$, then $G_{v \leftarrow C} \in SPG$ and $G_{v \leftarrow D} \in SPG$.*

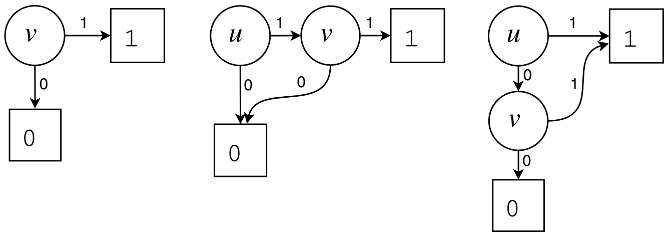


Fig. 1. Binary graphs A, C, and D.

Hereafter, if we say that $G \in SPG$ has n nodes, then the terminal nodes 0 and 1 are not included in this count. In the figures we direct 1-edges from left to right and 0-edges from up to down, without labels 1 and 0. The next theorem shows that the class of superpositional graphs is closed under superposition.

Theorem 1 ([6]). *If* $G, H \in SPG$ *and* $v \in V(G) \setminus \{0, 1\}$*, then* $G_{v \leftarrow H} \in SPG$.

An example in Fig. 2 characterizes the process of finding the graph $G_{v \leftarrow E}$.

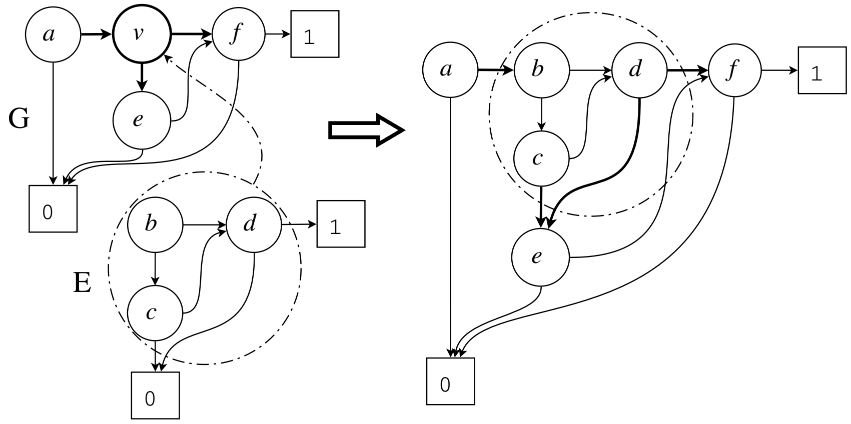


Fig. 2. The superposition $v \leftarrow E$ in the graph G.

In the next theorem we list common properties of the class SPG:

Theorem 2 ([6]). *Let* $G \in SPG$*. Then:*

1. *G has a unique root;*
2. *G is planar;*
3. *G is acyclic;*
4. *there exists a directed path through all intermediate nodes and this Hamiltonian path is unique (we say that G is uniquely traceable);*
5. *G is homogenous (only one type of edges enters into every node $v \in V(G)$).*

Since we could not find any more properties, we formulate a hypothesis:

Hypothesis 1. *A binary graph is a superpositional graph if and only if it is planar homogenous traceable graph.*

3 Structure of the Class *SPG*

All attempts to prove the hypothesis failed. Furthermore, there exists a 4-node binary graph, which has all these properties but is not a superpositional graph. On the graph in Fig. 3 a 0-edge starting from the node v should enter into the node z. This graph becomes planar if we draw the edge $v \to 0$ in a way that it does not cross edges $w \to z$ and $z \to 1$. Nevertheless we cannot find a sequence

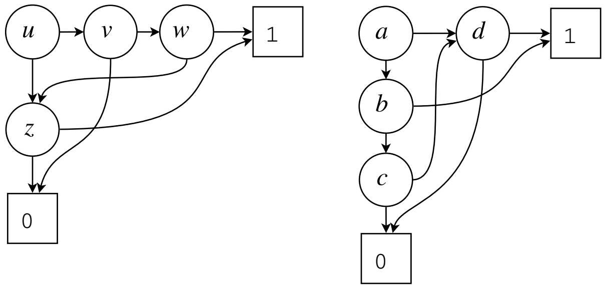


Fig. 3. The graphs satisfying the properties of the Theorem 2, but not being SPGs.

of elementary superpositions generating it. Hence, some unlisted properties are still missing.

To find similar structures, we try to count n-node superpositional graphs. By counting manually we find that the number of 1-node superpositional graphs is 1, 2-node 2, 3-node 6, 4-node 22 and 5-node 90. The sequence $1, 2, 6, 22, 90, \ldots$ is long enough to check whether there exists a structure that describes the same sequence. Sequences like this and references to similar structures can be found in The Encyclopedia of Integer Sequences [7]. It turns out that the found sequence matches with large Schröder numbers (Sequence $A006318$ in [7]).

This observation is not very helpful – we could not translate any of the problems, described by large Schröder numbers, into the properties of binary graphs. We can use only the following integer sequence as a starting point of our search for a solution to the problem:

$$1, 2, 6, 22, 90, 394, 1806, 8558, 41586, 206098, 1037718, \ldots \tag{1}$$

4 Logical Description of the Class *SPG*

Our next goal is to find such a family of propositional formulae $\mathcal{F}_n$, which would depend on parameter n (the number of nodes in superpositional graph), and, when valuating integers $1, 2, 3, \ldots$ to the parameter n, would produce the Sequence (1) as the answer to the counting problem $\#SAT(\mathcal{F}_n)$.

Let us suppose $\mathcal{F}_n^*$ is the required propositional formula. When searching for the formula and getting some approximation $\mathcal{F}_n$ for it, it might happen that

$$\#SAT(\mathcal{F}_n^*) = \#SAT(\mathcal{F}_n)$$

holds if $n < k$, but does not hold if $n = k$. There are two possibilities:

– $\#SAT(\mathcal{F}_k^*) < \#SAT(\mathcal{F}_k)$, the described set of properties is underconstrained. Try to find some extra property or constrain some of them;
– $\#SAT(\mathcal{F}_k^*) > \#SAT(\mathcal{F}_k)$, the set of properties found is overconstrained. Try to weaken some property.

In the first case, there exists an assignment α, where $\mathcal{F}_k(\alpha) = true$, but the assignment α does not represent any superpositional graph. In the second case, there exists the assignment β that represents a certain superpositional graph, but $\mathcal{F}_k(\beta) = false$. An analysis of the received problematical assignments (graphs) provides us with information for improving the properties. The goal is to refine the logical description until the received integer sequence coincides with the Sequence (1). The last step is to prove by traditonal graph-theoretical methods that the set of properties found describes exactly the class SPG.

A bottleneck of this method could be the fact that we must calculate manually or write a separate program for each approximation, which constructs a propositional formula for every value to the family parameter. The problem will be solved by the translator for metaformulae [5], which translates the description of the family of propositional formulae into propositional formula corresponding to the value of the parameter.

4.1 Presentation of Binary Graph with Propositional Variables

According to Theorem 2, there exists a unique Hamiltonian path in $G \in SPG$, which gives a canonical enumeration of the nodes of G. Hereafter, let the Hamiltonian path of an n-node superpositional graph G consist of nodes $v_1, \ldots, v_n$; and let v_{n+1} represent both the terminal 0 and the terminal 1. Such an approach is not restrictive (the 0-edges (1-edges) can not point to the node 1 (0)) but enables a simpler description of properties.

The meaning of the propositional variables $x_{i,j}$ and $y_{i,j}$ is, respectively, the existence of 0-edge and 1-edge from node v_i to node v_j:

$$x_{i,j} = \begin{cases} 1, & \text{if there exists a 0-edge } v_i \to v_j; \\ 0, & \text{otherwise;} \end{cases}$$

$$y_{i,j} = \begin{cases} 1, & \text{if there exists a 1-edge } v_i \to v_j; \\ 0, & \text{otherwise.} \end{cases}$$

Since superpositional graphs are acyclic, then in case there exists an edge from node v_i to node v_j, certainly $i < j$. Therefore, propositional formulae for the class *SPG* are defined on the set

$$X = \{x_{i,j}, y_{i,j} : 1 \leq i < j \leq n+1\}$$

and the required formula $\mathcal{F}_n^*(X)$ should satisfy the condition

$$\mathcal{F}_n^*(\alpha) = \begin{cases} 1, & \text{if the graph represented by } \alpha \text{ is a superpositional graph;} \\ 0, & \text{otherwise} \end{cases}$$

for every assignment α to X.

4.2 Descriptive Properties of the Class *SPG* as Propositional Formulae

The properties of the class *SPG* described in Sect. 2 now be expressed by means of propositional formulae. In the descriptions, we use the operators *and*, *or*, *xor*, $\rightarrow$ (implication), $\neg$ (negation) and *exactlyone* (allowed operators and rules can be found in [5]). In each successive approximation, we add a new property or remove any excessive ones.

Approximation I: Binary graphs with unique Hamiltonian path. The fact that the graph is acyclic is considered already when selecting the variables.

There is exactly one edges (0-edge or 1-edge) between the successive nodes of Hamiltonian path (except between the last two nodes):

$$\mathcal{P}_1 \equiv \bigwedge_{1 \leq i \leq n-1} xor(x_{i,i+1}, y_{i,i+1})\,.$$

Each node (except the two last) has exactly one exiting 0-edge:

$$\mathcal{P}_2 \equiv \bigwedge_{1 \leq i \leq n} exactlyone(x_{i,j} : i < j \leq n+1)\,.$$

Each node (except the two last) has exactly one exiting 1-edge:

$$\mathcal{P}_3 \equiv \bigwedge_{1 \leq i \leq n} exactlyone(y_{i,j} : i < j \leq n+1)\,.$$

Considering that the previous properties must hold for every superpositional graph, an approximation of $\mathcal{F}_n^*$ for the class *SPG* is

$$\mathcal{F} \equiv \mathcal{P}_1 \& \mathcal{P}_2 \& \mathcal{P}_3\,.$$

When applying the translator of propositional formulae to $\mathcal{F}$ and hereafter the counter of satisfying assignments to the result, we get the Sequence $A000165$:

$$1, 2, 8, 48, 384, 3840, 46080, \ldots$$

With the use of the translator and the counter, we can be convinced that the corrresponding members of sequences of formulae $\mathcal{P}_1\&\mathcal{P}_2$, $\mathcal{P}_2\&\mathcal{P}_3$ and $\mathcal{P}_1\&\mathcal{P}_3$ are larger than in case of $\mathcal{F}$. Hence, we cannot skip any of these properties.

Approximation II: Adding homogeneity. Every SPG is homogenous (Theorem 2), i.e., 0-edges and 1-edges cannot enter the same internal node at the same time. As there exists also the Hamiltonian path, we can equally say that if a 0-edge $v_i \to v_j$ exists, then the edge $v_{j-1} \to v_j$ is a 0-edge:

$$\mathcal{P}_4 \equiv \bigwedge_{1 \le i \le (n-2)} \bigwedge_{(i+2) \le j \le n} (x_{i,j} \to x_{j-1,j}) .$$

Similarly, if there exists a 1-edge $v_i \to v_j$, then the edge $v_{j-1} \to v_j$ is an 1-edge:

$$\mathcal{P}_5 \equiv \bigwedge_{1 \le i \le (n-2)} \bigwedge_{(i+2) \le j \le n} (y_{i,j} \to y_{j-1,j}) .$$

Now the approximation of $\mathcal{F}_n^*$ for the class SPG is

$$\mathcal{F} \equiv \mathcal{P}_1\&\mathcal{P}_2\&\mathcal{P}_3\&\mathcal{P}_4\&\mathcal{P}_5$$

and the corresponding sequence for it is $A000142$:

$$1, 2, 6, 24, 120, 720, \ldots$$

However, while the number of 4-node superpositional graphs must be 22, it is obvious that in the set of received assignments there are 2 assignments that do not represent any superpositional graph. Therefore, there must exist a yet undeclared property that appears in case $n \ge 4$.

Approximation III: Adding the strong planarity. To better visualize the properties of superpositional graphs, hereafter, we draw them in such way that their Hamiltonian paths are on straight lines. Analyzing the 24 assignments we received in case $n = 4$, we see that 2 of them do not represent any superpositional graph (Fig. 4). We guess that, in either case, the problem is caused by crossing

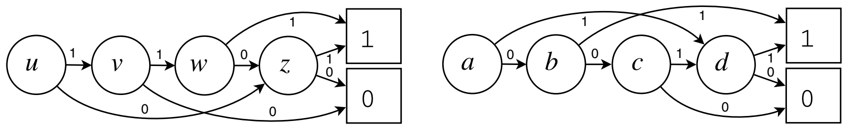


Fig. 4. Forbidden situations in case $\mathcal{F} \equiv \mathcal{P}_1\&\mathcal{P}_2\&\mathcal{P}_3\&\mathcal{P}_4\&\mathcal{P}_5$.

same type of edges. We raise a hypothesis that edges of the same type can not cross and add the corresponding properties to the approximation. Let $1 \le k < l < p < r \le n + 1$. According to the hypothesis, there can not be a situation where there the 0-edges $v_k \to v_p$ and $v_l \to v_r$ exist at the same time, i.e., 0-edges do not cross (Fig. 5):

$$\mathcal{P}_6 \equiv \bigwedge_{1 \le k < l < p < r \le n+1} (\overline{x_{k,p} \& x_{l,r}}) .$$

Herewith dually, 1-edges do not cross (Fig. 5):

$$\mathcal{P}_7 \equiv \bigwedge_{1 \le k < l < p < r \le n+1} (\overline{y_{k,p} \& y_{l,r}}) .$$

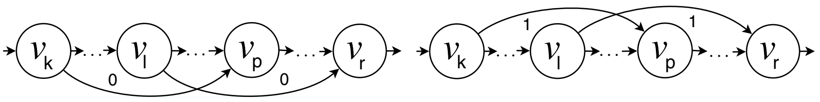


Fig. 5. Situations forbidden by the properties $\mathcal{P}_6$ and $\mathcal{P}_7$.

Definition 4. *We say that edges $v_k \to v_p$ and $v_l \to v_r$ are* crossing *edges if $k < l < p < r$.*

Definition 5. *We say that a binary traceable graph is* strongly planar *if it has no crossing* 0*-edges and no crossing* 1*-edges.*

Obviously, the properties $\mathcal{P}_6$ and $\mathcal{P}_7$ express the strong planarity. In [6] we see that if a binary graph is strongly planar, it is also planar. The opposite is generally not true. Now for the formula

$$\mathcal{F} \equiv \mathcal{P}_1 \& \mathcal{P}_2 \& \mathcal{P}_3 \& \mathcal{P}_4 \& \mathcal{P}_5 \& \mathcal{P}_6 \& \mathcal{P}_7$$

we get the Sequence $A001181$:

$$1, 2, 6, 22, 92, 422, \ldots$$

We see that the situation is better, but our description is nevertheless incorrect (the accurate sequence is $1, 2, 6, 22, 90, 394, \ldots$). One could suppose that either some problem connected with the described properties is more general than the one we described or some yet unknown property is missing.

Approximation IV: Removing the homogeneity. By using the translator and the counter, we can see that one can omit the properties $\mathcal{P}_4$ and $\mathcal{P}_5$ from the set of properties. Consequently, we can propose a hypothesis that the formula

$$\mathcal{P}_1 \& \mathcal{P}_2 \& \mathcal{P}_3 \& \mathcal{P}_6 \& \mathcal{P}_7 \longrightarrow \mathcal{P}_4 \& \mathcal{P}_5$$

is a tautology and it can be proved (Theorem 7 in [6]):

Theorem 3. *If a binary traceable graph is strongly planar, then it is also homogenous.*

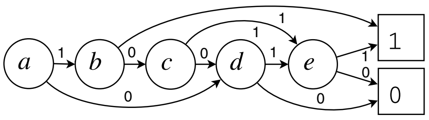


Fig. 6. A forbidden situation in case $\mathcal{F} \equiv \mathcal{P}_1 \& \mathcal{P}_2 \& \mathcal{P}_3 \& \mathcal{P}_6 \& \mathcal{P}_7$.

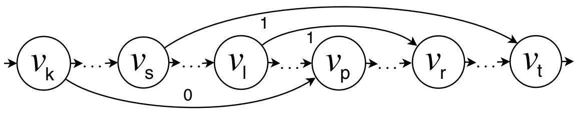


Fig. 7. Situation forbidden by the property $\mathcal{P}_8$.

Hence, the new approximation for the formula $\mathcal{F}_n^*$ is

$$\mathcal{F} \equiv \mathcal{P}_1 \& \mathcal{P}_2 \& \mathcal{P}_3 \& \mathcal{P}_6 \& \mathcal{P}_7 \,.$$

Approximation V: Adding the cofinality. We analyze the two assignments emerging from the last approximation in case $n = 5$. One of the found problematic graphs is depicted in Fig. 6. The second problematic graph is obviously dual to the first one, i.e., it can be found from the other by exchanging 0- and 1-edges. We suppose that, due to existence of the 0-edge $a \to d$, the 1-edges $b \to 1$ and $c \to e$ should point to the same node, i.e., the situation in Fig. 7 is forbidden:

$$\mathcal{P}_8 \equiv \bigwedge_{1 \le k < s < l < p < r < t \le n+1} (\overline{x_{k,p} \& y_{l,r} \& y_{s,t}}) \,.$$

Analogically: all 0-edges that cross the same 1-edge must point to the same node:

$$\mathcal{P}_9 \equiv \bigwedge_{1 \le k < s < l < p < r < t \le n+1} (\overline{y_{k,p} \& x_{l,r} \& x_{s,t}}) \,.$$

Definition 6. *We say that a binary traceable graph is* 1-cofinal (0-cofinal) *if all* 1*-edges (*0*-edges), starting between the endpoints of some* 0*-edge (*1*-edge) and crossing it, are entering into the same node.*

Definition 7. *We say that a binary traceable graph is* cofinal *if it is* 1*-cofinal and* 0*-cofinal.*

After adding the properties expressing cofinality $\mathcal{P}_8$ and $\mathcal{P}_9$ to the approximation and running the counter, we get the Sequence (1).

Using the translator and the counter, we establish that none cane be omitted from the set of properties $\{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_6, \mathcal{P}_7, \mathcal{P}_8, \mathcal{P}_9\}$. We checked that in case $n = 1 \ldots 9$, this set of properties has exactly the same number of satisfying assignments as is the number of n-noded superpositional graphs. Hence, the class SPG can probably be described by the formula

$$\mathcal{F} \equiv \mathcal{P}_1 \& \mathcal{P}_2 \& \mathcal{P}_3 \& \mathcal{P}_6 \& \mathcal{P}_7 \& \mathcal{P}_8 \& \mathcal{P}_9$$

or written directly, not using composition:

$$\begin{aligned}
\mathcal{F} \equiv & \bigwedge_{1 \le i \le n-1} xor(x_{i,i+1}, y_{i,i+1}) \\
& \& \bigwedge_{1 \le i \le n} exactlyone(x_{i,j} : i < j \le n+1) \\
& \& \bigwedge_{1 \le i \le n} exactlyone(y_{i,j} : i < j \le n+1) \\
& \& \bigwedge_{1 \le k < l < p < r \le n+1} (\overline{x_{k,p} \& x_{l,r}}) \\
& \& \bigwedge_{1 \le k < l < p < r \le n+1} (\overline{y_{k,p} \& y_{l,r}}) \\
& \& \bigwedge_{1 \le k < s < l < p < r < t \le n+1} (\overline{x_{k,p} \& y_{l,r} \& y_{s,t}}) \\
& \& \bigwedge_{1 \le k < s < l < p < r < t \le n+1} (\overline{y_{k,p} \& x_{l,r} \& x_{s,t}}) \,.
\end{aligned}$$

It is clear that if the description is not correct, then an inaccuracy or even a missing property can appear only in case $n \ge 10$. Therefore, we can raise the hypothesis:

Hypothesis 2. *A binary graph is a superpositional graph if and only if it is a strongly planar cofinal traceable graph.*

We proved this hypothesis in [6] (Theorem 9).

5 Discussion

In the process of approximating the logical description we got some integer sequences, characterizing certain subclasses of binary graphs. All these sequences are present in The Encyclopedia of Integer Sequences. This indicates that these subclasses of binary graphs may have some connections with mathemathical structures described by these sequences. Therefore it might be interesting to find bijections between

– double-downgrading permutations and traceable binary graphs ($A000165$);
– permutations and homogeneous binary graphs ($A000142$);
– Baxter permutations and strongly planar binary graphs ($A001181$);
– separable permutations and superpositional graphs ($A006318$).

We have to admit that the real process of finding the hypothesis was not so smooth as described above. The SSBDD-community used to draw diagrams without terminal nodes. The reason is that all edges, pointing to the terminal nodes can be restored unambiguously due to homogeneity of the superpositional graph. In our first attempt [4] (see Sect. 4.2) we followed this tradition and defined propositional variables only for edges between internal nodes.

Similarly to the previous description, let the Hamiltonian path of an n-node SPG G consists of nodes $v_1, \ldots, v_n$. The meaning of the propositional variables $x_{i,j}$ and $y_{i,j}$ is, respectively, the existence of 0-edge and 1-edge from node v_i to node v_j. After rather sophisticated series of iterations we received a formula, which described the Sequence (1):

$$\begin{aligned}
\mathcal{P} \equiv & \bigwedge_{1\le i\le n-1} xor(x_{i,i+1}, y_{i,i+1}) \\
& \& \bigwedge_{1\le i\le n-1} atmostone(x_{i,j} : i < j \le n) \\
& \& \bigwedge_{1\le i\le n-1} atmostone(y_{i,j} : i < j \le n) \\
& \& \bigwedge_{1\le i\le (n-2)} \bigwedge_{(i+2)\le j\le n} (x_{i,j} \rightarrow x_{j-1,j}) \\
& \& \bigwedge_{1\le i\le (n-2)} \bigwedge_{(i+2)\le j\le n} (y_{i,j} \rightarrow y_{j-1,j}) \\
& \& \bigwedge_{1\le k<l<p<r\le n} ((x_{k,p} \& y_{l,r}) \rightarrow (\bigwedge_{k\le s\le (l-1)} (\bigvee_{(s+1)\le t\le r} y_{s,t}))) \\
& \& \bigwedge_{1\le k<l<p<r\le n} ((y_{k,p} \& x_{l,r}) \rightarrow (\bigwedge_{k\le s\le (l-1)} (\bigvee_{(s+1)\le t\le r} x_{s,t}))) \\
& \& \bigwedge_{1\le i<j\le n-1} (y_{i,j+1} \rightarrow (\bigwedge_{i\le k\le j-1} (x_{k,k+1} \rightarrow (\bigvee_{k+1\le p\le j+1} y_{k,p})))) \\
& \& \bigwedge_{1\le i<j\le n-1} (x_{i,j+1} \rightarrow (\bigwedge_{i\le k\le j-1} (y_{k,k+1} \rightarrow (\bigvee_{k+1\le p\le j+1} x_{k,p})))) \,.
\end{aligned}$$

Detailed description of the derivation of the formula and formulation of properties can be found in [4] (see Sect 4.2). It was impossible to define "nice" graph-theoretic properties, corresponding to the parts of the formula, due to the complexity of the formula. Hence, inevitably, the question arises whether our chosen set of propositional variables was suitable.

It appeared that the reason of the complexity was the indirect description of the properties of the edges, pointing to the terminal nodes. By enhancing the set of variables (Sect 4.1 in this paper) we got the opportunity to describe explicitly these properties.

Acknowledgements

This work has been supported by Estonian Ministry of Education and Research targeted financing grants $SF0182712s06$ and $SF0140013s10$.

References

1. Bryant, R.: Graph-based Algorithms for Boolean Function Manipulation. IEEE Transaction on Computers C 35, 677–691 (1986)
2. Jutman, A., Peder, A., Raik, J., Tombak, M., Ubar, R.: Structurally Synthesized Binary Decision Diagrams. In: 6th International Workshop on Boolean Problems, Freiberg University, pp. 271–278 (2004)
3. Lee, C.: Representation of Switching Circuits by Binary Decision Diagrams. Bell. Syst. Tech. Journal 38, 985–999 (1959)
4. Peder, A: Superpositional Graphs and Finding the Description of Structure by Counting Method. Phd thesis, Tartu (2010), `http://hdl.handle.net/10062/14853`
5. Peder, A., Isotamm, A., Tombak, M.: A Meta-Compiler for Propositional Formulae. In: Seventh Symposium on Programming Languages and Software Tools, Szeged, Hungary, pp. 250–261 (2001)
6. Peder, A., Tombak, M.: Superpositional Graphs. Acta et Commentationes Universitatis Tartuensis de Mathematica 13, 51–64 (2009), `http://www.ut.ee/~ahtip/LOG_COMP/PederTombakACTA2009.pdf`
7. Sloane, N.: On-Line Encyclopedia of Integer Sequences, `http://www.research.att.com/~njas/sequences/`
8. Stamm-Wilbrandt, H.: Programming in Propositional Logic or Reductions: Back to the Roots (Satisfiability). Technical reports of Department of Computer Science III, University of Bonn, pp 24 (1993)
9. Stanley, R.P.: Hipparchus, Plutarch, Schröder and Hough. Am. Math. Monthly 104(4), 344 (1997)
10. Ubar, R.: Test Generation for Digital Circuits Using Alternative Graphs (in Russian). In: Proc. Tallinn Technical University, 409, Tallinn TU, Estonia, pp. 75–81 (1976)
11. Ubar, R., Devadze, S., Raik, J., Jutman, A.: Parallel X-fault Simulation with Critical Path Tracing Technique. In: Proc. of the IEEE/ACM DATE Conference, Dresden, Germany, p. 6 (2010)

On Approximating the d-Girth of a Graph

David Peleg[1], Ignasi Sau[2], and Mordechai Shalom[3]

[1] Department of Computer Science, Weizmann Institute of Science, Rehovot, Israel
david.peleg@weizmann.ac.il
[2] AlGCo project-team, CNRS, Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), Montpellier, France
ignasi.sau@lirmm.fr
[3] TelHai Academic College, Upper Galilee, 12210, Israel
cmshalom@telhai.ac.il

Abstract. For a finite, simple, undirected graph G and an integer $d \geq 1$, a *mindeg-d subgraph* is a subgraph of G of minimum degree at least d. The *d-girth* of G, denoted $g_d(G)$, is the minimum size of a mindeg-d subgraph of G. It is a natural generalization of the usual girth, which coincides with the 2-girth. The notion of d-girth was proposed by Erdős *et al.* [13,14] and Bollobás and Brightwell [7] over 20 years ago, and studied from a purely combinatorial point of view. Since then, no new insights have appeared in the literature. Recently, first algorithmic studies of the problem have been carried out [2,4]. The current article further explores the complexity of finding a small mindeg-d subgraph of a given graph (that is, approximating its d-girth), by providing new hardness results and the first approximation algorithms in general graphs, as well as analyzing the case where G is planar.

Keywords: generalized girth, minimum degree, approximation algorithm, hardness of approximation, randomized algorithm, planar graph.

1 Introduction

Degree-constrained subgraph problems have attracted considerable attention in the last decades, resulting in a large body of literature (see e.g. [15,1,2,14,17,4, 13,8,23,21]). Beyond the theoretical importance of these problems, the reasons for such intensive study are mainly rooted in their wide applicability in the areas of interconnection networks and routing algorithms, among others. This article studies the computational complexity of one such problem, presented next.

For a finite, simple, undirected graph G and an integer $d \geq 1$, a *mindeg-d subgraph* is a subgraph of G of minimum degree at least d. The *d-girth* of G, denoted $g_d(G)$, is the minimum size of a mindeg-d subgraph of G. The notion of d-girth was proposed and studied by Erdős *et al.* [13,14] and Bollobás and Brightwell [7] (using different terminology). Combinatorial bounds on the d-girth of a graph can also be found in [18,6]. For $d = 2$, $g_2(G)$ coincides with the *girth* of G, hence the d-girth can be seen as a natural generalization of the usual girth. Our interest is in the corresponding optimization problem of finding a minimum-size mindeg-d subgraph of a given graph (namely, one of size $g_d(G)$). For $d = 1$, this problem is trivial, as any edge constitutes an optimal solution. For $d = 2$, the problem corresponds exactly to finding the shortest cycle in G (as every subgraph of minimum degree at least 2 contains a cycle), and thus can be solved in polynomial time. For a fixed integer $d \geq 1$, our optimization problem is formally defined as follows.

The d-GIRTH Problem
Input: A simple undirected graph $G = (V, E)$.
Output: A minimum-size subset $S \subseteq V$ such that $G[S]$ has minimum degree at least d.

Note that $\mathrm{OPT}_{d\text{-GIRTH}}(G) = g_d(G)$. Until very recently, the computational complexity of the d-GIRTH problem had not been studied in the literature. It has been proved in [2] that for any fixed $d \geq 3$, the d-girth of a graph cannot be approximated within any constant factor, unless P = NP. Concerning approximation algorithms, the only positive result is an $\mathcal{O}(n/\log n)$-approximation algorithm for minor-free graphs [2]; approximation algorithms for the d-GIRTH problem in general graphs were missing in the literature. On the other hand, the problem has been recently studied in [4] from the *parameterized complexity* point of view [12], taking as the parameter the number of vertices in a solution. It was shown that the problem is $W[1]$-hard in general graphs, and admits FPT algorithms in minor-free families of graphs [4].

It is worth mentioning that the d-GIRTH problem is closely related to the *traffic grooming* problem, which is fundamental in modern optical networks. Loosely speaking, an important particular case of the traffic grooming problem can be stated, in graph-theoretical terms, as partitioning the edges of a given graph into subgraphs with bounded number of edges, while minimizing the total number of vertices in the partition. Traffic grooming has been proved to be a computationally hard problem [3,9], and good approximation algorithms for the d-GIRTH problem would directly translate into efficient approximation algorithms for traffic grooming. See [4] for more details about this relation.

Our results. Section 2 focuses on hardness results. The hardness results of [2] are substantially improved by proving that for any $d \geq 3$ and any $\varepsilon > 0$, there is no polynomial-time algorithm for the d-GIRTH problem with approximation ratio $2^{\mathcal{O}(\log^{1-\varepsilon} n)}$ unless $\mathrm{NP} \subseteq \mathrm{DTIME}\left(2^{\mathcal{O}(\log^{1/\varepsilon} n)}\right)$. These hardness results hold even in graphs with degrees d or $d+1$. Section 3 provides the first approximation algorithms for the d-GIRTH problem in general graphs. We first present a randomized algorithm with approximation ratio $n/\log n$ in Section 3.1. We then present another randomized algorithm with better performance in high-degree graphs (Section 3.2), and a deterministic algorithm for low-degree graphs (Section 3.3). In Section 4 we turn to the case where the input graph is planar. We prove that the d-GIRTH problem is NP-hard in planar graphs for $d \in \{3, 4, 5\}$ (Section 4.1), present a deterministic approximation algorithm (Section 4.2), and show that the problem can be solved exactly in *subexponential* time (Section 4.3). A concluding discussion appears in Section 5.

We would like to point out that in view of our results, the d-GIRTH problem appears to be rather difficult. Although the approximation ratios obtained are in some sense weak, the performance of our algorithms is not far from the best approximation algorithms for other very hard graph optimization problems like MAXIMUM CLIQUE, CHROMATIC NUMBER, or LONGEST PATH. Our work will hopefully trigger further research on the d-GIRTH problem.

Notation. All the graphs considered in this paper are finite, simple, and undirected. We use standard graph terminology, see for instance [10]. Unless stated otherwise, we denote the number of vertices of the input graph G by n. We use $\deg_G(v)$, $\delta(G)$, and $\Delta(G)$ to denote the *degree* of a vertex v in G, the *minimum degree* of G, and the *maximum degree* of G, respectively. We use $H \subseteq G$ to denote the fact that H is a *subgraph* of G. Given a subset $S \subseteq V(G)$, $G[S]$ denotes the subgraph of G *induced* by the vertices in S. For convenience, we use 'log' to denote the natural logarithm.

2 Hardness Results for General Graphs

It is proved in [2] that for any $d \geq 3$, the d-GIRTH problem is not in APX unless P = NP. Theorem 2 given in this section improves the hardness results of [2], relying on a slightly stronger complexity assumption. The ideas are inspired mainly by [17], and the proof builds upon the reductions and the constructions presented in [2]. Before proceeding to the improved hardness in Section 2.2, we first describe in Section 2.1 the families of graphs constructed in [2].

2.1 Preliminaries: Some Families of Graphs

For the sake of intuition, it is helpful throughout this section to think about the case $d = 3$. Given a fixed integer $d \geq 3$, we proceed to construct a class of graphs $\mathcal{G}_d$ starting from the class of d-regular graphs whose number of edges is $d \cdot (d-1)^{\ell}$ for some positive integer ℓ. Given such a d-regular graph H, with $|V(H)| = n$, we construct a graph $G = f(H) \in \mathcal{G}_d$ as follows. By assumption, we have that $|E(H)| = nd/2 = d \cdot (d-1)^{\ell}$ for some integer ℓ. Let T be the complete d-ary rooted tree (that is, internal vertices have degree d) with root r and height $\ell + 1$, which has $d \cdot (d-1)^{\ell}$ leaves and $1 + d \cdot \left((d-1)^{\ell+1} - 1\right)/(d-2)$ vertices overall. We identify the leaves of T with the elements in $E(H)$, and denote this set – slightly abusing notation – by E (that is, $E \subseteq V(T)$). We add another copy of E, called F, and $d-1$ edge-disjoint perfect matchings on $E \cup F$, inducing a bipartite graph with partition classes E and F. We also identify the vertices of F with the elements in $E(H)$. Now we add a set A of $|V(H)|$ new vertices identified with the elements in $V(H)$, and join them to the vertices in F according to the incidence relations in H: we add an edge between a vertex in F corresponding to $e \in E(H)$ and a vertex in A corresponding to $u \in V(H)$

if and only if e contains u. This completes the construction of G. Note that the vertices in E have degree d, and those in F have degree $d+1$. An illustration of such a graph G for $d=3$ can be found in [22]. The important property of these graphs is that any solution to the d-GIRTH problem contains all vertices of G, except possibly some vertices in set A (see [2] for more details).

We now define a *graph squaring* operation for graphs in the family $\mathcal{G}_d$. Given a graph $G \in \mathcal{G}_d$, we describe the construction of G^2, and repeating inductively k times the same construction defines the graph G^{2^k}, a typical element of the class $\mathcal{G}_d^{2^k} = \{G^{2^k} \mid G \in \mathcal{G}_d\}$, for any $k \geq 0$. For every vertex v in G, construct a graph G_v as follows: first, take a copy of G, and choose $d_v = \deg_G(v)$ other arbitrary vertices $x_1, \ldots, x_{d_v}$ of degree d in $T \subseteq G$. Then, replace each of these vertices x_i by the following:

- if $d \geq 3$ is odd: a graph obtained from K_{d+1} by removing a perfect matching (e.g., a C_4 for $d=3$).
- if $d \geq 4$ is even: a graph obtained from K_{d+2} by removing a cycle going through $d+1$ vertices. Let v^* be the vertex of degree $d+1$ in this graph.

Next, join d of the vertices of this new graph (different from v^*) to the d neighbors of x_i, $i = 1, \ldots, d_v$. Let G_v be the graph obtained in this way. Note that G_v has exactly d_v vertices of degree $d-1$. Now, take a copy of G, and replace each vertex v by G_v. Then, join each of the d_v neighbors of v in G to one of the d_v vertices of degree $d-1$ in G_v. This completes the construction of the graph G^2. We have that $|V(G^2)| = |V(G)|^2 + o(|V(G)|^2)$, because each vertex of G gets replaced by a copy of G where some of the vertices were replaced by a graph of size $d+1$ or $d+2$. An illustration of G^2 for $d=3$ can be found in [22].

Theorem 1 ([2]). *For any fixed $d \geq 3$, finding a constant-factor polynomial-time approximation algorithm for the d-GIRTH problem in the class of graphs $\bigcup_{k \geq 0} \mathcal{G}_d^{2^k}$ is NP-hard.*

2.2 Improved Hardness Results

The following technical lemma is a consequence of the constructions of Section 2.1.

Lemma 1. *For any $d \geq 3$, let G be a graph of the class $\mathcal{G}_d$ constructed in Section 2.1, and let G^2 be the graph constructed from G by the graph squaring operation.*

(i) *If $g_d(G) = \ell$, then $g_d(G^2) \leq 2\ell^2$; and*
(ii) *given a solution in G^2 containing m vertices, we can obtain in polynomial time a solution in G containing at most $\sqrt{m}$ vertices.*

Proof: The first claim follows from the fact that, given a solution $S \subseteq V(G)$ to the d-GIRTH problem in G, a feasible solution S_2 to the d-GIRTH problem in the square graph G^2 can be obtained by choosing the copies of G corresponding to vertices in S, and by choosing again in each such copy the vertices defined by S. From the construction of G^2 it follows that $|S_2| \leq |S| \cdot (|S| + (d+1)^2) \leq 2|S|^2$, as the degree of any vertex $v \in V(G)$ is at most $d+1$, and in the copy of G in G^2 corresponding to vertex v, $\deg(v)$ vertices are replaced by graphs on at most $d+2$ vertices.

In order to prove the second claim, let $S_2 \subseteq V(G^2)$ be a solution to the d-GIRTH problem in G^2, with $|S_2| = m$. We distinguish two cases. First, if S_2 contains vertices from fewer than $\sqrt{m}$ copies of G, then the solution in G defined by the vertices corresponding to these copies has size at most $\sqrt{m}$. Otherwise, there exists a copy G_v of G intersecting S_2 in which at most $\sqrt{m}$ vertices belong to S_2. Then, the solution in G defined by the vertices in G_v belonging to S_2 contains at most $\sqrt{m}$ vertices. $\square$

Theorem 2. *For any $d \geq 3$ and any $\varepsilon > 0$, there is no polynomial-time algorithm for the d-GIRTH problem with approximation ratio $2^{\mathcal{O}(\log^{1-\varepsilon} n)}$ unless $\mathrm{NP} \subseteq \mathrm{DTIME}\left(2^{\mathcal{O}(\log^{1/\varepsilon} n)}\right)$. The theorem holds even for the class of graphs with minimum degree d and maximum degree $d+1$.*

Proof: Let $d \geq 3$ and $\varepsilon > 0$ be fixed, and suppose that there exists a polynomial-time approximation algorithm $\mathcal{A}$ that approximates the d-GIRTH problem within a ratio $2^{\mathcal{O}(\log^{1-\varepsilon} n)}$. Let $G = (V, E)$ be an instance of the d-GIRTH problem belonging to the class of graphs $\mathcal{G}_d$ defined in Section 2.1, with $|V| = n$ and $g_d(G) = \ell$. For a positive integer k, let G^{2^k} be the graph obtained from G by applying k times the graph squaring operation defined in Section 2.1. Note that for any $k \geq 0$, the vertices of G^{2^k} have degree d or $d+1$, and that $|V(G^{2^k})| = n^{2^k} + o(n^{2^k}) = \Theta(n^{2^k})$. Let p be the smallest integer such that $N = |V(G^{2^p})| \geq 2^{\log^{1/\varepsilon} n}$. Note that $N = \Theta(n^{2^p})$, so $2^p = \Theta(\log N / \log n) = \Theta(\log^{1-\varepsilon} N)$. Consequently, $2^{\mathcal{O}\left(\frac{\log^{1-\varepsilon} N}{2^p}\right)} = \mathcal{O}(1)$. By repeatedly applying Lemma 1(i), it follows that $g_d(G^{2^p}) \leq 2^{2^p - 1} \cdot \ell^{2^p}$. Then, algorithm $\mathcal{A}$ finds in time polynomial in N a solution to the d-GIRTH problem in G^{2^p} of size at most $2^{2^p-1} \cdot \ell^{2^p} \cdot 2^{\mathcal{O}(\log^{1-\varepsilon} N)}$. Then, by repeatedly applying Lemma 1(ii), we can find a solution to the d-GIRTH problem in G of size at most

$$\left(2^{2^p-1} \cdot \ell^{2^p} \cdot 2^{\mathcal{O}(\log^{1-\varepsilon} N)}\right)^{1/2^p} \;\leq\; \ell \cdot 2^{\mathcal{O}\left(\frac{\log^{1-\varepsilon} N}{2^p}\right)} \;=\; \mathcal{O}(\ell).$$

This implies that the d-GIRTH problem can be approximated in the class $\bigcup_{k \geq 0} \mathcal{G}_d^{2^k}$ within a constant factor in time polynomial in N, that is, in time $2^{\mathcal{O}(\log^{1/\varepsilon} n)}$. But since finding a constant-factor approximation algorithm for the d-GIRTH problem in $\bigcup_{k \geq 0} \mathcal{G}_d^{2^k}$ is NP-hard by Theorem 1, it follows that $\mathrm{NP} \subseteq \mathrm{DTIME}\left(2^{\mathcal{O}(\log^{1/\varepsilon} n)}\right)$. $\square$

3 Approximation Algorithms for General Graphs

Section 3.1 presents a randomized approximation algorithm in general graphs and a detailed analysis of its approximation ratio. Section 3.2 proposes another randomized algorithm for graphs with high minimum degree, and discusses the relation of this algorithm with a combinatorial result of Erdős *et al.* [14]. Finally, Section 3.3 presents a deterministic approximation algorithm for graphs with low maximum degree.

3.1 A Randomized $(n/\log n)$-Approximation

Theorem 3. *For any $d \geq 3$, the d-*GIRTH *problem admits a polynomial-time randomized approximation algorithm with ratio $n/\log n$.*

Proof: A graph G is said to be *valid* if $\delta(G) \geq d$ and $V(G) \neq \emptyset$. Consider the subroutine REDUCE(G, v), that given a valid graph G and $v \in V(G)$, finds the maximum (not necessarily proper) induced subgraph of $G \setminus \{v\}$ with $\delta(G') \geq d$.

```
Procedure REDUCE($G, v$)
    $G' = G$
    remove $v$ and all its incident edges from $G'$
    while($\delta(G') < d$ and $V(G') \neq \emptyset$)
        { choose an arbitrary node $v' \in V(G')$ with degree less than $d$
          remove $v'$ and all its incident edges from $G'$ }
    return $G'$.
```

Clearly the graph returned by Procedure REDUCE is either empty or valid. We now consider the following randomized algorithm.

```
Algorithm RANDOMREDUCE($G$)
    while($G \neq \emptyset$)
        { RR $\leftarrow G$
          pick a node $v$ of $G$ uniformly at random
          $G \leftarrow$ REDUCE($G, v$) }
    return RR.
```

Clearly the algorithm returns a valid subgraph RR. We now analyze its performance. Assume the algorithm performs k iterations. Let G_i be the graph after iteration i, and $n_i = |V(G_i)|$. Clearly, $n_{\text{RR}} = |V(\text{RR})| = n_k < n_{k-1} < \ldots < n_1 < n_0 = n$. Let OPT be some minimum size valid subgraph of G, i.e., an optimal solution to the d-GIRTH problem in G, and let $\rho(n)$ be the approximation ratio of the algorithm (to be fixed later). Let $n_{\text{OPT}} = |V(\text{OPT})| = g_d(G)$. Consider the event that the algorithm is successful in finding a valid subgraph of the desired size, and the sub-event that the subgraph found by the algorithm happens to contain the optimal solution OPT, namely,

$$\begin{aligned} \textsc{Succ} &= (n_{\text{RR}} \leq \rho(n) \cdot n_{\text{OPT}}), \\ \textsc{Succ}^+ &= \textsc{Succ} \wedge (V(\text{OPT}) \subseteq V(\text{RR})). \end{aligned}$$

Then

$$\begin{aligned}\mathbf{Pr}[\textsc{Succ}] \;\geq\; \mathbf{Pr}[\textsc{Succ}^+] \;=\; \mathbf{Pr}[\, n_{\text{RR}} \leq \rho(n)\cdot n_{\text{OPT}} \wedge V(\text{OPT}) \subseteq V(\text{RR})\,] \\ = \prod_{i=0}^{k}\left(\frac{n_i - n_{\text{OPT}}}{n_i}\right).\end{aligned}$$

The last equality holds because $V(\text{OPT}) \subseteq V(G_i)$ for every $i \leq k$. If $V(\text{OPT}) \subseteq V(G_i)$, then $V(\text{OPT}) \subseteq V(G_{i+1})$ if and only if the node v chosen in iteration i is not in $V(\text{OPT})$, which happens with probability $(n_i - n_{\text{OPT}})/n_i$.

Note that at each step at least one node is removed, thus $k \leq n - n_{\text{RR}}$. For fixed RR, the minimum of the last expression is attained when $k = n - n_{\text{RR}}$, which implies $n_i = n - i$. Therefore,

$$\mathbf{Pr}[\textsc{Succ}] \;\geq\; \prod_{i=0}^{n-n_{\text{RR}}}\left(\frac{n-i-n_{\text{OPT}}}{n-i}\right) \;=\; \frac{n-n_{\text{OPT}}}{n}\;\cdots\;\frac{n_{\text{RR}}-n_{\text{OPT}}}{n_{\text{RR}}}\;.$$

If $n_{\text{OPT}} = \Omega(n/\rho(n))$, then any solution is a $\rho(n)$-approximation, so we assume $n_{\text{OPT}} = o(n/\rho(n))$, implying $n_{\text{RR}} = o(n)$, and therefore $n - n_{\text{RR}} \geq n_{\text{OPT}}$. Then the highest $n - n_{\text{RR}} - n_{\text{OPT}}$ factors of the nominator cancel out with the corresponding lowest ones in the denominator. We obtain

$$\begin{aligned}\mathbf{Pr}[\textsc{Succ}] &\geq \frac{n_{\text{RR}}-1}{n}\;\cdots\;\frac{n_{\text{RR}}-n_{\text{OPT}}}{n-n_{\text{OPT}}+1} \geq \left(\frac{n_{\text{RR}}-n_{\text{OPT}}}{n}\right)^{n_{\text{OPT}}} \\ &= \Omega\left(\left(\frac{n_{\text{RR}}}{n}\right)^{n_{\text{OPT}}}\right) = \Omega\left(\left(\frac{\rho(n)\cdot n_{\text{OPT}}}{n}\right)^{n_{\text{OPT}}}\right). \qquad (1)\end{aligned}$$

Let $f(n) = \log n / n_{\text{OPT}}$. Then $\log n/n \leq f(n) \leq \log n$. By taking $\rho(n) = (n/\log n)\cdot(f(n)/e^{c\cdot f(n)})$ for some constant c, and substituting n_{OPT} and $\rho(n)$ in (1), we get

$$\mathbf{Pr}[\textsc{Succ}] \geq \Omega\left(\left(1/e^{c\cdot f(n)}\right)^{\frac{\log n}{f(n)}}\right) = \Omega\left(e^{-c\cdot\log n}\right) = \Omega(n^{-c}).$$

For any $\varepsilon > 0$, if we run Algorithm RandomReduce $\log(1/\varepsilon)\cdot n^c$ times and choose the best solution, the probability of success is amplified to at least

$$1 - \left(1 - \frac{1}{n^c}\right)^{\log(1/\varepsilon)\cdot n^c} \;=\; 1 - (1/e)^{\log 1/\varepsilon} \;=\; 1 - \varepsilon\;,$$

and the approximation ratio is at least $\rho(n) = (n/\log n)\cdot(f(n)/e^{c\cdot f(n)})$. Note that $\rho(n) \leq n/\log n$. Indeed, this is achieved with equality (up to a constant factor) when $f(n) = 1$, i.e., $n_{\text{OPT}} = \log n$. Otherwise, when $f(n) > 1$ we have $f(n)/e^{c\cdot f(n)} < 1$, and when $f(n) < 1$ we have $f(n)/e^{c\cdot f(n)} \leq f(n) < 1$. $\square$

3.2 A Better Algorithm for High-Degree Graphs

In this section we provide another randomized algorithm for graphs with high minimum degree, and make a connection with known combinatorial results concerning subgraphs with given minimum degree.

Proposition 1. *For any $d \geq 3$ and any function $f(n)$ such that $8 \log n \leq f(n) \leq n$, there exists a polynomial-time randomized approximation algorithm for the d-GIRTH problem in the class of graphs with minimum degree at least $d \cdot f(n)$, with approximation ratio $\frac{16n \cdot \log n}{d \cdot f(n)}$.*

Proof: Let G be a graph with minimum degree at least $d \cdot f(n)$. The algorithm is very simple: it chooses each vertex independently with probability $8 \log n / f(n)$. Let H be the graph induced in G by the vertices chosen by the algorithm, and let $n_0 = |V(H)|$. The variable n_0 is the sum of n independent Boolean random variables $B_1, \ldots, B_n$, and its expected value is $8n \cdot \log n / f(n)$. Therefore, applying the Chernoff-Hoeffding bound we get

$$\mathbf{Pr}\left[n_0 > 2 \cdot \frac{8n \cdot \log n}{f(n)}\right] \leq e^{-\frac{8n \cdot \log n}{3f(n)}} = n^{-\frac{8n}{3f(n)}} \leq \frac{1}{n^2},$$

so $|V(H)| \leq 16n \cdot \log n / f(n)$ with high probability. Let us now argue about the degree of a vertex $v \in V(H)$. Since $\deg_G(v) \geq d \cdot f(n)$, the expected value of $\deg_H(v)$ is at least $d \cdot f(n) \cdot 8 \log n / f(n) = 8d \cdot \log n$. Applying the Chernoff-Hoeffding bound again we get

$$\mathbf{Pr}\left[\deg_H(v) < d\right] \leq \mathbf{Pr}\left[\deg_H(v) < \frac{8d \cdot \log n}{2}\right] \leq e^{\frac{-8d \cdot \log n}{8}} = \frac{1}{n^d} < \frac{1}{n^2},$$

relying on the fact that $d \geq 3$. Therefore, using the union bound we get

$$\mathbf{Pr}\left[\delta(H) < d\right] \leq |V(H)| \cdot \frac{1}{n^2} \leq \frac{n}{n^2} = \frac{1}{n}.$$

Hence H is a valid solution to the d-GIRTH problem with probability at least $1 - 1/n$. Finally, the approximation ratio follows from the fact that any solution has at least $d + 1$ vertices. $\square$

The proof of Proposition 1 implies that for a graph G with $\delta(G) \geq d \cdot k$, one can find w.h.p. a subgraph H with $\delta(H) \geq d$ and $|V(H)| \leq \frac{16n \cdot \log n}{k}$. This can be thought of as a weaker but *constructive* version of the following result about subgraphs with minimum degree at least d.

Theorem 4 (Erdős *et al.* [14]). *Let $d \geq 2$ and $k > 1$ be given. Every n-vertex graph G with at least $\lceil d \cdot k \cdot n \rceil$ edges has a subgraph H with $\delta(H) \geq d$ and $|V(H)| \leq \lceil n/k \rceil$.*

Note that the combinatorial result of Theorem 4 is stronger than the one that follows from the proof of Proposition 1 in two ways. First, the required premise concerns only the number of edges of G, instead of its minimum degree. Second, the size of the subgraph obtained in the proof of Proposition 1 is greater by a factor $16 \log n$ than the one given by Theorem 4. However, the proof of Theorem 4 is *non-constructive* (at least, in polynomial time).

3.3 A Deterministic Algorithm for Low-Degree Graphs

Note that the hardness results of Section 2 hold even if the degrees of the input graph are either d or $d+1$. In the following proposition we provide a *deterministic* approximation algorithm in the more general case where the input graph has degree bounded by an appropriate function of the size of the input graph.

Proposition 2. *For any integer $d \geq 3$, there exists a deterministic polynomial-time $\mathcal{O}\left(\frac{n \cdot \log\log n}{\log n}\right)$-approximation algorithm for the d-*GIRTH *problem in the class of n-vertex graphs with maximum degree $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$.*

Proof: The algorithm consists in an exhaustive search in order to try building a solution to the d-GIRTH problem of appropriate size, by trying all possibilities of obtaining such a solution starting from each vertex of G. The algorithm stops at some point in order to keep the running time polynomial, and if no solution has been found so far, it outputs the whole graph.

Let the maximum degree of G be at most b, and let k be the maximum size of a solution that the algorithm can find (both b and k will be specified later). The algorithm proceeds as follows. Starting from a given vertex, it tries to build a feasible solution $S \subseteq V(G)$ by adding vertices one by one to S. At a given moment, if some vertex $v \in S$ has strictly less than d neighbors in S, it chooses a neighbor of v in $V(G) \setminus S$, and adds it to S. This process continues until either all vertices of G have degree at least d in $G[S]$, or $|S| = k$.

For an integer ℓ, with $0 \leq \ell \leq k$, we define $f(\ell)$ to be the *remaining* running time of the algorithm assuming that all possible solutions of size at most ℓ have been already considered. Therefore, by definition $f(0)$ is the overall running time of the algorithm. Once ℓ vertices have been already chosen into S, with $\ell \geq 1$, the number of choices for a neighbor of each vertex of S in $V(G) \setminus S$ is at most b, so it holds

$$f(\ell) \ \leq \ \ell \cdot b \cdot f(\ell+1). \tag{2}$$

On the other hand, at the beginning the algorithm chooses an arbitrary vertex of G, so

$$f(0) \ \leq \ n \cdot f(1). \tag{3}$$

Starting from Equation (3), using Equation (2) recursively, and assuming that the algorithm stops when $|S| = k$, we get

$$f(0) \ \leq \ n \cdot b^k \cdot k! \cdot f(k+1). \tag{4}$$

As the running time must be polynomial in n, from Equation (4) it follows that $n \cdot b^k \cdot k! = n^{\mathcal{O}(1)}$, that is, $b^k \cdot k! \leq n^c$ for some positive integer c. In other words,

$$k \cdot \log b + k \cdot \log k \ \leq \ c \cdot \log n. \tag{5}$$

If we further impose that $b \leq k$, a sufficient condition for Equation (5) to be satisfied is that $k = \mathcal{O}\left(\frac{\log n}{\log\log n}\right)$. That is, for graphs with maximum degree $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$, the above procedure constitutes a polynomial-time $\mathcal{O}\left(\frac{n\cdot\log\log n}{\log n}\right)$-approximation algorithm. □

4 Planar Graphs

In this section we focus on the case where the input graph is restricted to be a planar graph. The first important observation is that as any planar graph has a vertex of degree at most 5 [10], in a planar graph there exist feasible solutions to the d-GIRTH problem only for $d \leq 5$. Note that the hardness results of [2], and therefore also those of Section 2, do not apply to planar graphs, as the constructed graphs are highly nonplanar. In Section 4.1 we prove that the d-GIRTH problem is NP-hard in planar graphs for $d \in \{3, 4, 5\}$. We then discuss approximation algorithms in Section 4.2, and present a subexponential exact algorithm in Section 4.3.

4.1 Hardness Results

Theorem 5. *For $d \in \{3, 4, 5\}$, the d-GIRTH problem is NP-hard in planar graphs with maximum degree at most $3d$.*

Proof: The reduction is from MINIMUM VERTEX COVER (VC for short) in planar graphs with maximum degree at most 3, which is known to be NP-hard [16]. Note that MINIMUM VERTEX COVER admits a PTAS in planar graphs [5]. For the sake of presentation, we first state the reduction for $d = 3$, and then we show how to modify the gadgets for $d \in \{4, 5\}$.

Let H be a planar graph with $\Delta(H) \leq 3$, an instance of the MINIMUM VERTEX COVER problem, which we can assume to be connected (see Fig. 1(a) for an example). To build G from H, we first replace each edge $e = \{u, v\} \in E(H)$ by the gadget depicted in Fig. 1(b), containing u, v, and 12 new vertices. Among these vertices, let S_e^u (resp. S_e^v) be the three vertices adjacent to u (resp. v). Vertices $x_e, y_e, z_e, x'_e, y'_e, z'_e$ are colored *white* and vertices in S_e^u and S_e^v are colored *black* (see Fig. 1(b)). Now, for each face f of H (including the external one) consisting of edges $e_0, e_2, \ldots, e_{k-1}$ such that e_i is incident to e_{i+1} for $i = 0, \ldots, k-1$ (indices taken modulo k), we add the following edges. Assume without loss of generality that all the white vertices corresponding to f are of the form x_e, y_e, z_e. For $i = 0, \ldots, k-1$, add an edge between vertex z_{e_i} and vertex $y_{e_{i+1}}$, the indices being taken modulo k. These edges between white vertices corresponding to different edges are called *face edges*. This completes the construction of G, which is illustrated in Fig. 1(c). Note that G is a planar graph with maximum degree at most 9.

Consider a solution $S \subseteq V(G)$ to the 3-GIRTH problem in G. By construction, S cannot contain just black vertices or vertices corresponding to vertices in H, so at least one white vertex belongs to S, say x_e for an edge $e = \{u, v\} \in E(H)$.

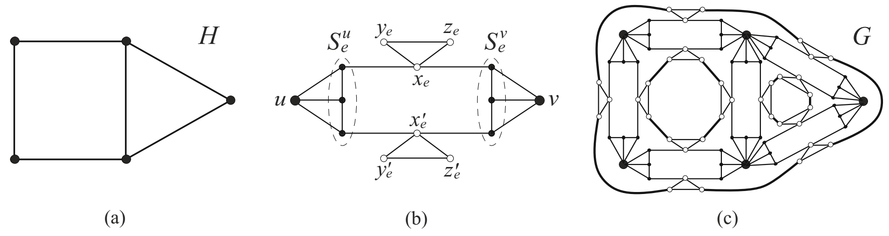


Fig. 1. Reduction in the proof of Theorem 5 for $d = 3$: (a) Instance H of Minimum Vertex Cover. (b) Gadget corresponding to an edge $e = \{u, v\} \in E(H)$. (c) Instance G of the 3-girth problem built from H.

Due to the degree constraints and since x_e is adjacent to only 2 white vertices, vertex x_e forces either all the vertices in S_e^u or all the vertices in S_e^v to belong to S, which in turn also forces vertex x'_e to be in S. Due to the face edges, once a vertex x_e is in S, so are all the white vertices corresponding to edges in the same face as edge $e \in E(H)$. Thus, white vertices inductively force *all* $6 \cdot |E(H)|$ white vertices to be in S. Now recall that for a pair of white vertices x_e, x'_e to have degree at least 3 in $G[S]$, either all the vertices in S_e^u or in S_e^v must belong to S, so any optimal solution to the 3-girth in G contains exactly $3 \cdot |E(H)|$ black vertices. Finally, note that if the vertices in S_e^u (resp. S_e^v) belong to S, they force vertex u (resp. v) to belong to S. As for each edge $e = \{u, v\} \in E(H)$, either u or v must belong to S, we conclude that there is a bijection between vertex covers of H and feasible solutions to the 3-girth in G.

The above discussion implies that $g_3(G) = 9 \cdot |E(H)| + \text{OPT}_{\text{VC}}(H)$. (For instance, in the example of Fig. 1, any optimal vertex cover of H of size 3 defines an optimal subgraph of G on 57 vertices with minimum degree at least 3.) As Minimum Vertex Cover is NP-hard, so is the 3-girth problem.

For $d \in \{4, 5\}$, one just needs to modify the gadget of Fig. 1(b) with respect to the reduction for $d = 3$; the corresponding gadgets for $d \in \{4, 5\}$ can be found in [22]. □

4.2 Approximation Algorithms

In this section we derive the existence of a deterministic approximation algorithm with ratio $n/f(n)$ and running time $2^{\mathcal{O}(f(n))} \cdot n$ when the input graph G is restricted to be planar. In particular, this provides an alternative to the polynomial-time $n/\log n$-approximation algorithm provided in [2] for minor-free graphs.

Proposition 3. *For any function $f(n) \leq n$, there exists a deterministic approximation algorithm for the d-girth problem in n-vertex planar graphs with approximation ratio $n/f(n)$ and running time $2^{\mathcal{O}(f(n))} \cdot n$.*

Proof: It is well-known that the number of non-isomorphic planar graphs on k vertices is $2^{\mathcal{O}(k)}$ [24]. In addition, this set of graphs can be generated in time proportional to its size using the algorithm in [19]. Given a planar graph G on n vertices and an arbitrary function $f(n) \leq n$, we generate all non-isomorphic planar graphs on $f(n)$ vertices in time $2^{\mathcal{O}(f(n))}$. We remove from the list the graphs with minimum degree less than d. Then, for each graph H in this list, we test whether G contains a subgraph isomorphic to H using the recent results for planar subgraph isomorphism [11], in time $2^{\mathcal{O}(f(n))} \cdot n$. If none of these subgraphs is found, we output the whole graph G. This procedure clearly yields a $(n/f(n))$-approximation algorithm running in time $2^{\mathcal{O}(f(n))} \cdot n$. □

4.3 Exact Algorithms

In this section we show that the problem can be solved in *subexponential* time when the input is restricted to planar graphs. Recall that there exist valid solutions only for $d \leq 5$.

Theorem 6. *For any $d \geq 3$, the d-*GIRTH *problem can be solved exactly in planar graphs in time $2^{\mathcal{O}(\sqrt{n} \cdot \log n)}$.*

Proof: We use the classical divide-and-conquer approach. By the planar separator Theorem [20], every n-vertex planar graph has a vertex separator W of size at most $c\sqrt{n}$, for some small constant $c \leq 2\sqrt{2}$, such that after the removal of W the graph is partitioned into two disconnected subgraphs on vertex sets Z_1 and Z_2, each of cardinality at most $2n/3$. In addition, such separator W can be found in time $\mathcal{O}(n)$.

Our algorithm proceeds recursively as follows. The separator given by [20] divides the problem into two or more smaller problems. In order to build a feasible solution to the d-GIRTH problem, exhaustively check every subset of vertices in the separator, and then for each subset check every possible set of up to d neighbors in Z_1 and Z_2. As usual, the subproblems are solved by applying the method recursively, and the solutions to the subproblems are combined to give a solution to the original problem in the input graph G.

To analyze the running time, define the function $f(\ell)$ to be the time required by the algorithm to process a graph on ℓ vertices. Hence $f(n)$ is the overall running time of our algorithm, and we can assume that $f(1) = 1$. As the number of choices for a subset of the separator W is $2^{|W|} \leq 2^{c\sqrt{n}}$, and the number of choices in Z_1 or Z_2 for a set of at most d neighbors of each vertex in the separator is at most $\binom{|Z_i|}{d}$, $i = 1, 2$,

$$\begin{aligned} f(n) &\leq 2^{|W|} \cdot \left(\binom{|Z_1|}{d} \cdot \binom{|Z_2|}{d} \right)^{|W|} \cdot 2 \cdot f(2n/3) \leq 2^{c\sqrt{n}} \cdot (2n/3)^{2dc\sqrt{n}} \cdot 2 \cdot f(2n/3) \\ &= 2^{c_1 \sqrt{n} \cdot \log n} \cdot f(2n/3) \leq 2^{c_1 \left(\sqrt{n} \cdot \log n + \sqrt{2n/3} \cdot \log(2n/3) + \dots \right)} \cdot f(1) \\ &\leq 2^{c_1 \log n \cdot \left(\sqrt{n} + \sqrt{2n/3} + \sqrt{4n/9} + \dots \right)} \leq 2^{c_1 \sqrt{n} \cdot \log n \cdot \left(\sum_{i=0}^{\infty} (2/3)^i \right)} = 2^{c_2 \sqrt{n} \cdot \log n}, \end{aligned}$$

where c_1, c_2 are suitable constants defined by the above equations, depending on c and d. □

5 Concluding Remarks

This article studies the problem of approximating the d-girth of a graph, the order of a smallest subgraph with minimum degree at least d, for a fixed integer $d \geq 3$, and makes first steps towards understanding the computational complexity of this apparently hard problem. We now summarize our results and present several possible lines for further research.

We proved that for any $d \geq 3$ and $\varepsilon > 0$, there is no polynomial-time algorithm for the d-GIRTH problem with approximation ratio $2^{\mathcal{O}(\log^{1-\varepsilon} n)}$ in graphs with degrees d or $d+1$ unless NP $\subseteq$ DTIME $\left(2^{\mathcal{O}(\log^{1/\varepsilon} n)}\right)$. We suspect that the problem is even harder than this. In the spirit of [17] for the LONGEST PATH problem, we present the following conjecture.

Conjecture 1. For every fixed $d \geq 3$, there is no polynomial-time approximation algorithm for the d-GIRTH problem with ratio $n^{1-\delta}$, for some constant $\delta > 0$ unless P = NP.

We provided the first approximation algorithms for the d-GIRTH problem in general graphs. Specifically, we presented a randomized algorithm with approximation ratio $n/\log n$ in any graph, another randomized algorithm with better performance in high-degree graphs, and a deterministic algorithm for low-degree graphs. These approximation ratios could hopefully be improved, although it looks like a challenging task. Our latter two approaches for high- and low-degree graphs complement each other in some sense, so it would be interesting to try to combine them into a better algorithm.

We also studied the case where the input graph is planar. We proved that the d-GIRTH problem is NP-hard in planar graphs for $d \in \{3, 4, 5\}$, presented a deterministic approximation algorithm (with the same ratio as the algorithm for general graphs) based on a recent result for subgraph isomorphism [11], and showed that the problem can be solved exactly in subexponential time. This latter result may provide some clue to the possible existence of a PTAS in planar graphs, which remains wide open. So far, none of the many approaches to obtaining a PTAS in planar graphs seems to fit the d-GIRTH problem.

We point out that some of our results do not strongly use specific properties of the d-GIRTH problem, and can be applied to the class of problems of the form "minimum subgraph with property P", in particular to the problem of finding a *d-regular* subgraph of minimum size. It would be interesting also to study other variants of the problem, like minimizing the number of edges of a subgraph with minimum degree at least d (in planar graphs, this version is equivalent to the original one, modulo a constant factor), or considering the vertex-weighted version.

Finally, the reader is also referred to several nice combinatorial conjectures of Erdős *et al.* [13,14] about the existence of small subgraphs with given minimum degree.

References

1. Addario-Berry, L., Dalal, K., Reed, B.A.: Degree constrained subgraphs. Discrete Applied Mathematics 156(7), 1168–1174 (2008)
2. Amini, O., Peleg, D., Pérennes, S., Sau, I., Saurabh, S.: Degree-Constrained Subgraph Problems: Hardness and Approximation Results. In: Bampis, E., Skutella, M. (eds.) ALGO/WAOA 2008. LNCS, vol. 5426, pp. 29–42. Springer, Heidelberg (2009), `www.cs.technion.ac.il/~ignasi/Pubs/APP+10.pdf`
3. Amini, O., Pérennes, S., Sau, I.: Hardness and Approximation of Traffic Grooming. Theoretical Computer Science 410(38-40), 3751–3760 (2009)
4. Amini, O., Sau, I., Saurabh, S.: Parameterized Complexity of the Smallest Degree-Constrained Subgraph Problem. In: Grohe, M., Niedermeier, R. (eds.) IWPEC 2008. LNCS, vol. 5018, pp. 13–29. Springer, Heidelberg (2008), `www.cs.technion.ac.il/~ignasi/Pubs/ASS10.pdf`
5. Baker, B.S.: Approximation Algorithms for NP-Complete Problems on Planar Graphs. Journal of the ACM 41(1), 153–180 (1994)
6. Bermond, J.-C., Peyrat, C.: Induced Subgraphs of the Power of a Cycle. SIAM Journal on Discrete Mathematics 2(4), 452–455 (1989)
7. Bollobás, B., Brightwell, G.: Long Cycles in Graphs with no Subgraphs of Minimal Degree 3. Discrete Mathematics 75, 47–53 (1989)
8. Cheriyan, J., Thurimella, R.: Approximating Minimum-Size k-Connected Spanning Subgraphs via Matching (extended abstract). In: Proc. of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 292–301 (1996)
9. Chow, T., Lin, P.J.: The ring grooming problem. Networks 44, 194–202 (2004)
10. Diestel, R.: Graph Theory, vol. 173. Springer, Heidelberg (2005)
11. Dorn, F.: Planar Subgraph Isomorphism Revisited. In: Proc. of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS). LIPIcs, vol. 5, pp. 263–274 (2010)
12. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
13. Erdős, P., Faudree, R.J., Gyárfás, A., Schelp, R.H.: Cycles in Graphs Without Proper Subgraphs of Minimum Degree 3. Ars Combinatorica 25(B), 195–201 (1988)
14. Erdős, P., Faudree, R.J., Rousseau, C.C., Schelp, R.H.: Subgraphs of Minimal Degree k. Discrete Mathematics 85(1), 53–58 (1990)
15. Gabow, H.N.: An Efficient Reduction Technique for Degree-Constrained Subgraph and Bidirected Network Flow Problems. In: Proc. of the 15th Annual ACM Symposium on Theory of Computing (STOC), pp. 448–456 (1983)
16. Garey, M.R., Johnson, D.S.: The Rectilinear Steiner Tree Problem is NP-Complete. SIAM Journal on Applied Mathematics 32(4), 826–834 (1977)
17. Karger, D., Motwani, R., Ramkumar, G.: On Approximating the Longest Path in a Graph. Algorithmica 18(1), 82–98 (1997)
18. Kézdy, A.: Studies in Connectivity. PhD thesis, Univ. of Illinois, Urbana-Champaign (1991)
19. Li, Z., Nakano, S.-I.: Efficient generation of plane triangulations without repetitions. In: Yu, Y., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 433–443. Springer, Heidelberg (2001)
20. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM Journal on Applied Mathematics 36, 177–189 (1979)
21. Nutov, Z.: Approximating Directed Weighted-Degree Constrained Networks. In: Goel, A., Jansen, K., Rolim, J.D.P., Rubinfeld, R. (eds.) APPROX and RANDOM 2008. LNCS, vol. 5171, pp. 219–232. Springer, Heidelberg (2008)

22. Peleg, D., Sau, I., Shalom, M.: On approximating the d-girth of a graph (manuscript), www.lirmm.fr/~sau/Pubs/d-girth.pdf
23. Safari, M., Salavatipour, M.R.: A Constant Factor Approximation for Minimum λ-Edge-Connected k-Subgraph with Metric Costs. In: Goel, A., Jansen, K., Rolim, J.D.P., Rubinfeld, R. (eds.) APPROX and RANDOM 2008. LNCS, vol. 5171, pp. 233–246. Springer, Heidelberg (2008)
24. Tutte, W.T.: A census of planar triangulations. Canadian Journal of Mathematics 14, 21–38 (1962)

SScAC: Towards a Framework for Small-Scale Software Architectures Comparison

Petr Praus, Slávka Jaroměřská, and Tomáš Černý

Department of Computer Science and Engineering, Czech Technical University, Charles square 13, 121 35 Prague 2, CZ
{prauspet,jaromsla,tomas.cerny}@fel.cvut.cz

Abstract. We present a framework for small-scale software architecture comparison (SScAC). Although a considerable chunk of software architectures are developed in small teams, not much related work exists on this topic. The proposed framework introduces a method to formalize these comparisons and aims to be simple enough to be used in small-scale projects at the same time. Still we believe it is of sufficient complexity to support comparisons that take into account different aspects of solved problem. The main purpose of the framework is to ease certain architectural choices by giving the designer a reasoned recommendation based on previously specified requirements on system's qualities. It can also help validate the suitability of chosen design patterns. We show the practical use of the framework on case study solving architectural decision for Key Word In Context.

Keywords: Software reuse; Software quality metrics; Software integration.

1 Introduction

To architect a software application the developer has to face an important and often not easy decisions. What architectural style should he apply? Which design patterns to use? The importance of these choices is obvious. Selection of appropriate invariants can positively affect many aspects as well as cause negative impacts on system performance, behaviour and maintenance. Therefore all architectural decisions have to respect requirements on such a system. In this paper we introduce a comparison framework that helps to evaluate architectural choices for development of a small-sized systems. When the developer hesitates which architectural style [7] to select SScAC framework provides him with a reasoned recommendation based on previously specified requirements on system's qualities. To demonstrate the practical use of framework we present a case study solving architectural decision for Key Word In Kontext (KWIC) problem (see [12]) introduced by D. Parnas in 1972 that is still used as a classroom example.

2 Related Work

An existing method for analyzing software architectures called SAAM [9], grounds its analysis in concrete domain – Human-Computer Interaction and user interface domain. The method does not provide any metrics for predictive evaluation. Instead they come up with example-based method for performing qualitative evaluation. To evaluate the suitability of architecture in terms of modifiability they prepare an example task to test the desired property (we call this a goal), e.g. modifiability. Then it is measured how difficult (how many changes in program) it is to achieve the desired task. This approach is acceptable as long as we assess only one property but when we want to compare more achitectures from broader point of view it becames cumbersome and possibly time consuming.

Successor of SAAM, ATAM [10] provides a very comprehensive tool for analysing software architectures including management of teams performing such an analysis. In contrast, SScAC framework targets small projects with at most a half dozen developers, who either woudn't use any architecture evaluation method at all or would drown in ATAM's or similar large-scale method's complexity.

An approach presented by [2,3] characterizes quality attributes (we call them goals) and captures architectural patterns that are used to achieve these attributes. They call these patterns attribute primitives. But applying a pattern to achieve a certain quality can negatively affect other qualities[1]. Therefore all qualities have to be assessed at once and all their dependencies must be considered which is non-trivial task. In our framework we break down these complex relationships into smaller units (so called "properties") that are mapped to all affected goals. Therefore the user does not have to take into account the dependencies. [3] mentions following example:

> An interpreter makes easier the creation of new functions or modification of existing function. Macro recording and execution is an example of an interpreter. This is an excellent mechanism for achieving modifiability at run-time, but it also has a strong negative influence on performance.

If a designer's primary goal was modifiability an interpreter would be a suitable choice. Otherwise if one of your primary goals was also performance it would get a poor grade which might negatively affect rating of the whole design. Direct cause of such result would be usage of inappropriate attribute primitive. When not using SScAC framework such an implication might escape your attention.

A model for predicting a set of most suitable architectural styles that are worth considering when implementing a certain system is proposed by [4]. The selection is based on use cases that need to be met by a system configuration. The fitness of particular architectural style is represented by the Conformance Confidence Index (CCI). The main drawback of this metric is that it does not reflect required quality properties. However, this approach can be effectively combined with our framework. First, designer chooses the prospective architectural styles based on CCI. Then he compares them using our framework with respect to their qualities.

Another related approach to software quality metrics is [1] that is based on goals and questions answerable in a measurable way.

[1] For example, one might sacrifice reliability for the sake of the performance.

3 Proposal of SScAC Framework

In our comparison framework we target smaller scale applications. User has several architecture designs and this framework's purpose is to assist the developer in choosing the most suitable one for a specified problem. Successful framework usage presumes equivalent input conditions for all designs. This means they must have the same functionality achieved by very similar algorithms. User should be aware the framework only provides comparison of properties resulting from architectural features. For example we do not consider business point of view[2] or security[3]. *In addition all grades are comparable only in context of one framework use because user grading is expected to be relative to other compared architectures.* The solution to the given problem can be usually achieved by meeting variously important goals. These goals are chosen by the designer and framework outputs grades representing how well compared designs fit the specified problem. For describing common architectural goals our framework defines default set of properties. These are subsequently graded by the developer based on how well they are supported by each design. To justify a grade the developer might use techniques outlined in [3]. For example by discovering design patterns associated with certain properties.

To reach a certain goal we need a combination of relevant properties but because they are not equivalent each one has a different weight. Framework also provides predefined goals mapped to properties with respective weights. One property can be used to achieve several goals – every time with a different weight. In addition user has the freedom to extend framework with new goals (perhaps even with new properties) but for common use cases he should not have the need.

3.1 SScAC Use Guideline

Typical application of this framework consists of these steps:

- Prepare implementation drafts of your project based on various architectures but keep functionality and algorithms as similar as possible.
- Choose goals you are interested in (such as performance) and give them relative weights. Their summation should be always 1, the more important the goal for you the higher the weight.
- Rate all the goal-relevant properties relatively for each design (0-4, the better the support of the property in the design the higher the grade).
- Multiply grades of properties by their weight and sum the results within each goal. This gives you grades for each goal you set. Repeat this step for each design.
- Approximate comparison can be calculated by multiplying grade of each goal with it's weight (defined in step 1) and summing the results for each design. This gives you rough idea how much each design is suitable for your purpose.

[2] Developer skill set, available tools etc.

[3] A security is in fact a set of functionalities. And because these are required to be the same, there is nothing else left to solve.

3.2 Default Goals

Since the most small-scale projects target similar architectural traits we define a default set of goals that should cover common framework usage. In this section we describe them. Numbering in brackets corresponds to Table 4 in appendix, where the concrete weight values are defined.

The aim of the *performance* goal is to predict overall speed of the system. *Parallelization* (1) property represents how often the parallelization concept occures in the given implementation draft. Bottlenecks can be either hardware or logical. Typical hardware bottleneck would be hard-drive access or throttled network connection. Logical bottlenecks are overloaded design components such as trackers in BitTorrent[5] system. The *no bottlenecks* (2) property therefore represents absence of these. *Synchronization* (3) property represents the support and simplicity for synchronization mechanisms such as locks. If you are forced to maintain additional mechanisms to achieve synchronization, this property should get a low grade. *Robustness penalty* (4) expresses how clumsy and performance impeding the implementation of security mechanisms is. Among these could be repeated checks, backups etc. *Stable data format* (5) gets a high rating when for multiple reuse of the same data across the system the data's format does not need to be changed.

Multi-user responsiveness goal differs from performance in two properties – instead of general parallelization property we define *multiple request parallelization* (6). The criteria is whether the system uses mechanisms supporting stable response time for the majority of users. Another difference is the definition of *synchronization* (7). This time it expresses the ability of the system to cope with simultaneous access to shared resources. The aim is to minimalize average waiting time for the operation results.

Memory effiency is intended to be used in cases of limited hardware resources or manipulation with large amounts of data. It encompasses *no repeated data copying* (8), *single copy of data* (9)(e.g. Flyweight [6]) and effective *object reuse* (10). For example instead of repeated creating of threads for each request one might use a thread pool.

Modifiability consists of two equal goals (both weights 0.5)– *changeability*, which enables easy strategy changes (replacement), and *expanding* functionality concentrated on adding it. In general we are interested in how much work is needed to make a change or to reuse a component. We measure this as number of components but you should always take into account their size and complexity.

To evaluate *changeability* goal you should imagine a sample situation in which you need to replace some functionality. You should count *how many original components needs to be changed* (11) and *how many new components you have to add* (12). The lower the numbers the higher the grades. *Expanding functionality* uses the same properties as changeability although the model situation is different – this time the user wants to add new functionality.

We understand *reusability* in two ways. First reusability in scope of the system, second across multiple systems. They both capture *how many components the reused one is dependant on* (13) (more precisely how many compoments must be added to reuse it – this definition is crucial in terms of one system), *how much the reused one must be changed* (14) and *how many additional components must be added to integrate it* (15) (e.g. adapter [6]). But all of them should be evaluated in the right context – reuse in single or multiple systems.

Testability represents how easy it is to test that all components in the system are functioning properly. Easily testable system should not need unnecessarily *complex mock objects* (MO not required – 16) and it should be possible to *test small units of code* (17). *Reliability* is quite complex goal. It encompasses *bug prevention* (18), handling user errors, robustness features and *maintainability* (19). Proper bug prevention can be achieved by good testability. The architectural part of maintanability is strongly supported by good modifiability. Concerning user errors the system should provide *easy error handling* (20) and *achievable recovery mechanisms* (21). Reliable system should be robust. Therefore we need to determine how much effort is necessary to *capture state* (22) of the system and to create a *failover* (23).

4 Case Study: Keyword in Context (KWIC)

This case study provides some typical examples of comparison between different software designs (Pipe and Filter, Event-based and Aspect). We will be interested in performance, modifiability and realibility goals further described in subsection 4.2. As a demonstration example we chose widely known, simple and yet illustrative enough problem of Keyword in Context thoroughly investigated by Parnas in his 1972 article [12]. We extended standard set of architectural designs of KWIC compared in [7] by David Garlan and Mary Shaw with the new one based on aspect oriented programming (AOP) proposed by Gregor Kiczales and collective [11] in 1997. It should serve as a representative of new programming paradigms solving one of the Achilles heels of object-oriented programming – the cross-cutting issue[11].

4.1 About KWIC

Parnas proposed the following solution in [12]:

> Any line may be "circularly shifted" by repeatedly removing the first word and appending it at the end of the line. The KWIC index system outputs a listing of all circular shifts of all lines in alphabetical order.

Circular shifter and alphabetizer create a simple index with huge data redundancy. We decided to update this idea and implement index using hash map. Whereas hash map key is indexed keyword and its value is a list of occurrences of the particular keyword in the source text. Based on this idea our solution consists of the following actions:

1. line reading
2. splitting lines into tokens

3. indexing and storing tokens (logically in parallel)
4. creating contexts using index and stored tokens
5. outputting contexts

It is essential that all compared implementations maintain the same functionality level.

Pipe-and-Filter KWIC. Based on above *actions* we designed following filters: Input, Tokenizer, Index, Context finder and Output. They correspond to actions 1-5. Exception to this is storing of tokens which ideally should be accomplished in parallel with their indexation. Tokens must be passed through Index to Context finder. This aspect forces designers into counterintuitive design decisions and could lower performance. Hence it will be avoided in other implementations. To leverage strong advantage of P&F style, each filter runs in it's own thread.

Aspect KWIC. One of the biggest architectural advantages of P&F is component isolation and resulting suitability for parallelization. Aspect KWIC is designed to preserve these features. Components correspond to filters and they run in their own thread. On the other hand multiple data formatting and passing is troublesome invariant of P&F style. This cross-cutting concern is easily solved by AOP. Each logical communication relationship between components is generalized as an abstract aspect. This aspect has its own *provider* and a list of *consumers*. A "finished" pointcut is defined to set the state of the provider. E.g. for Tokenizer as a provider, there is TokenizingAspect (supplying Index and ContextFinder) whose "finished" pointcut watches for termination of Tokenizer thread. To supply the consumers the aspect defines "action" pointcut responsible for watching for product creation events of providing component. These products are subsequently passed to each consumer. In Fig. 1 you can see the logical relationship/data-flow structure among components. The black lines represent data flow (facilitated by their associated aspects) from providers (dots) to their consumers (arrows).

Event-Based KWIC. The logical design of the third implementation is similar to aspect KWIC but it applies more traditional technology – implicit invocation. We used standard Java components: class Observer and interface Observable. KWIC components (as defined previously: Input, Tokenizer...) use Observer pattern to pass events between them. Similarly to aspect KWIC, Tokenizer is observed by two components – Index and ContextFinder. The only exception to implicit invocation is passing of index data structure from Index component to ContextFinder. Unlike in two aforementioned implementations

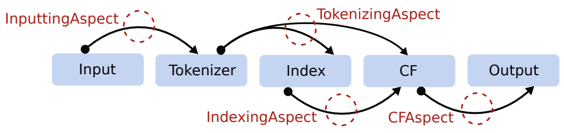


Fig. 1. Aspect KWIC schema

threads are not part of the design. Related figures of Event and P&F can be found in [7] due to limited space.

4.2 Application of SScAC Framework

We demonstrate the use of the framework on two test examples. In the first one (Table 1), we are interested in a system that has better-than-average performance but is moderately modifiable at the same time. Therefore we are seeking two properties: performance and modifiability with respective weights 0.6 and 0.4 based on previously outlined needs. Modifiablity is composite goal and consists of changeability and expandability. In the second example we wish to achieve highest possible reliability (Table 2).

Grades in the tables are based on following reasoning. Parallelization is supported by Aspect and P&F because they use threads in contrast to Event. They got high grades even though their context finder and output components have to wait until the previous threads are finished but important is the comparison to Event. When comparing to a different approach (e.g. stream) they could get lower grades (*Paralellization*). Neither implementation contains significant hardware or logical bottlenecks (*No bottlenecks*). P&F and Aspect implementations require synchronized queues to support multi-threaded working model which is not essential for KWIC core functionality (*Synchronization*). Aspect uses relatively costly mechanisms to bind pointcuts and advices together (see [8]). Moreover it contains additional components (aspects) to provide interaction between core components. Event uses relatively cheaper mechanisms (standard Java API observer pattern implementation) to achieve the same. P&F uses only explicit method invocation (*No robustness penalty*). In P&F currently processed data

Table 1. Performance and modifiability related property evaluation - test 1

Goal-Subgoal/Property	Weight	P&F	Aspect	Event
PERFORMANCE				
Parallelization	0.25	3	3	1
No bottlenecks	0.25	4	4	4
Synchronization	0.2	2	2	4
No robustness penalty	0.2	4	2	3
Stable data format	0.1	1	4	4
MODIFIABLITY - CHANGEABILITY				
How many components to change	0.7	4	4	3
How many components to add	0.3	4	3	4
MODIFIABLITY - EXPANDABILITY				
How many components to change	0.7	4	4	3
How many components to add	0.3	4	3	4

Table 2. Reliability related property evaluation - test 2

Goal/Property	W1[a]	W2[b]	P&F	Aspect	Event
Reliability					
Bug prevention (testability)	0.25	0.4	2.5	4	3
Capturable state of the system	0.125	0			
Easy failover	0.125	0			
Easy error handling	0.125	0.2	0	4	1
Achievable rec. mechanisms	0.125	0			
Maintenance (modifiability)	0.25	0.4	4	3.7	3.3
Testability					
No compl. mock objects	0.5		4	4	3
Small unit testing possible	0.5		1	4	3

[a] Default weights
[b] Recalculated weights

are parsed by each filter and subsequently passed to next filter in its standard data format (*Stable data format*).

How many components do we need to change and add in order to change functionality: In P&F we can simply replace a filter – 0 changed, 1 added; In Aspect we need to replace a component and a related aspect – 0 changed, 2 added; In Event we need to replace one component. Because the components do not have common interface we are forced to make larger (compared to others) changes in initialization (e.g. statically binding observers and observables) – 1 changed, 1 added (*Modifiability - changeability*). Expandability is in this case almost identical to changeability, therefore we will not elaborate this further.

Some of the properties of the reliability goal are irrelevant to our case study. Therefore we need to discard them and recalculate weights of the remaining properties while keeping the same ratio between them (default and recalculated weights can be seen in Table 2). Maintainance in our framework is put for architectural purposes on a par with modifiability. The modifiability grades have been already explained (*Maintainence*). Bug prevention can be seen as testability which is a separate goal consisting of two properties: no complicated objects and small unit testing possible. To test the correct functionality of P&F filters we just need to prepare an input string and its corresponding output string. Testing the *Aspect* implementation is strongly supported by AOP mechanisms, because testing in principle is cross-cutting concern. In *Event* implementation we are forced to simulate events from observed components and create observers to evaluate tested component's behavior (*No complicated mock objects*). The *Filter* implementation is the smallest unit we can test (no testable public methods) in *P&F*. Aforementioned AOP mechanisms allow us to violate encapsulation and thoroughly test each component. *Event* provides standard testing possibilities but compared to *Aspect* we are limited by the necessity to obey encapsulation

Table 3. Implementation evaluation results

Goal	Weight	P&F	Aspect	Event
Performance	0.6	3.05	2.95	3.05
Modifiability	0.4	4	3.7	3.3
$\sum weight \cdot goal$		3.43	3.25	3.15
Reliability	1	2.6	3.88	2.72

(*Small unit testing possible*). Feasible handling of errors in *P&F* without violating its architectural style's contract is practically impossible. Error can either propagate through all the remaining filters or to implement a listener to handle exceptions occuring in filters. Again AOP principles allow us to easily implement feasible and centralized error handling. Error handling in *Event* implementation would require an extra layer of listeners and instead of throwing exceptions, an error listeners is notified, while distinction them common listeners must exist. Alternatively a large try-catch block in the initialization component could exist (*Easy error handling*). Table 3 shows results of both tests. In the first one we look for an implementation that would be performant and modifiable. Performance was slightly more important. Overall grades gave us a hint that *P&F* should be the most suitable option (score 3.43/4). On the other hand *Event* has shown up as the worst choice (score 3.15/4), but as you can see the difference is relatively small (just 7% of the total score). The aim of the second test was to find the most reliable implementation. *Aspect* proved to be by far the best choice (score 3.88/4, 29% better than *Event*).

4.3 Evaluation

We implement all three KWIC designs. To validate design *performance* scores suggested by the SSCaC, we perform the following test. We measure the time it takes to a given implementation to process a set of queries (A-H) on *War and Peace* by Leo Tolstoy (572,633 words). Words included in queries are based on simple frequency analysis. Results were normalized by removing minimum, maximum and creating median from sample of 11 independent measurements.

Query *A* represents 15 most frequent words. Following queries then progressively remove the most frequent word to discover the relation between the runtime of application and the portion of query keywords in the entire text. See axis in Fig. 2. Last three queries contain real-scenario combination of words.

When processing a short query, according to graph in Fig. 2 *Event* turns out to be the quickest solution even though it does not implement multiple threads. In contrast *P&F* is the best at longer queries. *Aspect* is slightly slower in all cases. This could be possibly attributed to performance penalty introduced by AspectJ which is further discussed in [8]. Table 3 (SScAC results) shows *P&F* and *Event* are on a par from the performance point of view. This corresponds with results of aforementioned measurement. An interesting observation can be made when

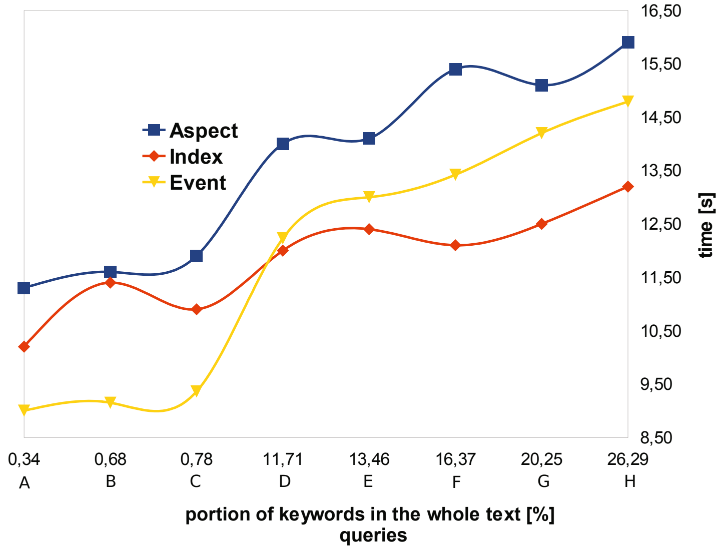


Fig. 2. Relationship between the run-time of application and the portion of query keywords in the whole text

looking at *Event* and *Aspect* graph behavior. They both tend to rise roughly at the same values and keep the similar angle across all the queries. This could be attributed to similar fundamental design characteristics – in both cases an event occurs, someone notices and reacts by invoking an associated code. (*Event*: state changed – observable notifies – observer reacts, *Aspect*: method invoked – poincut – advice). Though similar in design characteristics they diametrically differ in the way they are used by a developer. This is mainly reflected in testability and easy error handling. It should be noted that graph can be misleading. The spread of measured values is quite large, performance can be affected by other influences and the difference between implementations is more or less constant.

5 Conclusion

We have presented SScAC framework for general small-scale architecture comparison. It was partialy inspired by [3] and [4]. Nonnegligible chunk of the software development is done in small teams. However, none of the existing work provides metrics for predictive evaluation on a small-scale level. SScAC framework is designed to fill this gap and allows small teams to assess their design in a more formal manner and taking into account multiple variously important design goals at the same time. For our framework to be usable we had to avoid steep learning curve and complexity associated with some of the more advanced methods. On a negative side user of our framework has to develop at least draft implementations of compared designs and maintain the same functionality across all of them. Unfortunately it is sometimes hard to discern functionality from design aspects. Future work on framework might include elaboration of new goals

Table 4. Goals and properties overview

Goal/Property	Weight	Comments on weights
PERFORMANCE		
(1)	0.25	Parallelization concepts *directly increase* performance.
(2)	0.25	Bottlenecks establish *upper limits* on speed.
(3)	0.2	Effective synch. mech. *support* higher performance.
(4)	0.2	Robustness is *consuming* computing time.
(5)	0.1	Changing data format is *consuming* computing time.
RESPONSIVENESS		
(6)	0.3	*Increase* in performance, *stable* request-handling time.
(2)	0.25	Bottlenecks establish *upper limits* on speed.
(7)	0.25	*Stable* request-handling time for multiple users.
(4)	0.1	Robustness is *consuming* computing time.
(5)	0.1	Changing data format is *consuming* computing time.
MEM. EFFICIENCY		
(8)	0.2	It *improves memory fragmentation.*
(9)	0.5	It *strongly supports* effective usage of memmory.
(10)	0.3	Poor object reuse could *signify higher fragmentation.*
EXPANDABILITY		
(11)	0.7	We *never* want to change the existing components.
(12)	0.3	The *less work*, the better.
CHANGEABILITY		
(11)	0.7	We *never* want to change the existing components.
(12)	0.3	The *less work*, the better.
REUS. IN THE SYS.		
(15)	0.4	It might be *an indicator of bad design.*
(13)	0.2	*Sometimes necessary* but the *less* the better.
(14)	0.4	We *never* want to change the existing components.
REUS. ACROSS SYS.		
(15)	0.3	*It might be necessary* but the *less* the better.
(13)	0.3	*Sometimes necessary* but the *less* the better.
(14)	0.4	We *never* want to change the existing components.
TESTABILITY		
(16)	0.5	*Unnecessary* additional work.
(17)	0.5	It leads to simpler tests and *lower* mistake possibility.
RELIABILITY		
(18)	0.25	The less bugs the higher reliability.
(22)	0.125	Easy backup.
(23)	0.125	Easy failover enables redundancy.
(20)	0.125	If too difficult some errors might not be caught.
(21)	0.125	The easier the recovery the shorter the outage.
(19)	0.25	Lower probability of a bug while maintaining.

such as *clean & readable design* or *scalability* in respect to medium-scale systems and further extension of goal tests. In terms of further academic research the proposed framework could contribute to work by L. Bass et al. [2,3] and elucidate the amount of influence certain pattterns have on architectural properties.

References

1. Basili, V.R., Caldiera, G., Rombach, H.D.: The goal question metric approach. Encyclopedia of software engineering 1, 528–532 (1994)
2. Bass, L., Klein, M., Bachmann, F.: Quality attribute design primitives. Software Engineering Institute Technical Report CMU/SEI-2000-TN-017 (2000)
3. Bass, L., Klein, M., Bachmann, F.: Quality attribute design primitives and the attribute driven design method. Software Product-Family Engineering (2001)
4. Bhattacharya, S., Perry, D.E.: Predicting Architectural Styles from Component Specification. Automation, PTD (2005)
5. Cohen, B.: Incentives build robustness in bittorrent. In: Workshop on Economics of Peer-to-Peer Systems, Berkeley, USA (May 2003)
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc., Boston (1995)
7. Garlan, D., Shaw, M.: An introduction to software architecture. Advances in Software Engineering and Knowledge Engineering 1, 1–40 (1993)
8. Hilsdale, E., Hugunin, J.: Advice weaving in AspectJ. In: Proceedings of the 3rd International Conference on Aspect-Oriented Software Development, pp. 26–35. ACM, New York (2004)
9. Kazman, R., Bass, L., Webb, M., Abowd, G.: SAAM: A method for analyzing the properties of software architectures. In: Proceedings of the 16th International Conference on Software Engineering. IEEE Computer Society Press, Los Alamitos (1994)
10. Kazman, R., Klein, M., Clements, P.: ATAM: Method for architecture evaluation. CMU/SEI (2000)
11. Kiczales, G.J., Lamping, J.O., Lopes, C.V., Hugunin, J.J., Hilsdale, E.A., Boyapati, C.: Aspect-oriented programming (October 15,2002); US Patent 6,467,086
12. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Communications of the ACM 15(12), 1058 (1972)

Folk Theorems on the Correspondence between State-Based and Event-Based Systems

Michel A. Reniers[1] and Tim A.C. Willemse[2]

[1] Department of Mechanical Engineering, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands
[2] Department of Computer Science, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands

Abstract. Kripke Structures and Labelled Transition Systems are the two most prominent semantic models used in concurrency theory. Both models are commonly believed to be equi-expressive. One can find many ad-hoc embeddings of one of these models into the other. We build upon the seminal work of De Nicola and Vaandrager that firmly established the correspondence between stuttering equivalence in Kripke Structures and divergence-sensitive branching bisimulation in Labelled Transition Systems. We show that their embeddings can also be used for a range of other equivalences of interest, such as strong bisimilarity, simulation equivalence, and trace equivalence. Furthermore, we extend the results by De Nicola and Vaandrager by showing that there are additional translations that allow one to use minimisation techniques in one semantic domain to obtain minimal representatives in the other semantic domain for these equivalences.

1 Introduction

Concurrency theory, and process theory in general, deal with the analysis and specification of behaviours of reactive systems, *i.e.*, systems that continuously interact with their environment. Over the course of the past decades, a rich variety of formal languages have been proposed for modelling such systems effectively. At the level of the semantics, however, consensus seems to have been reached over the models used to represent these behaviours. Two of the most pervasive models are the state-based model generally referred to as *Kripke Structures* and the event-based model known as *Labelled Transition Systems*, henceforth referred to as KS and LTS.

The common consensus is that both the KS and LTS models are on equal footing. This is supported by several embeddings of one model into the other that have been studied in the past, see below for a brief overview of the relevant literature. As far as we have been able to trace, in all cases embeddings of both semantic models were considered modulo a single behavioural equivalence. For instance, in their seminal work [8], De Nicola and Vaandrager showed that there are embeddings in both directions showing that stuttering equivalence [1] in KS coincides with divergence-sensitive branching bisimulation [4] in LTS. The embeddings, however, look a bit awkward from the viewpoint of concrete equivalence relations.

On the basis of these results, one cannot arrive at the conclusion that the embeddings also work for a larger set of equivalences. For instance, it is very easy to come up with a mapping that reflects and preserves branching-time equivalences while breaking linear-time equivalences, by exposing observations of branching through the encodings. Note that it is equally easy to construct encodings that break branching-time equivalences while reflecting and preserving some linear-time equivalences, *e.g.*, by including some form of determinisation in the embeddings.

Our contributions are as follows. Using the KS-LTS embeddings lts and ks of De Nicola and Vaandrager in [7], in Section 3 we formally establish the following relations under these embeddings:

1. bisimilarity in KS reflects and preserves bisimilarity in LTS;
2. similarity in KS reflects and preserves similarity in LTS;
3. trace equivalence in KS reflects and preserves completed trace equivalence in LTS.

These results add to the credibility that indeed both worlds are on equal footing, and it may well be that the embeddings ks and lts are in fact canonical.

As already noted in [7], there is no immediate correspondence between the embeddings lts and ks. For instance, one cannot move between KS and LTS and back again by composing lts and ks. We mend this situation by introducing two additional translations, *viz.*, lts^{-1} and ks^{-1}, that can be used to this end. Moreover, we show that combining these with the original embeddings enables one to minimise with respect to an equivalence in KS by minimising the embedded artefact in LTS (and *vice versa*).

From a practical point of view, our contributions allow one to smoothly move between both semantic models using a single set of translations. This reduces the need for implementing dedicated software in one setting when one can take advantage of state-of-the-art machinery available in the other setting.

Related Work. In their seminal paper (see [8]) on logics for branching bisimilarity, De Nicola and Vaandrager established, among others, a firm correspondence between the divergence-sensitive branching bisimilarity of [4], and stuttering equivalence [1]. Their results spawned an interest in the relation between temporal logics in the LTS and the KS setting, see *e.g.* [6,7]. The latter both contain the embeddings that we use in this paper, differing slightly from the ones proposed in [8], which in turn were in part inspired by the (unpublished) embedding by Emerson and Lei [2]. The tight correspondence between stuttering equivalence and branching bisimilarity that was exposed, led Groote and Vaandrager to define algorithms for deciding said equivalences in [5]. Their algorithms (and their correctness proofs), however, are stated directly in terms of the appropriate setting, and do not appear to use the embeddings lts and ks (but they might have acted as a source of inspiration).

Apart from the few documented cases listed above, many ad-hoc embeddings are known to work for equivalences that are not sensitive to abstraction. For instance, one can model the state labelling in a Kripke Structure by means of labelled self-loops, or directly on the edges to the next states, thereby exposing the same information. Such embeddings, however, fail for equivalences that are sensitive to abstraction, such as stuttering equivalence, which roughly compresses sequences of states labelled with the same state information.

Outline. In Section 2, we formally introduce the computational models KS and LTS, along with the embeddings ks and lts. The latter are proved to preserve and reflect the additional three pairs of equivalence relations stated above. In Section 4, we introduce the inverses ks^{-1} and lts^{-1}, and we show that these can be combined with ks and lts, respectively, to obtain our minimisation results. We finish with a brief summary of our contributions and an outlook to some interesting open issues. Details of the proofs for our results can be found in [9].

2 Preliminaries

Central in both models of computation that we consider, *i.e.*, KS and LTS, are the notions of *states* and *transitions*. While the KS model emphasises the information contained *in* such states, the LTS model emphasises the *state changes* through some action modelling a real-life event. Let us first recall both models of computation.

Definition 1. *A* Kripke Structure *is a structure* $\langle\, S, AP, \rightarrow, L \,\rangle$*, where*

- S *is a set of states;*
- AP *is a set of atomic propositions;*
- $\rightarrow \subseteq S \times S$ *is a total transition relation,* i.e.*, for all* $s \in S$*, there exists* $t \in S$*, such that* $(s,t) \in \rightarrow$*;*
- $L : S \rightarrow 2^{AP}$ *is a state labelling.*

By convention, we write $s \rightarrow t$ whenever $(s,t) \in \rightarrow$.

Remark 1. The transition relation in the KS model is traditionally required to be total. Our results do not depend on the requirement of totality, but we choose to enforce totality in favour of a smoother presentation and more concise definitions. Without totality, slightly more complicated treatments of the notions of paths and traces (see also Section 3.4) are needed.

With the above restriction in mind, we define the LTS model with a similar restriction imposed on it.

Definition 2 (Labelled Transition System). *A structure* $\langle\, S, Act, \rightarrow \,\rangle$ *is an* LTS*, where:*

- S *is a set of states;*
- Act *is a set of actions;*
- $\rightarrow \subseteq S \times (Act \cup \{\tau\}) \times S$ *is a total transition relation,* i.e.*, for all* $s \in S$*, there are* $a \in Act$*,* $t \in S$*, such that* $(s,a,t) \in \rightarrow$*.*

In lieu of the convention for KS, we write $s \xrightarrow{a} t$ whenever $(s,a,t) \in \rightarrow$.

Note that in the setting of the LTS model, a special constant τ is assumed outside the alphabet of the set of actions *Act* of any concrete transition system. This constant is used to represent so-called silent steps in the transition system, modelling events that are unobservable to any witness of the system.

In [7], De Nicola and Vaandrager considered embeddings called lts and ks, which allowed one to move from KS models to LTS models, and, *vice versa*, from LTS models to KS models. We repeat these embeddings below, starting with the embedding from KS into LTS.

Definition 3. *The embedding* $\mathsf{lts} : \mathsf{KS} \to \mathsf{LTS}$ *is defined as* $\mathsf{lts}(K) = \langle S', Act, \to \rangle$ *for arbitrary Kripke Structures* $K = \langle S, AP, \to, L \rangle$*, where:*

- $S' = S \cup \{\bar{s} \mid s \in S\}$*, where it is assumed that* $\bar{s} \notin S$ *for all* $s \in S$*;*
- $Act = 2^{AP} \cup \{\bot\}$*;*
- $\to$ *is the smallest relation satisfying:*

$$\frac{}{s \xrightarrow{\bot} \bar{s}} \qquad \frac{s \to t \qquad L(s) = L(t)}{s \xrightarrow{\tau} t}$$

$$\frac{}{\bar{s} \xrightarrow{L(s)} s} \qquad \frac{s \to t \qquad L(s) \neq L(t)}{s \xrightarrow{L(t)} t}$$

The fresh symbol $\bot$ is used to signal a forthcoming encoding of the state information of the Kripke Structure. Encoding the state information by means of a self-loop $s \xrightarrow{L(s)} s$ introduces problems in preserving and reflecting equivalences that are sensitive to abstraction.

Definition 4. *The embedding* $\mathsf{ks} : \mathsf{LTS} \to \mathsf{KS}$ *is formally defined as* $\mathsf{ks}(T) = \langle S', AP, \to, L \rangle$ *for Labelled Transition System* $T = \langle S, Act, \to \rangle$*, where:*

- $S' = S \cup \{(s, a, t) \in \to \mid a \neq \tau\}$*;*
- $AP = Act \cup \{\bot\}$*, where* $\bot \notin Act$*;*
- $\to$ *is the least relation satisfying:*

$$\frac{}{s \to (s, a, t)} \qquad \frac{}{(s, a, t) \to t} \qquad \frac{s \xrightarrow{\tau} t}{s \to t}$$

- $L(s) = \{\bot\}$ *for* $s \in S$*, and* $L((s, a, t)) = \{a\}$*.*

In ks, the fresh symbol $\bot$ is used to label the states from the Labelled Transition System. The τ-transitions are treated differently from concrete actions, allowing one to reflect equivalences that abstract from sequences of τ-transitions.

Observe that, as already stated in [7], due to the artefacts introduced by the embeddings, moving from LTS to KS and back again yields transition systems incomparable to the original ones. Consequently, in LTS, one cannot take advantage of tools for minimising in the setting of KS, and *vice versa*. We defer further discussions on this matter to Section 4.

3 Preservations and Reflections of Equivalences under lts and ks

The embeddings lts and ks have already been shown to preserve and reflect stuttering equivalence [1] and divergence-sensitive branching bisimulation [4] by De Nicola and Vaandrager. In this section, we introduce three additional pairs of equivalences and show that these are also preserved by the embeddings lts and ks. Our choice for these four equivalences is motivated largely by the limited set of equivalences available in the KS model (contrary to the LTS model, which offers a very fine-grained lattice of equivalence relations, see [3]).

Remark 2. For reasons of brevity, throughout this paper we define equivalence relations on states within a single LTS (resp. KS) rather than equivalence relations between different models in LTS (resp. KS). Note that this does not incur a loss in generality, as it is easy to define the latter in terms of the former.

3.1 Similarity

Both KS and LTS have well-developed theories revolving around the notion of similarity. We first formally define both notions.

Definition 5. *Let $K = \langle S, AP, \rightarrow, L \rangle$ be a Kripke Structure. A relation $B \subseteq S \times S$ is a* simulation relation *iff for every $s, s' \in S$ satisfying $(s, s') \in B$:*

- *$L(s) = L(s')$;*
- *for all $t \in S$, if $s \rightarrow t$, then $s' \rightarrow t'$ for some $t' \in S$ such that $(t, t') \in B$.*

For states $s, s' \in S$, we say s is simulated by s' if there is a simulation relation B, such that $(s, s') \in B$. States $s, s' \in S$ are said to be similar*, denoted $K \models s \simeq s'$ iff there are simulation relations B and B', such that $(s, s') \in B$ and $(s', s) \in B'$.*

Remark 3. It should be noted that when lifting our notion of similarity to an equivalence relation between different models in KS, the first requirement is sometimes stated as $L(s) = L'(s') \cap AP$, where L' is the state labelling of the second KS model, and AP is the set of atomic propositions of the first KS model. In this case, some form of abstraction is included, and care should be taken to deal with this properly when lifting all our results to such a setting.

Definition 6. *Let $T = \langle S, Act, \rightarrow \rangle$ be a Labelled Transition System. A relation $B \subseteq S \times S$ is a* simulation relation *iff for every $s, s' \in S$ satisfying $(s, s') \in B$:*

- *for all $t \in S$ and $a \in Act \cup \{\tau\}$, if $s \xrightarrow{a} t$, then $s' \xrightarrow{a} t'$ for some $t' \in S'$ such that $(t, t') \in B$.*

State $s \in S$ is said to be simulated by state $s' \in S$ if there is a simulation relation B, such that $(s, s') \in B$. States $s, s' \in S$ are similar*, denoted $T \models s \simeq s'$ iff there are simulation relations B and B', such that $(s, s') \in B$ and $(s', s) \in B'$.*

The theorems below state that indeed, embedding lts preserves and reflects KS-similarity through LTS-similarity (see Theorem 1), and *vice versa*, embedding ks preserves and reflects LTS-similarity through KS-similarity (Theorem 2).

Theorem 1. *Let $K = \langle S, AP, \to, L \rangle$ be an arbitrary Kripke Structure. Then, for all $s, s' \in S$, we have $K \models s \simeq s'$ iff $\mathsf{lts}(K) \models s \simeq s'$.*

Theorem 2. *Let $T = \langle S, Act, \to \rangle$ be a Labelled Transition System. Then for all $s, s' \in S$, we have $T \models s \simeq s'$ iff $\mathsf{ks}(T) \models s \simeq s'$.*

3.2 Bisimilarity

A stronger notion of equivalence that is rooted in the same concepts as similarity, is *bisimilarity*. Again, bisimilarity has been defined in both KS and LTS, and we here show that both definitions agree through the embeddings lts and ks.

Definition 7. *Let $K = \langle S, AP, \to, L \rangle$ be a Kripke Structure. States $s, s' \in S$ are said to be* bisimilar*, denoted $K \models s \underline{\leftrightarrow} s'$ iff there is a* symmetric simulation relation *B, such that $(s, s') \in B$.*

Similarly, we define bisimilarity in the setting of LTS as follows:

Definition 8. *Let $T = \langle S, Act, \to \rangle$ be a Labelled Transition System. States $s, s' \in S$ are* bisimilar*, written $T \models s \underline{\leftrightarrow} s'$ iff there is a* symmetric simulation relation *B, such that $(s, s') \in B$.*

Theorem 3. *Let $K = \langle S, AP, \to, L \rangle$ be a Kripke Structure. Then for all $s, s' \in S$, we have $K \models s \underline{\leftrightarrow} s'$ iff $\mathsf{lts}(K) \models s \underline{\leftrightarrow} s'$.*

Theorem 4. *Let $T = \langle S, Act, \to \rangle$ be a Labelled Transition System. For all $s, s' \in S$, we have $T \models s \underline{\leftrightarrow} s'$ iff $\mathsf{ks}(T) \models s \underline{\leftrightarrow} s'$.*

3.3 Stuttering Equivalence – Divergence-Sensitive Branching Bisimilarity

In this section, we merely repeat the definitions for stuttering equivalence and divergence-sensitive branching bisimilarity. In Section 4, we come back to these equivalence relations and state several new results for these.

The following definition for stuttering equivalence is taken from [8], where it is shown to coincide with the original definition by Brown, Clarke and Grumberg [1]. We prefer the former phrasing because of its coinductive nature.

Definition 9. *Let $K = \langle S, AP, \to, L \rangle$ be a Kripke Structure. A* symmetric *relation $B \subseteq S \times S$ is a* divergence-blind stuttering equivalence *iff for all $(s, s') \in B$:*

- *$L(s) = L(s')$;*
- *for all $t \in S$, if $s \to t$, then there exist $s'_0, \ldots, s'_n \in S$, such that $s' = s'_0$ and $(t, s'_n) \in B$, and for all $i < n$, $s'_i \to s'_{i+1}$ and $(s, s'_i) \in B$.*

Definition 10. *Let $K = \langle S, AP, \to, L \rangle$ be a Kripke Structure. Let the Kripke Structure $K_d = \langle S_d, AP_d, \to_d, L_d \rangle$ be defined as follows:*

- *$S_d = S \cup \{s_d\}$ for some fresh state $s_d \notin S$;*
- *$AP_d = AP \cup \{d\}$ for some fresh proposition $d \notin AP$;*

– $\rightarrow_d = \rightarrow \cup \{(s, s_d) \mid s$ *is on an infinite path of states labelled* $L(s)$, *or* $s = s_d\}$;
– *for all* $s \in S$, $L_d(s) = L(s)$, *and* $L_d(s_d) = \{d\}$.

States $s, s' \in S$ *are said to be* stuttering equivalent, *notation:* $K \models s \approx_{\mathrm{s}} s'$ *iff there is a divergence-blind stuttering equivalence relation* B *on* S_d *of* K_d, *such that* $(s, s') \in B$.

The origins of divergence-sensitive branching bisimilarity can be traced back to [4]. In [10], Van Glabbeek *et al* demonstrate that various incomparable phrasings of the divergence property all coincide with the original definition. For our purposes the following formulation is most suitable.

Definition 11. *Let* $T = \langle S, Act, \rightarrow \rangle$ *be a Labelled Transition System. A symmetric relation* $B \subseteq S \times S'$ *is a* divergence-sensitive branching simulation relation *iff for all* $(s, s') \in B$:

– *if there is an infinite sequence of states* $s_0\ s_1\ s_2 \cdots$ *such that* $s = s_0$ *and* $s_i \xrightarrow{\tau} s_{i+1}$ *for all* i, *then there exist a mapping* $\sigma : \mathbb{N} \rightarrow \mathbb{N}$, *and an infinite sequence of states* $s'_0\ s'_1\ s'_2 \cdots$ *such that* $s' = s'_0$, $s'_k \xrightarrow{\tau} s'_{k+1}$ *and* $(s_{\sigma(k)}, s'_k) \in B$ *for all* $k \in \mathbb{N}$;
– *for all* $t \in S$ *and* $a \in Act \cup \{\tau\}$, *if* $s \xrightarrow{a} t$, *then* $a = \tau$ *and* $(t, s') \in B$, *or* $s' \xrightarrow{\tau^*} s^* \xrightarrow{a} t'$ *for some* $s^*, t' \in S$ *such that* $(s, s^*) \in B$ *and* $(t, t') \in B$.

States $s, s' \in S$ *are divergence-sensitive branching bisimilar, notation* $s \underline{\leftrightarrow}_{\mathrm{dsb}} s'$ *iff there is a symmetric divergence-sensitive branching simulation relation* B, *such that* $(s, s') \in B$.

3.4 Trace Equivalence – Completed Trace Equivalence

Trace equivalence and completed trace equivalence are the only linear-time equivalence relations that we consider in this paper. In defining these equivalence relations, we require some auxiliary notions, basically defining what a *computation* is in our respective models of computation.

Definition 12. *Let* $K = \langle S, AP, \rightarrow, L \rangle$ *be a Kripke Structure. A* path *starting in state* $s \in S$ *is an infinite sequence* $s_0\ s_1\ \ldots$, *such that* $s_i \rightarrow s_{i+1}$ *for all* i, *and* $s = s_0$. *The set of all paths starting in* s *is denoted* $\mathsf{Paths}(s)$.

Basically, a path formalises how a single computation evolves in time. Actually, it is the information contained in the states that are visited along such a computation that is often of interest, as it shows how the state information evolves in time. This is exactly captured by the notion of a *trace*.

Definition 13. *Let* $K = \langle S, AP, \rightarrow, L \rangle$ *be a Kripke Structure. Let* $\pi = s_0\ s_1\ \ldots$ *be a path starting in* s_0. *The* trace *of* π, *denoted* $\mathsf{Trace}(\pi)$, *is the infinite sequence* $L(s_0)\ L(s_1)\ \ldots$. *For a set of paths* Π, *we set*

$$\mathsf{Traces}(\Pi) = \{\mathsf{Trace}(\pi) \mid \pi \in \Pi\}$$

States $s, s' \in S$ *are* trace equivalent, *denoted* $K \models s \simeq_{\mathrm{t}} s'$, *if* $\mathsf{Traces}(\mathsf{Paths}(s)) = \mathsf{Traces}(\mathsf{Paths}(s'))$.

Remark 4. In the presence of non-totality of the transition relation of a Kripke Structure, it no longer suffices to consider only the infinite paths as the basis for defining trace equivalence. Instead, *maximal* paths are considered, which in addition to the infinite paths, also contains paths made up of sequences of states that end in a sink-state, *i.e.*, a state without outgoing edges.

For models in LTS, we define similar-spirited concepts; for the origins of the definition, we refer to Van Glabbeek's lattice of equivalences [3].

Definition 14. *Let $T = \langle\, S, Act, \rightarrow \,\rangle$ be a Labelled Transition System. A* run *starting in a state $s \in S$ is an infinite, alternating sequence of states and actions $s_0\ a_0\ s_1\ a_1\ \ldots$ satisfying $s_i \xrightarrow{a_i} s_{i+1}$ for all i, and $s = s_0$. The set of all runs starting in s_0 is denoted $\mathsf{Runs}(s_0)$.*

Definition 15. *Let $T = \langle\, S, Act, \rightarrow \,\rangle$ be a Labelled Transition System. The* trace *of a run $\rho = s_0\ a_0\ s_1\ a_1\ \ldots$, denoted $\mathsf{Trace}(\rho)$, is the infinite sequence $a_0\ a_1\ \cdots$. For a set of runs R, we define*

$$\mathsf{Traces}(R) = \{\mathsf{Trace}(\rho) \mid \rho \in R\}$$

States $s, s' \in S$ are completed trace equivalent*, denoted by $T \models s \simeq_{\mathrm{t}} s'$ iff $\mathsf{Traces}(\mathsf{Runs}(s)) = \mathsf{Traces}(\mathsf{Runs}(s'))$.*

Theorem 5. *Let $K = \langle\, S, AP, \rightarrow, L \,\rangle$ be a Kripke Structure. For all $s, s' \in S$, we have $K \models s \simeq_{\mathrm{t}} s'$ iff $\mathsf{lts}(K) \models s \simeq_{\mathrm{t}} s'$.*

In a similar vein, we obtain that completed trace equivalence in LTS is preserved and reflected by trace equivalence in KS.

Theorem 6. *Let $T = \langle\, S, Act, \rightarrow \,\rangle$ be a Labelled Transition System. Let $s, s' \in S$ be arbitrary states. We have $T \models s \simeq_{\mathrm{t}} s'$ iff $\mathsf{ks}(T) \models s \simeq_{\mathrm{t}} s'$.*

4 Minimisations in LTS and KS

As we concluded in Section 2, the mappings lts and ks cannot be used to freely move to and fro the computational models. Instead, we introduce two additional mappings, *viz.*, lts^{-1} and ks^{-1} that act as inverses to lts and ks, respectively, and we show that these can be used to come to our results for minimisation. Here, we focus on the computationally most attractive equivalences, *viz.*, *bisimilarity* and *stuttering equivalence*.

Let $\sim\ \in \{\underline{\leftrightarrow}, \approx_{\mathrm{s}}\}$ and $\leftrightarrow\ \in \{\underline{\leftrightarrow}, \underline{\leftrightarrow}_{\mathrm{dsb}}\}$ be arbitrary equivalence relations on KS and LTS, respectively. For a given model K in KS, its *quotient* with respect to $\sim$ is denoted $K_{/\sim}$. Similarly, for a given model T in LTS, its *quotient* with respect to $\leftrightarrow$ is denoted $T_{/\leftrightarrow}$. We assume unique functions $\sim$-$\min_{\mathsf{KS}}$ for KS, and $\leftrightarrow$-$\min_{\mathsf{LTS}}$ for LTS that uniquely determine transition systems that are isomorphic to the quotient. If, from the equivalence relation $\sim$, the setting is clear, we drop the subscripts and write $\sim$-min instead.

4.1 Minimisation in KS via Minimisation in LTS

We first characterise a subset of models of LTS for which we can define our inverse lts^{-1} of lts.

Definition 16. *Let $T = \langle\, S, Act, \rightarrow \,\rangle$ be a Labelled Transition System. Then T is reversible iff*

1. *$Act = 2^{AP} \cup \{\perp\}$, for some set AP;*
2. *for all $s, s' \in S$ and $a \in Act \cup \{\tau\}$, if $s \xrightarrow{a} s'$, then $s' \xrightarrow{\perp}$;*
3. *for all $s, s', s'' \in S$ such that $s \xrightarrow{\perp} s'$ and $s \xrightarrow{\perp} s''$, we require that $s' \xrightarrow{a}$ and $s'' \xrightarrow{a'}$ implies $a = a'$ for all actions $a, a' \in Act$.*

Note that any embedding $\mathsf{lts}(K)$ of a Kripke Structure K is a reversible Labelled Transition System. Reversibility is preserved by the quotients for $\underline{\leftrightarrow}$ and $\underline{\leftrightarrow}_{\text{dsb}}$, as stated by the following proposition.

Proposition 1. *Let T be an arbitrary reversible Labelled Transition System. Then $T_{/\leftrightarrow}$, for $\leftrightarrow\ \in \{\underline{\leftrightarrow}, \underline{\leftrightarrow}_{\text{dsb}}\}$, is reversible.*

The embedding lts introduces a fresh, *a priori* known action label $\perp$. We treat this constant differently from all other actions in our reverse embedding.

Definition 17. *Let $T = \langle\, S, Act, \rightarrow \,\rangle$ be a reversible Labelled Transition System. We define the Kripke Structure $\mathsf{lts}^{-1}(T)$ as the structure $\langle\, S', AP, \rightarrow, L\,\rangle$, where:*

- *$S' = \{s \in S \mid s \xrightarrow{\perp}\}$;*
- *AP is such that $Act = 2^{AP} \cup \{\perp\}$;*
- *$\rightarrow$ is the least relation satisfying the single rule:*

$$\frac{s \xrightarrow{a} s' \qquad a \neq \perp \qquad s \xrightarrow{\perp}}{s \rightarrow s'}$$

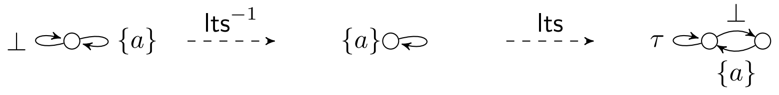
- *$L(s) = a$ for the unique a such that $s \xrightarrow{\perp} s' \xrightarrow{a}$.*

The following proposition establishes that lts^{-1} is the inverse of embedding lts.

Proposition 2. *We have $\mathsf{lts}^{-1} \circ \mathsf{lts} = \mathsf{Id}$.*

Note that reversibility of a Labelled Transition System T is too weak to obtain $(\mathsf{lts} \circ \mathsf{lts}^{-1})(T) = T$, as the following example illustrates:

Example 1. Consider the reversible Labelled Transition System left below.



Following $\mathsf{lts} \circ \mathsf{lts}^{-1}$ leads to the LTS at the right, via the middle Kripke Structure. It is clear that the latter is not isomorphic to the original LTS. $\square$

Lemma 1. *We have* $\underline{\leftrightarrow}\text{-}min_{\mathsf{LTS}} \circ \mathsf{lts} \circ \underline{\leftrightarrow}\text{-}min_{\mathsf{KS}} = \mathsf{lts} \circ \underline{\leftrightarrow}\text{-}min_{\mathsf{KS}}$.

Lemma 2. *We have* $\underline{\leftrightarrow}_{\mathrm{dsb}}\text{-}min_{\mathsf{LTS}} \circ \mathsf{lts} \circ \approx_{\mathrm{s}}\text{-}min_{\mathsf{KS}} = \mathsf{lts} \circ \approx_{\mathrm{s}}\text{-}min_{\mathsf{KS}}$.

Before we present the main theorems concerning the minimisations in KS through minimisations in LTS, we first show that it suffices to prove such results for Kripke Structures that are already minimal; see the lemma below.

Lemma 3. *Let* $\sim\ \in \{\underline{\leftrightarrow}, \approx_{\mathrm{s}}\}$ *and* $\leftrightarrow\ \in \{\underline{\leftrightarrow}, \underline{\leftrightarrow}_{\mathrm{dsb}}\}$ *such that* lts *preserves and reflects* $\sim$ *through* $\leftrightarrow$. *Then*

$$\sim\text{-}min = \mathsf{lts}^{-1} \circ \leftrightarrow\text{-}min \circ \mathsf{lts} \circ \sim\text{-}min$$

implies

$$\sim\text{-}min = \mathsf{lts}^{-1} \circ \leftrightarrow\text{-}min \circ \mathsf{lts}$$

Proof. Assume that we have

$$\sim\text{-min} = \mathsf{lts}^{-1} \circ \leftrightarrow\text{-min} \circ \mathsf{lts} \circ \sim\text{-min} \qquad (*)$$

By definition of $\sim$-min, we find $\forall K : \sim\text{-min}(K) \sim K$. Since, by assumption, lts preserves and reflects $\sim$ through $\leftrightarrow$, we derive $\forall K : \mathsf{lts}(K) \leftrightarrow \mathsf{lts}(\sim\text{-min}(K))$. By definition of $\leftrightarrow$-min, this means that we have:

$$\leftrightarrow\text{-min} \circ \mathsf{lts} \;=\; \leftrightarrow\text{-min} \circ \mathsf{lts} \circ \sim\text{-min}$$

As lts^{-1} is functional, and $\leftrightarrow$-min preserves reversibility, we immediately obtain:

$$\mathsf{lts}^{-1} \circ \leftrightarrow\text{-min} \circ \mathsf{lts} = \mathsf{lts}^{-1} \circ \leftrightarrow\text{-min} \circ \mathsf{lts} \circ \sim\text{-min} \qquad (**)$$

The desired conclusion then follows by combining * and **. □

We finally state the two main theorems in this section.

Theorem 7. *We have* $\underline{\leftrightarrow}\text{-}min_{\mathsf{KS}} = \mathsf{lts}^{-1} \circ \underline{\leftrightarrow}\text{-}min_{\mathsf{LTS}} \circ \mathsf{lts}$.

Proof. Lemma 1 guarantees

$$\underline{\leftrightarrow}\text{-min}_{\mathsf{LTS}} \circ \mathsf{lts} \circ \underline{\leftrightarrow}\text{-min}_{\mathsf{KS}} = \mathsf{lts} \circ \underline{\leftrightarrow}\text{-min}_{\mathsf{KS}}$$

Functionality of lts^{-1}, combined with Proposition 1, we find:

$$\mathsf{lts}^{-1} \circ \underline{\leftrightarrow}\text{-min}_{\mathsf{LTS}} \circ \mathsf{lts} \circ \underline{\leftrightarrow}\text{-min}_{\mathsf{KS}} = \mathsf{lts}^{-1} \circ \mathsf{lts} \circ \underline{\leftrightarrow}\text{-min}_{\mathsf{KS}}$$

By Lemma 3, we then have our desired conclusion:

$$\underline{\leftrightarrow}\text{-min}_{\mathsf{KS}} = \mathsf{lts}^{-1} \circ \underline{\leftrightarrow}\text{-min}_{\mathsf{LTS}} \circ \mathsf{lts}$$

□

Theorem 8. *We have* $\approx_{\mathrm{s}}\text{-}min_{\mathsf{KS}} = \mathsf{lts}^{-1} \circ \underline{\leftrightarrow}_{\mathrm{dsb}}\text{-}min_{\mathsf{LTS}} \circ \mathsf{lts}$.

Proof. Similar to Theorem 7, using Lemma 2 instead of Lemma 1. □

4.2 Minimisation in LTS via Minimisation in KS

In the previous section, we showed that one can minimise in KS with respect to bisimilarity or stuttering equivalence, using the embedding lts, a matching equivalence relation in LTS and converting to KS again. In a similar vein, we propose a reverse translation for ks, which allows one to return to LTS from KS. We first characterise a set of Kripke Structures that are amenable to translating to Labelled Transition Systems.

Definition 18. *Let $K = \langle S, AP, \rightarrow, L \rangle$ be a Kripke Structure. Then K is reversible iff*

1. $AP = Act \cup \{\bot\}$ *for some set Act;*
2. $|L(s)| = 1$ *for all* $s \in S$*;*
3. *for all s for which $\bot \notin L(s)$, we require that for all s', s'', $s \rightarrow s'$ and $s \rightarrow s''$ implies both $s' = s''$ and $L(s') = \{\bot\}$.*

Proposition 3. *Let K be an arbitrary reversible Kripke Structure. Then $K_{/\sim}$, for $\sim \in \{\underline{\leftrightarrow}, \approx_{\mathrm{s}}\}$, is reversible.*

Definition 19. *Let $K = \langle S, AP, \rightarrow, L \rangle$ be a reversible Kripke Structure. The Labelled Transition System $\mathsf{ks}^{-1}(K)$ is the structure $\langle S', Act, \rightarrow \rangle$, where:*

- $S' = \{s \in S \mid L(s) = \{\bot\}\}$*;*
- *Act is such that* $Act = AP \setminus \{\bot\}$*;*
- $\rightarrow$ *is the least relation satisfying:*

$$\frac{s \rightarrow s' \qquad L(s) = L(s')}{s \xrightarrow{\tau} s'} \qquad\qquad \frac{s \rightarrow s'' \qquad a \in L(s'') \setminus \{\bot\} \qquad s'' \rightarrow s'}{s \xrightarrow{a} s'}$$

Proposition 4. *We have* $\mathsf{ks}^{-1} \circ \mathsf{ks} = \mathsf{Id}$.

Without further elaboration, we state the final results.

Theorem 9. *We have* $\underline{\leftrightarrow}\text{-}min_{\mathsf{LTS}} = \mathsf{ks}^{-1} \circ \underline{\leftrightarrow}\text{-}min_{\mathsf{KS}} \circ \mathsf{ks}$.

Theorem 10. *We have* $\underline{\leftrightarrow}_{\mathrm{dsb}}\text{-}min_{\mathsf{LTS}} = \mathsf{ks}^{-1} \circ \approx_{\mathrm{s}}\text{-}min_{\mathsf{KS}} \circ \mathsf{ks}$.

5 Conclusions

Our results in Section 3 naturally extend the fundamental results obtained by De Nicola and Vaandrager in [7,8]. In a sense, we could state that their embeddings ks and lts are canonical for four commonly used equivalence relations.

While the stated embeddings have traditionally been used to come to results about the correspondence between logics, the question whether they support minimisation modulo behavioural equivalences was never answered. Thus, in addition to the above stated results, we proved that indeed the embeddings ks and lts can serve as basic tools in the problem of minimising modulo a behavioural equivalence relation. To this end, we defined inverses of the embeddings to compensate for the fact that composing ks and lts does not lead to transition systems that are comparable (in whatever sense) to the one before applying the embeddings. The latter results are clearly interesting from a practical perspective, allowing one to take full advantage of state-of-the-art minimisation tools available for one computational model, when minimising in the other.

Our minimisation results are for two of the most commonly used equivalence relations that are, arguably, still efficiently computable. However, we do intend to extend our results also in the direction of (completed) trace equivalence and similarity. As a slightly more esoteric research topic, one could look for improving on the embedding lts, as, compared to the embedding ks, it introduces more "noise". For instance, it yields Labelled Transition Systems that have runs that cannot sensibly be related to paths in the original Kripke Structure.

References

1. Browne, M.C., Clarke, E.M., Grumberg, O.: Characterizing finite Kripke structures in propositional temporal logic. Theor. Comput. Sci. 59, 115–131 (1988)
2. Emerson, E.A., Lei, C.L.: Model checking under generalized fairness constraints. Technical report (1984)
3. 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)
4. van Glabbeek, R.J., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. Journal of the ACM 43(3), 555–600 (1996)
5. Groote, J.F., Vaandrager, F.W.: An efficient algorithm for branching bisimulation and stuttering equivalence. In: Paterson, M. (ed.) ICALP 1990. LNCS, vol. 443, pp. 626–638. Springer, Heidelberg (1990)
6. De Nicola, R., Fantechi, A., Gnesi, S., Ristori, G.: An action-based framework for verifying logical and behavioural properties of concurrent systems. Computer Networks and ISDN Systems 25(7), 761–778 (1993)
7. De Nicola, R., Vaandrager, F.W.: Action versus state based logics for transition systems. In: Guessarian, I. (ed.) LITP 1990. LNCS, vol. 469, pp. 407–419. Springer, Heidelberg (1990)
8. De Nicola, R., Vaandrager, F.W.: Three logics for branching bisimulation. Journal of the ACM 42(2), 458–487 (1995)
9. Reniers, M.A., Willemse, T.A.C.: Folk theorems on the correspondence between state-based and event-based systems (2010), arxiv.org/abs/1011.0136
10. van Glabbeek, R.J., Luttik, B., Trcka, N.: Branching bisimilarity with explicit divergence. Fundam. Inform. 93(4), 371–392 (2009)

Privacy, Liveliness and Fairness for Reputation*

Stefan Schiffner[1], Sebastian Clauß[2], and Sandra Steinbrecher[2]

[1] K.U.Leuven, ESAT/SCD/COSIC and IBBT
Kasteelpark Arenberg 10
B-3001 Leuven-Heverlee, Belgium
Stefan.Schiffner@esat.kuleuven.be

[2] Technische Universität Dresden
Institute of Systems Architecture
D-01062 Dresden, Germany
Sebastian.Clauss@tu-dresden.de|steinbrecher@acm.org

Abstract. In various Internet applications, reputation systems are typical means to collect experiences users make with each other. We present a reputation system that balances the security and privacy requirements of all users involed. Our system provides *privacy* in the form of information theoretic relationship anonymity w.r.t. users and the reputation provider. Furthermore, it preserves *liveliness*, i. e., all past ratings can influence the current reputation profile of a user. In addition, mutual ratings are forced to be simultaneous and self rating is prevented, which enforces *fairness*. What is more, without performing mock interactions —even if all users are colluding— users cannot forge ratings. As far as we know, this is the first protocol proposed that fulfills all these properties simultaneously.

1 Introduction

Many Internet applications integrate reputation systems [1]. Before interacting with others, users investigate their potential interaction partners' reputation profiles to find out whether they can trust them. As a reputation profile contains personal data, a compromise is needed between:

Liveliness. That means reputation should always consider all interactions. In particular the reputation system should not offer users a final state in which bad behavior no longer damages their reputation. This requirement addresses two problems many privacy-preserving reputations systems have: the exclusion of negative feedback, and the possible neglection of single ratings [2].

Fairness. Both interaction partners need to trust in the other one's correct behavior during rating. According to Camerer et al. and Dasgupta [3, 4] this is a trust game. This trust game is fair if every user has equal possibilities for rating interaction partners.

Privacy. Explicit reputation is personal data (Bygrave [5]) and should only be accumulated about users who agreed on this. Furthermore, reputation should be protected

* This work was supported by the Integrated Project ICT-2007-216483 PrimeLife on Privacy and Identity Management in Europe for Life. Further, it was supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government and by the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).

by means of technical data protection, as outlined by Mahler and Olsen [6]. In addition, privacy is in fact a pre-condition for fairness: To make a user's rating reflect the outcome of the trust game both users have to rate each other without any knowledge of the other one except his previous reputation and the interaction rated. This prevents retaliation based on the other one's just given rating. Also on a long run, a user who got a negative rating in the past must not be able to re-identify the user who rated her.

To our knowledge, none of the existing systems implements all these requirements. After briefly describing related work in Sect. 2 we show in Sect. 3 a reputation system that fulfills all: our system provides information-theoretic relationship anonymity, it preserves liveliness of reputation, and it provides fairness by allowing mutual ratings between interaction partners in such a way that none of the interaction partners can avenge a negative rating. In this work we do explicitly exclude prevention of mock interactions. Such prevention could be done outside the reputation system, e.g., by anonymously proving that an actual interaction took place, or by introducing transaction costs.

Further, our system fulfills the 'usual' requirements presented in [7]. We analyze our system with respect to those requirements in Sect. 4. Finally, we summarize our results and discuss open issues in Sect. 5.

2 Related Work

An overview on reputation system architectures is, e. g., provided by Voss in [8] while possible reputation functions are, e. g., outlined by Mui in [9]. For an economic introduction, we refer to Dellarocas' work [10]. In [7, 11, 8] the design of reputation systems is investigated from the perspective of privacy and security requirements, which is our focus. Below, we outline related work on privacy friendly reputation systems. Finally, we focus on incombinability of current approaches on liveliness and privacy with fairness and privacy.

Privacy-respecting reputation systems aim at ensuring anonymity of all users involved, namely the anonymity of the rater, the ratee and the users inquiring others' reputation. A *ratee is anonymous* if it is impossible to re-identify her in later transactions even for the rater. Analogously, a *rater is anonymous* if it is impossible to re-identify her in later transactions even for the ratee. On the other hand, an *inquiring user* is anonymous if it is impossible to re-identify her even if she inquires the reputation of a certain user more than once. In [12], Pavlov et al. propose to use anonymity services to achieve privacy for reputation systems; however, this approach only protects the inquiring user. In order to obtain anonymity of raters and ratees, it needs to be ensured that many users are indistinguishable by an attacker, so that they are in large anonymity sets. The possibility of recognizing users by reputation is limited if the set of possible reputations is limited [11] or if the reputation is only published as an estimated reputation, as proposed by Dellarocas in [13]. Androulaki et al. and Steinbrecher use transaction pseudonyms to avoid linkability between transactions [14, 15].

In order to obtain *anonymity of raters*, interactions and ratings related to these interactions need to be unlinkable. This can be reached by a reputation provider who only calculates a new user reputation after it collected not only one but several ratings, as in [16] by Dellarocas et al., or who only publishes an estimation of the actual reputation

[13]. Further, a rater can be anonymous against the reputation provider by using convertible credentials [11] or electronic cash [14, 2]. Furthermore, Kerschbaum proposed in [17] a provable secure reputation system. This system uses two trusted third parties to make ratings and reputations unlinkable, but does not provide anonymity for the interaction partners as their use case is the physical world where addresses are needed for good delivery and money transfer. From these systems none fulfils all our requirements.

Although *fairness* in addition to privacy can be reached by using fair exchange protocols for ratings as suggested in [18], the systems described in [14, 2] cannot apply this straight forward, because they make use of anonymous electronic cash, and most e-cash protocols are interactive.

3 The Reputation System

System Overview

For our system environment, we assume a community system allowing interactions among users. These interactions cause costs. An example of such a system is a marketplace where every user can be a seller (provider) or a buyer (client). Let U_2 be such a user offering interactions to other users. The community deploys a reputation system provided by a reputation provider *ReP*, who is keeping overall reputation accounts for all users of the community system. *ReP* collects positive and negative experiences of users' behavior during interactions in the form of interaction-derived reputation. If a user U_1 becomes interested in the interaction offered by U_2, she inquires U_2's reputation and based on this possibly signals her willingness to interact. U_2 may then also inquire U_1's reputation before she also agrees to the interaction. After their interaction, both rate each other and their rating is included in the other's reputation account at *ReP*. The reputation accounts contain all past ratings a user got, hence the reputation function can be arbitrary. The selection of an informative yet privacy-respecting reputation function is beyond the scope of this work. In Fig. 1 the actions users can perform in their specific roles are shown. Note the users are already registered with *ReP*.

System Details

In this section we detail our protocol. Users who want to use the system need to enroll as member with *ReP* to get a reputation account, which is bound to a long term

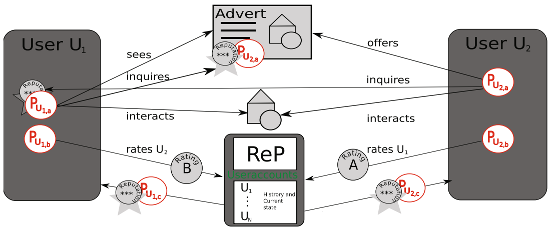


Fig. 1. Actions, user roles and items in the system

pseudonym. Then, the system itself works round-based. We denote the number of users in a round by N. Note that for every round this number can change, i. e., users can join and leave the system. In every round, a user can interact with at most one other user. During a round, this grouping of users into interaction partners cannot change. *ReP* maintains a reputation account for every user under the long-term pseudonym U_i. In this account it retains a chronological list that contains vectors of two bits for every round. The first bit of each of these vectors is the registration bit, which is 1 if the user registered an interaction for this round and 0 otherwise. The second bit is the received rating bit, which is 1 if the user received a positive rating and 0 otherwise.

During membership enrollment, *ReP* exchanges with every user a symmetric encryption key for private communication between *ReP* and the user (for readability we do not denote this encryption), and a message authentication key called *MAC key* that enables *ReP* to sent authenticated multicast messages to all users. Moreover, it holds a key $k_{ReP_{U_i}}$ for confidential and anonymous sending (via a DC-Net, see [19] for a explanation of the DC-Net) that it does not share with the user. We assume all keys to be long enough, so that for every round a fixed new part can be used so that the key lasts for many rounds.

All users and *ReP* are part of a DC-Net, that is, they share DC-Net keys with some users. Note, that the key sharing graph of a DC-Net only needs to be connected. How many keys a single participant shares with peers is a matter of trust, since a DC-Net participant can be de-anonymized if all users that share keys with her collude. For the rating phase *ReP* shares two symmetric rating keys with every user, namely one for a homomorphic encryption scheme and one for authenticating the rating token. The following steps are done in every round.

A round consists of 3 phases each implemented by a protocol described below, namely *initialization*, *registration* and *rating*. In addition there is a *show reputation* protocol, which can be performed whenever needed. These protocols are explained in detail in the following subsections.

Initialization. *ReP* generates a $M > N$ long bit vector that consists of one bit for every user initialized with 0. The vector needs to be longer than N to avoid random attacks, as we will show in Sect. 4.1. Then *ReP* generates the following tokens for every user U_i and sends these to him:

Registration token: *ReP* chooses a secret random number in the interval $[1 : M]$ which it has not chosen for another user. Then it forms M bit long vector U_i that is all 0, but in the position indicated by the chosen number for this user. We denote this vector as U_i, since it is in fact the ID of this user for this round. *ReP* encrypts this ID with the symmetric key $k_{ReP_{U_i}}$ it generated for the user, but does not share with the user. The encrypted token $k_{ReP_{U_i}}(U_i)$ is U_i's registration token, which *ReP* sends to the user via a secure channel and stores the position as the user's round registration ID.

Rating token: U_i's rating token is formed by a unique random number *ReP* assigned to her for this round and a MAC suitable for an authenticated multicast. Hence, every user can check whether the token is valid or not, but no adversary can test if a token is valid for a user that is not controlled by him. The adversary would need the MAC key of a user to check if a token would be accepted by this user. This token is both stored and also securely sent to U_i as round rating ID.

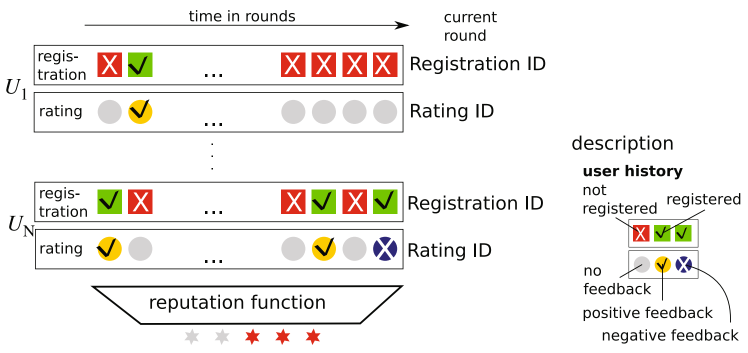


Fig. 2. Internal state of *ReP* with track of the past ratings and current user IDs

Internal state of ReP: In Fig. 2 an internal state of *ReP* is shown. Every user has a registration and rating history. From this, *ReP* calculates the reputation of a user by using the reputation function. As outlined above we abstract from the concrete reputation function used. In the column *current round* is shown which data needs to be stored in every round, namely the registration ID, which represents the user as a position in a vector and the rating ID, which is used as single-use return address for the rating phase.

Registration. Two users who are willing to interact establish a secure channel, e. g., using a authenticated Diffie-Hellman key exchange. Via this secure channel, they exchange their registration tokens encrypted with their DC-Net keys and their rating tokens. Both users check if the rating tokens are valid. For this, every user uses her MAC key to verify the MAC generated by *ReP* for this rating token. If the rating token is valid the encrypted registration tokens are sent to *ReP* via a secure channel (Fig. 3, *registration*.) Otherwise the user refuses the interaction. All users who decided *not* to interact in this round send a M bit long 0 vector encrypted with their DC-Net keys, and the declaration that it is in fact a 0 vector, to *ReP*.

ReP calculates the sum of all received tokens. As a result it gets the sum of all vectors encrypted with keys $k_{ReP_{U_i}}$ of all users U_i who want to interact with another user in this round. Since the users who will not interact with any other user disclosed this to *ReP*, *ReP* knows which keys to use to decipher.[1] If all users acted correctly, the resulting vector has a 1 at every position that represents a user who wants to interact. If a single user sends a malformed message, or uses her registration token more than once, the vector becomes meaningless with a high probability, i. e., the redundant bits are wrong. In the first case *ReP* sends to every user an acknowledgment message and the respective users can proceed with their interactions independently from *ReP*. In the second case *ReP* sends a fail message to every user and the protocol stops or is restarted.

We are aware of techniques to make DC-Nets robust against malicious users, such as presented in [20], but those are beyond the scope of this work. Fig. 3 illustrates the

[1] So these messages are in fact dummy messages, which *ReP* does not need at all, but it prevents from outsider attacks on the communication layer.

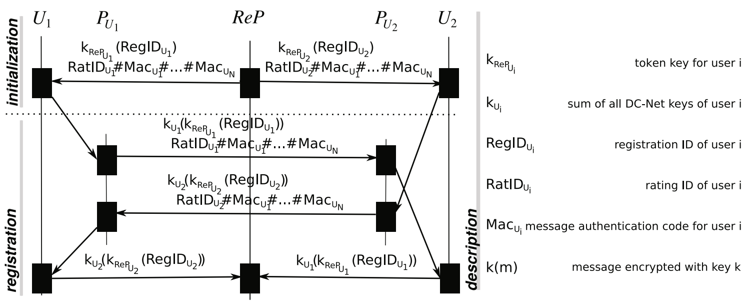

initialization
registration
description
U_1
P_{U_1}
ReP
P_{U_2}
U_2
$k_{ReP_{U_1}}(RegID_{U_1})$
$RatID_{U_1}\#Mac_{U_1}\#...\#Mac_{U_N}$
$k_{ReP_{U_2}}(RegID_{U_2})$
$RatID_{U_2}\#Mac_{U_1}\#...\#Mac_{U_N}$
$k_{U_1}(k_{ReP_{U_1}}(RegID_{U_1}))$
$RatID_{U_1}\#Mac_{U_1}\#...\#Mac_{U_N}$
$k_{U_2}(k_{ReP_{U_2}}(RegID_{U_2}))$
$RatID_{U_2}\#Mac_{U_1}\#...\#Mac_{U_N}$
$k_{U_2}(k_{ReP_{U_2}}(RegID_{U_2}))$
$k_{U_1}(k_{ReP_{U_1}}(RegID_{U_1}))$
$k_{ReP_{U_i}}$ token key for user i
k_{U_i} sum of all DC-Net keys of user i
$RegID_{U_i}$ registration ID of user i
$RatID_{U_i}$ rating ID of user i
Mac_{U_i} message authentication code for user i
$k(m)$ message encrypted with key k

Fig. 3. DC-Net-based interaction registration for a system with 2 users

initialization and *registration* protocol for two users U_1 and U_2 who know each other by the pseudonyms P_{U_1} and P_{U_2} respectively. This is run by any two users who want to interact.

Rating. After an interaction — in which *ReP* is not involved — the interaction partners rate each other. Therefore, the rating token that was exchanged during registration, viz., the token the ratee sent to the rater, is used. As described before, the rating token contains a random number which represents the ratee to *ReP*. The rater concatenates a rating bit (1 for a positive rating and 0 otherwise) to the rating token and sends this message via the DC-Net to *ReP*, thereby *ReP* does not publish its contribution to the DC-round, to avoid that other users learn the value of the feedback. If *ReP* keeps its part secret the message is effectively encrypted for it. Otherwise – as a DC-Net is a broadcast – every entity having the contribution of all users could compute the resulting message.

Contrary to the registration, *ReP* needs to receive every single rating message, not only their sum. For every rating a separate DC-Net round for sending is needed. Who actually sends in the next DC-round, is determined by anonymous reservation. This makes message collisions unavoidable. The most efficient collision resolving algorithm [21, 22] broadcasts the result of the collision. In order to avoid that the users learn how many positive feedback was given in a round, we use only the rating token part for resolving collisions and encrypt the feedback with the rating encryption key. Furthermore the feedback needs to be secured against attacks that leverage the homomorphic property of the encryption scheme. If the attacker adds an attack vector in the DC-round, the feedback must be destroyed with a high probability. This can be done by a randomized and redundant encoding of the two rating values. Thereby, a large pool of encodings is needed, so that the attacker cannot guess which of the encodings is used and for every encoding a different attack vector is needed to flip the rating. More formally, there must not exist an attack vector such that for many pairs of encodings of positive rating and negative rating, the sum of the negative rating and the attack vector results in a positive rating (and vice versa).

Show Reputation. If a user wants to show her reputation to someone, she needs *ReP*. First, the user sends the pseudonym, under which she wants to show the reputation to an inquiring user, to *ReP*. *ReP* concatenates her current reputation, the pseudonym received from the user and the current time. Further, for these concatenated parts *ReP* generates a MAC suitable for an authenticated multicast to all users. Then *ReP* sends this authenticated message to the user. The user sends this message to the inquiring user. The inquiring user can then decide whether she wants to interact or not. If she intends to interact, she uses the transaction pseudonym to contact the other user.

4 System Analysis

Requirements. As pointed out in [2], the following requirements should be fulfilled by a reputation system with regards to security (1-6) and privacy (7-9):

(1) *Integrity of ratings:* Ratings are preserved from manipulations.
(2) *Authorizability of ratings:* Only interaction partners may rate each other.
(3) *Liveliness of reputation:* Reputation considers all recent interactions.
(4) *Availability of reputation:* Inquirers are able to access other users' reputation. This problem is beyond the scope of this work.
(5) *Absolute linkability of users' registration:* Users can register only once. This problem is solved outside the comunication network.
(6) *Fairness of the underlying game-theoretic trust game:* Every user has equal possibilities for rating interaction partners but only them.
(7) *Raters' anonymity:* Users can rate anonymously, this is:
    (a) Attackers cannot link ratings and the respective raters.
    (b) Attackers cannot link ratings and the corresponding interactions.
(8) *Inquirers' anonymity:* Users can inquire reputation anonymously.
(9) *Ratees's anonymity:* Ratees are unlinkable to their past interactions, except that these contributed to their reputation.

Attackers. Because of the different attack goals we distinguish between *security* and *privacy attacker*. Both outsiders and all parties involved (users and reputation provider) can be interested in breaking security and privacy requirements.

Even though *ReP* behaves according to the protocol, it might be interested in who rates whom, and it might be interested in building user profiles from this information. As the privacy requirements (7), (8), and (9) aim at preventing this, *ReP* can be a privacy attacker as well. Hence, we assume *ReP* to be a *honest but curious* attacker.

For the security attacker, we assume a global attacker who can observe as well as modify all interactions between users and between users and *ReP*, but who cannot control *ReP*.

For the privacy attacker, we assume that he cannot observe who is communicating with whom, that is, all users are using an anonymity service on the communication layer. We further assume that the privacy attacker can only control a limited number of users so that a sufficiently large anonymity set (which contains the users not controlled by the attacker) is preserved. Finally, we assume that *ReP* does not cooperate with a user who takes part in an interaction as this would allow *ReP* to de-anonymize the user who interacts with this colluding user, because it can link received rating tokens to users.

4.1 Security Analysis

In general, our proposed reputation system is designed to preserve **fairness of the underlying game-theoretic trust game (6)**, as it enables both interaction partners and only them to rate each other and only for the interaction considered. We now need to show that *none can rate another user she has not interacted with* and that *none can forge ratings*. The latter would also damage the integrity of ratings (1).

The probabilities of a successful attack in the *registration and the rating phase* are given in the following.

Probabilities of forged registrations. In the *registration* protocol for an interaction users get a rating token that they need to give a rating in the rating phase. The *registration* protocol can only be performed successfully, if the *initialization* protocol has been completed correctly. As we assume the *ReP* to behave correctly, the registration tokens will be correctly formed. The submission of the tokens to the users is information-theoretically secure encrypted and authenticity is secured by an information-theoretically secure MAC. We do not need a reliable message transmission here, as the registration phase bases on a DC-Net which involves all users in the system. Thus, if a user received no or a malformed registration token she will not cooperate in the DC-Net protocol, and therefore a suppression of a registration token will lead to an invalid registration phase, i. e., both registration and initialization phase need to be repeated. After the *registration* protocol is completed successfully, all interacting users have a correct rating token.

The fairness of the *registration* protocol is provided by the all-or-nothing property of the DC-Net, viz., only if all messages were summed up the keys sum up to 0. Hence, it is not necessary to exchange registration tokens by fair exchange, since whenever a message is missing or manipulated, the *ReP* will detect this with a high probability and will send a fail message to all users. In the following paragraph we show how likely it is that an attacker guesses a registration token which leads to an accepted registration. Note that the attacker has no means to test whether the generated token is accepted before sending the token to the *ReP*.

More formally, with respect to the *registration* protocol, we calculate the probability that an attacker can manipulate a registration token in a meaningful way, i. e., that the manipulated token would sum up with all other contributions to a meaningful registration vector, which makes the *ReP* accept the round. We assume that K users interact, N users are taking part in this round and M is the length of the registration vector, hence $K \leq N \leq M$. The number of all possible registration vectors is 2^M, while the number of all valid sums of all registration vectors is $\frac{2^N}{2}$, since a valid sum can only have ones at N positions and it needs to have an even number of ones. Given the number K of interacting users, there are $N-K$ positions left the attacker can manipulate. Since the attacker cannot test whether his attack vector fullfills these properties the chance for a successful attack equals the propability that he chooses the right attack vector at random. A right vector is a vector which – when summed up with all other inputs – results in an accepted sum of registration vectors. Since the vectors are summed up *mod* 2, for every random vector of length M there are exactly as many attack vectors as there are valid registration vectors. Hence the success probability is: $\frac{2^{N-K}}{2^M}$.

Probabilities to forge a rating. When the *registration* protocol is completed successfully, both interaction partners have correct rating tokens. Now we need to analyze the *rating* protocol with respect to requirement (6). Manipulations of the rating value by users other than the rater should be detected, which provides *integrity of ratings*.

While the attack on the rating token forces a negative rating to some user that did not intend to interact at all, guessing a rating token enables the attacker to give a negative feedback to a user that interacts during this round, since the attacker could send this token to *ReP* and it would be detected as self rating and lead to a negative rating. We demonstrate below how likely manipulations of ratings are (1). With the method of encoding and encrypting the rating presented in Sect. 3, a user could only guess another bitstring for the encrypted rating, but she cannot verify her guess.

We now calculate the probability that a user can guess a correct rating token. Let K be the number of interacting users and M the length of the rating token. Note that the total number of users in the system is not important for this. As *ReP* would not accept a rating token for an unregistered user, only users that have registered an interaction can be attacked this way. Hence, the probability to correctly guess one token is $\frac{K}{2^M}$.

Furthermore, a user may try to rate herself. This can be done without disrupting the *rating* protocol, but if the interaction partner of this user also rated her, *ReP* will detect two ratings for this user, and can penalize her, e. g., with a bad rating. On the other hand, a user may send a 0-message instead of a rating value. Yet, this does not disrupt the *rating* protocol, but enables her interaction partner to rate herself. That is why we do not see any intention a user could have for sending a 0-message.

As for DC-Net in general, users can disrupt the communication by not behaving according to the protocol, i. e., using wrong keys etc. This would lead to disable a rating in this round. On the other hand, there exist techniques which make it possible to detect and ban such DC-Net users, which have been published, e. g., in [20]. Such methods can also be used here in order to eliminate users misbehaving in this protocol.

Liveliness, however, depends on the reputation function, but the reputation function can only preserve it, if the protocol ensures that *ReP* really receives both positive *and* negative ratings. This is ensured by providing *fairness of the underlying game-theoretic trust game (6)* as described above.

Authorizability of Ratings (2) is preserved jointly by the *registration* protocol and the *rating* protocol. During the *registration* protocol, every user receives a rating token from *ReP*, which can be used to rate her during the *rating* protocol. Also during the *registration* protocol, interaction partners must exchange their rating tokens, thereby the correctness of the tokens can be verified by the attached MAC. In this way, after that the *registration* protocol has been finished successfully, both partners have the rating tokens of their interaction partners. As both partners now have their own and their partner's rating token, they could rate themselves as well. If only one partner rates herself, this will lead to a conflict detectable by *ReP* during the *rating* protocol, there will be two ratings for one user, while another user got no ratings. However, if both interaction partners rate themselves, this will not be detected.

Furthermore, authorizability is only computational secure, as an unbounded attacker can observe the communication between two users and break the Diffie-Hellman key exchange. Hence such an attacker can steal rating tokens and use those to give both interaction partners a negative feedback. However, the attacker needs to break the Diffie-Hellman key exchange within the time between registration phase and rating phase of the same round, since a rating token is only valid in the round it is issued. Moreover, the attacker cannot link the tokens to usernames, but only to the transaction pseudonyms.

4.2 Privacy Analysis

We need to distinguish between the outside observer, the attacker who controls one or more users and an attacker who controls the *ReP* and some users. For all of these attackers, we need to analyze the different goals, namely inquirers' anonymity (8), ratees' anonymity (9) and raters' anonymity (7).

Outside attacker. Our protocol is based on the DC-Net. If we assume that a DC-net is used for the communication needed for the actual interaction, an outside attacker can achieve as much as he can by observing a DC-Net. That is, if he observes all communication lines, he can observe how many users are involved in total. However, since in a DC-net essentially everybody always sends and receives he cannot reduce the anonymity set further than this. This holds for all rolls, i. e., rater, ratee and inquiring user since he cannot distinguish them.

Attacker controlling (some) users. We assume that an attacker cannot control too many users; however, it is hard to quantify what 'too many' exactly is, since it depends on the security needs of the users. Furthermore, it is not sufficient to fix a certain fraction of all users. Given an attacker that controls all but one user of a group of users that all have the same reputation, the user who is not controlled by the attacker is effectively de-anonymized, even if the controlled users are only a small fraction of all users.

Inquirers' anonymity (8) is provided by the *show reputation* protocol: Both the inquirer and the user who shows the reputation act under pseudonym, and they communicate using an anonymous communication system as already described. If an attacker controls the showing user, the only extra information he gets with respect to an outside observer is that a user is interested in the reputation of the user under his control. This does not help for attacks within the system, but depending on the actual application the attacker might learn something about the inquirer, since she is interested in a certain product or interaction the user who shows the reputation is offering. However, minimizing the information needed within a interaction is beyond the scope of this work.

For *Ratees's anonymity (9)* an anonymous communication system is needed. Furthermore, all interactions are done under pseudonym, and different pseudonyms can be used for every other round. However, an attacker who controls other users might try to inquire all reputations of all other users. How much impact this has on the anonymity of the ratee depends on the reputation function, more precisely, it depends on how many different reputation values are possible and how those are distributed over the users. Furthermore, it depends on whether an attacker can detect if he queries a user twice or not, i. e., if users have one or more pseudonyms per round.[2]

The *raters' anonymity*: The interaction partners are anonymous to each other, as all communication between them is done under pseudonym, and they communicate using

[2] Users can have more than one offer under different pseudonyms, but can only do one interaction per round.

an anonymous communication service. Hence, if the interaction partner is controlled by the attacker, the only information the attacker learns more than an outside observer is the information revealed by the interaction itself and the rating. However, most of the time, the rating depends more on the interaction than on the user, so that the information an attacker learns is hard to quantify.

Attacker controlling ReP. We assume that *ReP* does only attack privacy, not the security of the system. Indeed, *ReP* is a strong privacy attacker since it can observe all registrations and all feedbacks. However, it cannot observe the actual interactions.

In the *registration* protocol, the users who do not intend to interact in this round indicate this to *ReP*. Hence, *ReP* can distinguish between interacting and non-interacting users. *ReP* will not get more information, as the registration process itself is anonymized in a DC-Net-like way, as described in Sect. 3. If K is the number of users that intend to interact, then for *ReP* it is equally likely with whom of the other $K-1$ other users a user interacted. Note that if there are only a few users interacting, *ReP* has a very high chance to guess correctly who is interacting with whom, e. g., *ReP* is certain about the interaction relation for $K = 2$.

In the *rating* protocol, each interacting user sends the rating token she received from her interaction partner to *ReP*. This is done using a DC-Net as well, so that the interacting users are anonymous within their group. *ReP* only gets the rating tokens together with the ratings. The rating token identifies the ratee, but it does not reveal with whom the ratee interacted. However, as already described in Sect. 4, *ReP* may not cooperate with one of the interacting users. Indeed, when they collude, they can de-anonymize the interaction partner of this user by recognizing the rating token. Furthermore, the *ReP* learns the value of the rating if the attacker can assume that ratings have a high correlation, namely, the opinion over the outcome of a interaction is for both partners often the same, the attacker can change the probabilities learned from the registration phase. However, this knowledge depends on the actual system.

4.3 Efficiency

ReP needs to send one message to every user for initializing a round. During the *registration* protocol, users willing to interact need to send one message each to their interaction partner, therefore they need to agree on a key, e. g., using a Diffie-Hellman key agreement protocol. Furthermore, only one message from every participant to *ReP* is needed to register all interactions. Finally, every user who took part in an interaction has to send a rating, which is only one DC-Net message per participant as well. This means if K users interacted, the collision resolution needs at most K messages per user.

In order to take part in the DC-Net for registration and rating, users need to exchange keys with each other as well as with *ReP*. The key exchange between the users needs to be offline and, furthermore, secure from the *ReP*. The key exchange with *ReP* can be done in parallel with the registration, which can be done by, e. g., paper mail. In order to provide anonymity between the users of a DC-Net, the key graph between the users of the DC-Net needs to be connected. However, the stronger the graph is connected the better anonymity is protected against internal attackers. For information-theoretic anonymity within the DC-Net, the keys need to be random. Calculations within the DC-Net are cheap, as these are only additions modulo the cardinality of the alphabet of the DC-Net. Please refer to [19] for details regarding the DC-Net.

All other encryptions needed are one-time-pad encryptions, i. e., additions modulo a given group. Such encryptions (and the corresponding decryptions) are done by *ReP* for the registration tokens, for the rating tokens, and for the rating part of the DC-Net message in every DC-round of the collision resolution during the rating protocol. As all these crypto operations are done by *ReP*, no key exchange is needed for that.

Moreover, *ReP* has to calculate MACs for the registration token, for the rating token, and for the message from *ReP* to the user during the *show reputation* protocol. For every MAC the cost are the one of applying the respective universal hash functions, that is usually cheap, and a one-time-pad [23].

5 Conclusion

In this paper we presented a reputation system that focuses particularly on fairness, liveliness and privacy, without sacrificing general security requirements.

Liveliness is achieved by registration before the actual interaction, which prevents silently suppressing bad ratings. Our system also provides fairness by enforcing simultaneous ratings between interaction partners, so that both partners have equal possibilities to rate each other. Last but not least it preserves anonymity between interaction partners and preserves privacy from the reputation provider, so that it does not get to know who interacts with (and rates) whom. In contrast to previous approaches that rely on the computational privacy of convertible credentials or anonymous cash our approach provides information-theoretic privacy. However, some information needs to be disclosed, namely the users' reputation. Given a sufficient number of interacting users one needs to choose a reputation function that gives usable feedback to the users on one side, but divides the user set in only a few large groups on the other side.

In Table 1 we compare our system with closely related ones, as described in Sect. 2.

Table 1. Comparison of reputation protocols

	Monotonic [14]	Non-monotonic [2]	this work
Authorizability (2)	no	yes	yes
Liveliness (3)	no, only-positive ratings	yes, negative ratings	yes, negative ratings
Fairness (6)	no	no, mutual ratings	yes
Anonymity (9)	yes, but timing	yes, less timing issues	yes

For future work, we plan a formal analysis of the natural bounds with respect to the tradeoff between privacy of the parties involved and the information needed. Second we plan to work on practicability. For example we assume registration and rating to be done synchronously. We will have to relax this in future work to asynchronism to make the protocol more flexible.

References

[1] Resnick, P., Kuwabara, K., Zeckhauser, R., Friedman, E.: Reputation systems. Communications of the ACM 43(12), 45–48 (2000)

[2] Schiffner, S., Clauß, S., Steinbrecher, S.: Privacy and liveliness for reputation systems. In: Martinelli, F., Preneel, B. (eds.) EuroPKI 2009. LNCS, vol. 6391, pp. 209–224. Springer, Heidelberg (2010)

[3] Camerer, C., Weigelt, K.: Experimental tests of a sequential equilibrium reputation model. Econometrica 56, 1–36 (1988)

[4] Dasgupta, P.: Trust as a commodity. In: Gambetta, D. (ed.) Trust: Making and Breaking Cooperative Relations, Department of Sociology, pp. 49–72. University Oxford (2000)

[5] Bygrave, L.: Data Protection Law, Approaching Its Rationale, Logic and Limits, p. 448. Kluwer Law International, The Hague (2002)

[6] Mahler, T., Olsen, T.: Reputation systems and data protection law. In: eAdoption and the Knowledge Economy: Issues, Applications, Case Studies, pp. 180–187. IOS Press, Amsterdam (2004)

[7] ENISA: Position paper. reputation-based systems: a security analysis (2007), `http://www.enisa.europa.eu/doc/pdf/deliverables/enisa_pp_reputation_based_system.pdf` (last visit 16/06/09)

[8] Voss, M.: Privacy preserving online reputation systems. In: International Information Security Workshops, pp. 245–260. Kluwer, Dordrecht (2004)

[9] Mui, L.: Computational Models of Trust and Reputation: Agents, Evolutionary Games, and Social Networks. PhD Thesis, Massachusetts Institute of Technology (2003)

[10] Dellarocas, C.: The digitization of word-of-mouth: Promise and challenges of online feedback mechanisms. Management Science, 1407–1424 (October 2003)

[11] Steinbrecher, S.: Enhancing multilateral security in and by reputation systems. In: Matyáš, V., Fischer-Hübner, S., Cvrček, D., Švenda, P. (eds.) IFIP/FIDIS 9.2, 9.6/11.6, 11.7/FIDIS. IFIP Advances in Information and Communication Technology, vol. 298, pp. 135–150. Springer, Heidelberg (2009)

[12] Pavlov, E., Rosenschein, J.S., Topol, Z.: Supporting privacy in decentralized additive reputation systems. In: The Second International Conference on Trust Management, Oxford, United Kingdom, pp. 108–119 (March 2004)

[13] Dellarocas, C.: Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In: Proceedings of the 2nd ACM conference on Electronic commerce, EC 2000, pp. 150–157. ACM Press, New York (2000)

[14] Androulaki, E., Choi, S.G., Bellovin, S.M., Malkin, T.G.: Reputation systems for anonymous networks. In: Borisov, N., Goldberg, I. (eds.) PETS 2008. LNCS, vol. 5134, pp. 202–218. Springer, Heidelberg (2008)

[15] Steinbrecher, S.: Design options for privacy-respecting reputation systems within centralised internet communities. In: Proceedings of IFIP Sec 2006, 21st IFIP International Information Security Conference: Security and Privacy in Dynamic Environments. IFIP, vol. 201, pp. 123–134. Springer, Heidelberg (2006)

[16] Dellarocas, C.: Research note – how often should reputation mechanisms update a trader's reputation profile? Information Systems Research 17(3), 271–285 (2006)

[17] Kerschbaum, F.: A verifiable, centralized, coercion-free reputation system. In: Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society, WPES 2009, pp. 61–70. ACM, New York (2009)

[18] Steinbrecher, S.: Balancing privacy and trust in electronic marketplaces. In: Katsikas, S.K., López, J., Pernul, G. (eds.) TrustBus 2004. LNCS, vol. 3184, pp. 70–79. Springer, Heidelberg (2004)

[19] Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology 1, 65–75 (1988)

[20] Waidner, M., Pfitzmann, B.: The dining cryptographers in the disco: unconditional sender and recipient untraceability with computationally secure serviceability. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, p. 690. Springer, Heidelberg (1990)

[21] Waidner, M.: Unconditional sender and recipient untraceability in spite of active attacks. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 302–319. Springer, Heidelberg (1990)

[22] Pfitzmann, A.: Diensteintegrierende Kommunikationsnetze mit teilnehmerüberprüfbarem Datenschutz. Phd thesis, Universität Karlsruhe, Fakultät für Informatik, Heidelberg (1989)

[23] Stinson, D.R.: Universal hashing and authentication codes. Des. Codes Cryptography 4(4), 369–380 (1994)

Minimizing Interference for the Highway Model in Wireless Ad-Hoc and Sensor Networks[⋆]

Haisheng Tan[1], Tiancheng Lou[2], Francis C.M. Lau[1], Yuexuan Wang[2], and Shiteng Chen[2]

[1] Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong, China
[2] Institute for Theoretical Computer Science, Tsinghua University, Beijing, 100084, China

Abstract. Finding a low-interference connected topology is one of the fundamental problems in wireless ad-hoc and sensor networks. The receiver-centric interference on a node is the number of other nodes whose transmission ranges cover the node. The problem of reducing interference through adjusting the nodes' transmission ranges in a connected network can be formulated as that of connecting the nodes by a spanning tree while minimizing interference. In this paper, we study minimization of the average interference and the maximum interference for the high-way model, where all the nodes are arbitrarily distributed on a line. Two exact algorithms are proposed. One constructs the optimal topology that minimizes the average interference among all the nodes in polynomial time, $O(n^3\Delta^3)$, where n is the number of nodes and Δ is the maximum node degree. The other algorithm constructs the optimal topology that minimizes the maximum interference in sub-exponential time, $O(n^3\Delta^{O(k)})$, where $k = O(\sqrt{\Delta})$ is the minimum maximum interference.

Keywords: wireless ad-hoc and sensor networks, interference minimization, topology control, combinatorial optimization, dynamic programming.

1 Introduction

Wireless ad-hoc and sensor networks consist of a set of nodes deployed across a region of interest. Each node has limited processing ability and is equipped with a wireless radio for communication. Compared with traditional wired networks, they do not have a fixed infrastructure. The nodes can adjust their transmission powers to achieve their desired transmission ranges which then form a multi-hop network. Wireless ad-hoc and sensor networks have many applications in real life such as environmental monitoring, intrusion detection, and health care. It is regarded as one of the most popular networking paradigms.

Due to the environments in which they are typically deployed, wireless nodes can only use relatively weak batteries. Energy is therefore at a premium which

[⋆] The work presented in this paper was supported in part by Hong Kong RGC-GRF grants (714009E), the National Basic Research Program of China Grant Nos. 2007CB807900, 2007CB807901 and the National Natural Science Foundation of China Grant No. 61073174.

however is critical for the network's lifetime. One direction for energy conservation is to reduce interference which occurs when communication between two nodes is interfered by another concurrent transmission nearby. Different models have been defined to depict the phenomenon [3,4,5,6,7]. Paper [14] proposed a stable and realistic interference model, called *the receiver-centric model*, where the interference on a node v is the number of other nodes whose transmission ranges cover v (Figure 1). In this paper, unless specified, the receiver-centric model is assumed.

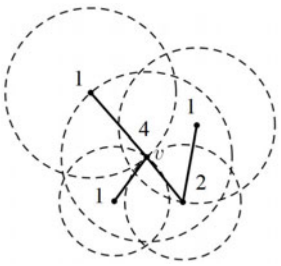


Fig. 1. The receiver-centric interference: the numbers are interference on each node

Topology control refers to selecting only a subset of the available communication links for data transmission, which has been widely used to construct networks with specific properties such as planarity, bounded node degree, the spanner property and low interference [13,1,10,8]. Researchers are not only interested in minimizing the average interference on the nodes, but also the maximum interference, because the maximum interference is closely related to the time when the first node runs out of energy, which could mean a halt of the entire network's operation. The problem of minimizing the maximum interference while preserving connectivity in two-dimensional networks has been proved to be NP-complete [2]. Authors of [9] proposed an algorithm that could bound the maximum interference by $O(\sqrt{\Delta})$ using the $\varepsilon - net$ theory in computational geometry. Here, n is the number of nodes and Δ is the maximum node degree in the topology when each node is set to the maximum transmission range and connected to all the other nodes in its range (If all the nodes have the same maximum transmission range, the topology is actually a unit-disk-graph). For minimizing average interference in 2D networks, paper [12] developed an asymptotically optimal algorithm with an approximation ratio of $O(logn)$. Researchers are also interested in the interference problem in 1D networks as there are also many application scenarios for 1D networks, such as bridges and tunnels. For minimizing the maximum interference on the exponential chain, authors in [14,15] proposed an asymptotically optimal algorithm and proved a tight lower bound of $\Omega(\sqrt{\Delta})$. Here *the exponential chain* means the nodes are distributed on a 1D line with the distances growing exponentially. Furthermore, for the general case, in which the nodes are arbitrarily distributed on a line, the so called *highway model*, they bounded the minimum maximum interference by $O(\sqrt{\Delta})$ and presented an approximation with ratio of $O(\sqrt[4]{\Delta})$.

In this paper, we study minimization of the average and the maximum interference for the highway model. Two exact algorithms are proposed. One is to construct the optimal connected topology with minimum average interference in $O(n^3\Delta^3)$ time. The other constructs the connected topology with minimum maximum interference. Here, minimizing the maximum interference is related to the second open problem proposed in [11], but we restrict the model to one dimension and add a constraint on the maximum transmission range of the nodes. Our algorithm runs in sub-exponential time, $O(n^3\Delta^{O(k)})$, where $k = O(\sqrt{\Delta})$ is the minimum maximum interference. We can see when Δ is small, which means a low maximum node degree, our algorithm is fast. To our knowledge, the former algorithm is the first polynomial-time algorithm for minimizing the average interference and the latter is the first sub-exponential-time algorithm for minimizing the maximum for the highway model.

The rest of the paper is organized as follows. In Section 2, we give the formal definitions of the interference model and the problem. Section 3 describes the no-cross property and gives the algorithm to minimize the average interference for the highway model. Section 4 describes how to minimize the maximum interference. Section 5 concludes the paper and points out some open problems and possible future work.

2 Models and Problem Definitions

We assume a wireless ad-hoc and sensor network in which the nodes are stationary after deployment in a region. If at some point they need to be moved, we can re-run the proposed algorithms using the new coordinates. The maximum transmission radius of the nodes is denoted as r_{max}. Each node can self-adjusts its transmission radius from 0 to r_{max} in a continuous manner. There are no obstacles to block the communications. Therefore, the maximum transmission range of a node v will be the disk centered at v with radius r_{max}. For the highway model, we assume r_{max} is not shorter than the farthest distance between two consecutive nodes, or else it is not possible to construct connected topology.

The network is modeled as an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of communication links. For the highway model, the n nodes in $V = \{v_0, v_1, ..., v_{n-1}\}$ are arbitrarily deployed along a line from left to right. We can view the line as an x-axis, and $v_0 = 0$. Then, each node u is denoted as its x-coordinate. An edge $(u, v) \in E$ exists only if both their transmission radii, r_u and r_v, are not shorter than their Euclidean distance $|u-v|$. Therefore, in G, the transmission radius of a node is equal to the distance to its farthest neighbor (Two nodes are neighbors means there is an edge incident on them.). In addition, we introduce the following terms. For a segment $\overline{v_sv_t}$ on the line, where $s \leq t$, the nodes *located on* $\overline{v_sv_t}$ are $\{v_s, v_{s+1}, ..., v_{t-1}, v_t\}$; the nodes outside $\overline{v_sv_t}$ are the other nodes that are not on it; the nodes *inside* $\overline{v_sv_t}$ are $\{v_{s+1}, v_{s+2}, ..., v_{t-1}\}$.

The receiver-centric interference model is adopted. The interference of a node v, denoted as $RI(v)$, is defined as the number of other nodes whose transmission ranges can cover v:

$$RI(v) = |\{u | u \in V/\{v\}, |u - v| \leq r_u\}|. \tag{1}$$

The average node interference in G, $RI_{avg}(G)$, can be defined as:

$$RI_{avg}(G) = \frac{\sum_{v \in V} RI(v)}{|V|}. \tag{2}$$

The maximum node interference, $RI_{max}(G)$, can be defined as:

$$RI_{max}(G) = max_{v \in V}\ RI(v). \tag{3}$$

Besides minimizing interference, we also need to preserve the network connectivity. Therefore, the optimal topology with the minimum interference should be a spanning tree on V. Therefore, our problems can be defined as:

Given n nodes arbitrarily distributed on a 1D line, construct a spanning tree, $G = (V, E)$, to connect all the nodes with edges no longer than r_{max}. The minimization of the average interference problem is to construct a spanning tree that minimizes $RI_{avg}(G)$, and the minimization of the maximum interference problem is to construct a spanning tree that minimizes $RI_{max}(G)$.

3 Minimizing the Average Interference

3.1 No-Cross Property

For a spanning tree $G = (V, E)$ constructed on the nodes along a line, we can draw all the edges on one side of the line. A *cross* means there are two edges that share at least a common point excluding their endpoints (Figure 2(a)). By

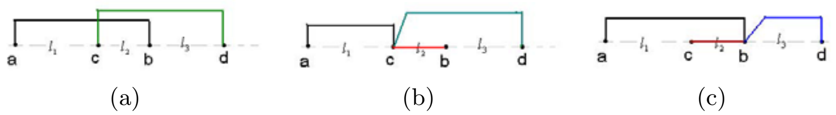


Fig. 2. a, b, c and d are four nodes distributed on a line, where $l_1 = c - a$, $l_2 = b - c$, and $l_3 = d - b$, and (a, b) and (c, d) are two edges: (a) (a,b) and (c, d) have a cross; (b) the cross removed when $l_1 \leq l_2 + l_3$; (c) the cross removed when $l_1 > l_2 + l_3$

adding and deleting edges, we show below that a cross can be removed without increasing interference on any nodes while preserving the network connectivity.

Theorem 1. *For a spanning tree connecting the nodes on a line with crosses, there is always another spanning tree to remove the crosses without increasing interference on any node.*

Proof. We prove this theorem by illustrating how to remove a cross. Without loss of generality, we handle the cross in Figure 2(a). Note that there can be other nodes distributed at any other places on the line and the four nodes need not be consecutive. For the case $l_1 \leq l_2 + l_3$, we remove the cross by replacing the edge (a, b) with (a, c) and adding (c, b) (Figure 2(b)). Firstly, we check whether the newly added edges, (a, c) and (c, b), are valid which means their lengths do not exceed r_{max}. Since $|a - c| = l_1 < l_1 + l_2 = |a - b|$ and (a, b) is valid, (a, c) is also valid. Similarly, (c, b) is also valid. Secondly, there are 3 nodes, a, b and c, whose edges are changed. We check whether the changes potentially make them interfere with any new nodes. For a, one of its longer edges (a, b) is replaced with a shorter one (a, c), so a cannot interfere with more nodes in the new topology. A similar conclusion can be arrived at for b. As for the node c, we add a new edge (a, c) of length l_1 and (b, c) of length l_2. However, in both topologies, c already has an edge (c, d) of length $l_2 + l_3$. Since $l_2 + l_3 > l_2$ and $l_2 + l_3 \geq l_1$, the new edges will not make c interfere with any new nodes. Therefore, the topology in Figure 2(b) would not add to the interference on any nodes. Thirdly, since there are still paths to connect the nodes a, b and the nodes c, d, the new topology is connected as long as the topology in Figure 2(a) is connected. Further, since deleting an edge will not increase any interference, we can destroy any cycles in the new topology by deleting edges to form a spanning tree. Therefore, for the case $l_1 \leq l_2 + l_3$, we can remove the cross to construct a new spanning tree without adding to the interference on any nodes. Similarly, we can prove that the above is also true when $l_1 > l_2 + l_3$ as illustrated in Figure 2(c), and the theorem is proved. $\square$

According to the no-cross property, if there is already an edge (v_s, v_t), all the nodes inside the segment $\overline{v_s v_t}$ can be only adjacent to nodes located on the segment, but not to any other nodes on the line. (Two nodes are adjacent means they are neighbors.) However, it does not mean that interference of the nodes inside the segment is independent of the nodes outside. The nodes inside $\overline{v_s v_t}$ can interfere with the ones outside, and vice versa. This gives an important clue for us to design algorithms to minimize the average or the maximum interference which are described in the following sections.

3.2 Algorithms to Minimize the Average Interference

General Ideas. Based on the no-cross property, in the optimal spanning tree with minimum average interference, the nodes can be separated into segments. The nodes inside each segment are only adjacent to the other nodes on the same segment. However, as mentioned above, interference of the nodes inside a segment is still independent of the outside. Therefore, we do not compute the total interference by summing up the interference on each individual node, but the interference created by each node. Here, interference created by a node v with transmission radius r_v, $CI(v, r_v)$, is defined as the number of other nodes covered by the transmission range of v:

$$CI(v, r_v) = |\{u | u \in V/\{v\}, |u - v| \leq r_v\}|, \tag{4}$$

so that $\sum_{v \in V} CI(v, r_v) = \sum_{v \in V} RI(v)$. $CI(v, r_v)$ is only influenced by r_v, which is determined by the neighbors of v, and the locations of the other nodes. If all the nodes inside $\overline{v_s v_t}$ can only be adjacent to the nodes on it, the total interference created by the inside nodes will be independent of the topology of the outside nodes; and vice versa. Moreover, to compute the optimal spanning tree, we need to determine 1) how to divide the line into segments and 2) how to connect the nodes on each segment. Therefore, we can construct the optimal spanning tree based on dynamic programming as follows.

Algorithms. Two auxiliary functions are defined. The function $F(s,t)$[1], where $s < t$, is to compute the topology on $\overline{v_s v_t}$ so that the total interference created by the nodes inside $\overline{v_s v_t}$ is minimized with the following conditions satisfied:

1) the transmission radius of v_s is r_{v_s}.
2) the transmission radius of v_t is r_{v_t}.
3) all the nodes inside $\overline{v_s v_t}$ can be only adjacent to the ones on the segment $\overline{v_s v_t}$.
4) each node inside $\overline{v_s v_t}$ has a path either to v_s or to v_t.

The function $G(s,t)$, where $s < t$, is to compute the topology on $\overline{v_s v_t}$ so that the total interference created by the nodes inside $\overline{v_s v_t}$ is minimized with the following conditions satisfied:
$1), 2), 3)$ are the same as the first three conditions of $F(s,t)$.
4) all the nodes on $\overline{v_s v_t}$ are connected to each other directly or by nodes on $\overline{v_s v_t}$.

Both the functions F and G return the minimum total interference created by the nodes inside $\overline{v_s v_t}$. If $+\infty$ is returned, it means there is no such a topology to satisfy all the conditions. Comparing the fourth conditions, for function F, to achieve connectivity among all the nodes, we actually assume there is already a path from v_s to v_t before adding any edges to the nodes inside $\overline{v_s v_t}$. For G, there is no such a path.

For a node v, the set of its potential neighbors, $N(v)$, are the nodes covered by v's maximum transmission range:

$$N(v) = \{u | u \in V/\{v\}, |u - v| \leq r_{max}\}. \tag{5}$$

Recall that the transmission radius of v is the distance to its farthest neighbors. So, the set of its potential transmission radii, $R(v)$, is

$$R(v) = \{|u - v| | u \in N(v)\}, \tag{6}$$

and $|R(v)| \leq |N(v)| \leq \Delta$. If v can only be adjacent to a subset nodes S, its potential neighbors $N(v,S)$ and its potential transmission radii $R(v,S)$ are $N(v,S) = N(v) \bigcap S$ and $R(v,S) = \{|u - v| | u \in N(v,S)\}$ respectively. To compute the functions F and G, we calculate and store each $CI(v, r_v)$ in an $n \times \Delta$ array. For $F(s,t)$, the boundary condition is there are no nodes inside $\overline{v_s v_t}$. For the other cases, to satisfy the condition 4), there must be at least one node v_m inside $\overline{v_s v_t}$ that is adjacent to v_p, where $v_p = v_s$ or v_t. Without loss of generality, we set $v_p = v_s$. Since v_s and v_m are connected as well as v_s and v_t in the assumption, there is already a path from v_m to v_t. Therefore, $F(s,t)$ consist of three parts, $F(s,m)$, $F(m,t)$ and $CI(m, r_{v_m})$. We can enumerate v_m and r_{v_m}, so F can be computed in Algorithm [1]. In line 1, we first check the boundary

[1] For conciseness, we use $F(s,t)$ to stand for $F(v_s, v_t, s, t, r_{v_s}, r_{v_t})$, and $G(s,t)$ to stand for $G(v_s, v_t, s, t, r_{v_s}, r_{v_t})$.

condition. The set S is defined to store the nodes on $\overline{v_s v_t}$ in line 2. Lines 3–10 are to compute the minimum interference created by the nodes inside $\overline{v_s v_t}$ recursively with the four conditions satisfied. As v_m can only be adjacent to the nodes on $\overline{v_s v_t}$, its potential transmission radii are defined as $R(v_m, S)$ in line 4. In line 7 we assume adding an edge (v_m, v_p), and line 8 is to compute $F(s, t)$.

Algorithm 1. Compute $F(s, t)$

```
1. if $s + 1 = t$ then    return $F = 0$    /* the boundary condition*/
2. $F = +\infty$    $S = \{v_s, v_{s+1}, ..., v_t\}$
3. for each $v_m \in S/\{v_s, v_t\}$ do
4.   $R(v_m, S) = \{|u - v_m| | u \in N(v_m) \bigcap S\}$
5.   for each $v_p \in \{v_s, v_t\}$ do
6.     for each $r_{v_m} \in R(v_m, S)$ do
7.       if $|v_p - v_m| \leq min(r_{v_m}, r_{v_p})$ then   /* assume adding an edge $(v_m, v_p)$ */
8.       $F = min(F, F(s, m) + F(m, t) + CI(m, r_{v_m}))$
9. return $F$
```

As for the function $G(s, t)$, in order to satisfy condition 4), there are two choices. One is that v_s is directly connected to v_t, such that $G(s, t) = F(s, t)$. The other is v_s and v_t are connected by some other nodes inside $\overline{v_s v_t}$. Then, there must be at least one node v_m inside $\overline{v_s v_t}$ which is adjacent to v_s, and $G(s, t)$ can consist of $F(s, m)$, $G(m, t)$, and $CI(m, r_{v_m})$. Similar to Algorithm 1, $G(s, t)$ can be computed in Algorithm 2.

Algorithm 2. Compute $G(s, t)$

```
1. $G = +\infty$
2. if $|v_s - v_t| \leq min(r_{v_s}, r_{v_t})$ then    /* assume adding an edge $(v_s, v_t)$ */
3.   $G = F(s, t)$
4. $S = \{v_s, v_{s+1}, ..., v_t\}$
5. for each $v_m \in S/\{v_s, v_t\}$ do
6.   $R(v_m, S) = \{|u - v_m| | u \in N(v_m) \bigcap S\}$
7.     for each $r_{v_m} \in R(v_m, S)$ do
8.       if $|v_s - v_m| \leq min(r_{v_m}, r_{v_s})$ then   /* assume adding an edge $(v_s, v_m)$ */
9.       $G = min(G, F(s, m) + G(m, t) + CI(m, r_{v_m}))$
10. return $G$
```

With F and G, the minimum average interference can be computed in Algorithm 3 by calling $G(0, n - 1)$.

Algorithm 3. Compute the minimum average interference

```
1. $total = +\infty$
2. for each $r_{v_0} \in R(v_0)$ do
3.   for each $r_{v_{n-1}} \in R(v_{n-1})$ do
4.   $total = min(total,$
5.     $CI(v_0, r_{v_0}) + CI(v_{n-1}, r_{v_{n-1}}) + G(0, n - 1))$
6. return $\frac{total}{n}$
```

When computing the minimum average interference, we record v_m and r_{v_m} for each function $G(s, t)$, and v_p, v_m and r_{v_m} for each function $F(s, t)$. Through tracing backwards, we can construct a connected topology of $n-1$ edges with the minimum average interference, which is the optimal spanning tree. For conciseness, we omit the traceback function here. The correctness of the above algorithms are verified through comparing our results with the outputs generated by the brute-force search, which runs slowly in the exponential time $O(n^{\Delta})$. Figure 3 gives an example of an optimal spanning tree for the 6-node exponential chain. In our method, time is mainly spent on computing functions F and G. Since the number of possible transmission radii of a node can not exceed Δ, the time complexity to compute the optimal spanning tree with minimum average interference is $O(n^3\Delta^3)$.

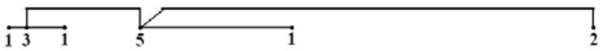


Fig. 3. The spanning tree for the 6-node exponential chain with minimum average interference $\frac{13}{6}$: the numbers next to each node is interference it creates

4 Minimizing the Maximum Interference

4.1 General Ideas

For the n nodes, $V = \{v_1, v_2, ..., v_{n-1}\}$, the minimum maximum node interference in all the possible spanning trees is denoted as k, where $k \leq \Delta \leq n-1$ since all the nodes have the same maximum transmission radius r_{max}. In this section, we first design an algorithm to check whether there is a spanning tree with the maximum interference no larger than k set from 1 to $n-1$. After computing k, we can construct the optimal tree with such a maximum interference by traceback.

For a segment $\overline{v_s v_t}$, even when the nodes inside are not allowed to be adjacent to the ones outside, they still interfere with the outside nodes. We record all the interference from the nodes on $\overline{v_s v_t}$ to the outside nodes as a set $C(v_s, v_t, k)$, where $s \leq k$. Each element $c(v_s, v_t, k) \in C(v_s, v_t, k)$, called *a skeleton* of the topologies on $\overline{v_s v_t}$, stores the following nodes and their transmission radii:

1) if $s > 0$ and $t < n-1$: the nodes on $\overline{v_s v_t}$ that interfere with v_{s-1} or v_{t+1};
2) if $s = 0$ and $t < n-1$: the nodes on $\overline{v_s v_t}$ that interfere with v_{t+1};
3) if $s > 0$ and $t = n-1$: the nodes on $\overline{v_s v_t}$ that interfere with v_{s-1};
4) if $s = 0$ and $t = n-1$: meaningless.

Specifically, $C(v, v, k)$ has $|R(v)|$ elements that store the node v and its potential transmission radii in $R(v)$. Since there must be no more than k nodes on $\overline{v_s v_t}$ that interfere with the left or the right nodes outside respectively, we call a skeleton $c(v_s, v_t, k)$ *valid* if and only if there are no more than k nodes in it that interfere with the first node left or right to $\overline{v_s v_t}$ respectively. Figure 4 gives an example of a valid skeleton $c(v_s, v_t, 3)$ and two different topologies built according to the skeleton on $\overline{v_s v_t}$, where only v_s and v_{s+2} interfere with v_{s-1}, and only

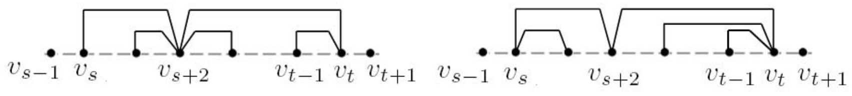


Fig. 4. The skeleton $c(v_s, v_t, 3) = \{(v_s, r_{v_s} = |v_s - v_{s+2}|), (v_{s+2}, r_{v_{s+2}} = |v_t - v_{s+2}|), (v_t, r_{v_t} = |v_t - v_{s+2}|)\}$ on the segment $\overline{v_s v_t}$. (a) and (b) are two possible topologies computed according to $c(v_s, v_t, 3)$

v_t interferes with v_{t+1}. Note that a valid skeleton does not guarantee that the maximum interference in the whole topology would not exceed the maximum, such as $RI(v_{s+2}) = 4 > 3$ in Figure 4(a).

Further, given $c(v_0, v_s, k)$, $c(v_s, v_t, k)$ and $c(v_t, v_{n-1}, k)$, to compute the topology on $\overline{v_s v_t}$, the following two requirements need to be satisfied: 1) together with the interference from nodes in $c(v_s, v_t, k)$, each node outside $\overline{v_s v_t}$ can not be interfered with more than k nodes; and 2), together with interference from nodes in $c(v_0, v_s, k)$ and $c(v_t, v_{n-1}, k)$, each node on $\overline{v_s v_t}$ can not be interfered with more than k nodes. Considering the mutual interference among the nodes on or outside each segment, we can design an algorithm to check whether there is a spanning tree with maximum interference no greater than k by dynamic programming as follows.

4.2 Algorithms

First of all, we define a function $Merge(c(v_{p_1}, v_{p_2}, k), c(v_{p_2+1}, v_{p_3}, k), ..., c(v_{p_{m-1}}, v_{p_m}, k))$, where $0 \le p_1 \le p_2 \le ... \le p_m \le n-1$, to merge the skeletons on the consecutive segments and return $c(v_{p_1}, v_{p_m}, k)$. The method is to check every node in the skeletons whether to interfere with the first node left or right to $\overline{v_{p_1} v_{p_m}}$. Note that after merging, the new skeleton $c(v_{p_1}, v_{p_m}, k)$ may not be valid. Similar to compute the average interference, here we define two auxiliary boolean functions. The function *boolean* $F^*(s, t, k)$[2], where $s < t$, is to check whether there is a topology on $\overline{v_s v_t}$ that satisfies the following conditions simultaneously:

1) the transmission radius of v_s is r_{v_s}.
2) the transmission radius of v_t is r_{v_t}.
3) all the nodes inside $\overline{v_s v_t}$ can be only adjacent to the ones on $\overline{v_s v_t}$.
4) the skeleton for $\overline{v_{s+1} v_{t-1}}$ is $c(v_{s+1}, v_{t-1}, k)$.
5) $RI(v) \le k$, for each v inside $\overline{v_{s+1} v_{t-1}}$.
6) each node inside $\overline{v_s v_t}$ have a path either to v_s or to v_t.

Similarly, the function *boolean* $G^*(s, t, k)$, where $s < t$, is to check whether there is a topology on $\overline{v_s v_t}$ that satisfies the following conditions simultaneously: $1), 2), 3), 4)$ and 5) are the same as the first five conditions for $F^*(s, t, k)$. 6) all the nodes on $\overline{v_s v_t}$ are connected to each other directly or by nodes on $\overline{v_s v_t}$.

[2] For conciseness, we use $F^*(s, t, k)$ to stand for $F^*(v_s, v_t, s, t, r_{v_s}, r_{v_t}, c(v_0, v_s, k), c(v_{s+1}, v_{t-1}, k), c(v_t, v_{n-1}, k))$, and $G^*(s, t, k)$ to stand for $G^*(v_s, v_t, s, t, r_{v_s}, r_{v_t}, c(v_0, v_s, k), c(v_{s+1}, v_{t-1}, k), c(v_t, v_{n-1}, k))$.

For $F^*(s,t,k)$, we still assume that there has been a path from v_s to v_t before adding any edges to the nodes inside $\overline{v_s v_t}$. When $s < t-1$, to satisfy the condition 6), there must be a node v_m inside $\overline{v_s v_t}$ that is adjacent to v_p, where $v_p = v_s$ or v_t. Therefore, there is a path from v_s to v_m as well as from v_m to v_t. With ensuring $RI(v_m) \leq k$, $F^*(s,t,k)$ is divided to check $F^*(s,m,k)$ and $F^*(m,t,k)$. So it can be computed in Algorithm 4. Lines 3–15 are to compute $F^*(s,t,k)$ recursively. In line 7, we assume adding an edge (v_m, v_p). Line 8 and 9 enumerate the possible skeletons on $\overline{v_{s+1} v_{m-1}}$ and $\overline{v_{m+1} v_{t-1}}$, and line 10 is to ensure the condition 4) is satisfied. In line 11, $c(v_0, v_m, k) = Merge(c(v_0, v_s, k), c(v_{s+1}, v_{m-1}, k), c(v_m, v_m, k))$ and $c(v_m, v_{n-1}, k) = Merge(c(v_m, v_m, k), c(v_{m+1}, v_{t-1}, k), c(v_t, v_{n-1}, k))$, and line 12 is to check their validity. All the three components for $F^*(s,m,k)$ are checked in line 13, and line 16 returns the value of $F^*(s,t,k)$.

Algorithm 4. Compute *boolean* $F^*(s,t,k)$

```
1.  if $s+1=t$ then   return $F^* = true$      /* the boundary condition*/
2.  $S = \{v_s, v_{s+1}, ..., v_t\}$
3.  for each $v_m \in S/\{v_s, v_t\}$   do
4.      $R(v_m, S) = \{|u - v_m| | u \in N(v_m) \bigcap S\}$
5.      for each $v_p \in \{v_s, v_t\}$ do
6.          for each $r_{v_m} \in R(v_m, S)$ do
7.              if $|v_p - v_m| \leq min(r_{v_m}, r_{v_p})$ then   /* assume adding an edge $(v_m, v_p)$ */
8.                  for each $c(v_{s+1}, v_{m-1}, k) \in C(v_{s+1}, v_{m-1}, k)$   do
9.                      for each $c(v_{m+1}, v_{t-1}, k) \in C(v_{m+1}, v_{t-1}, k)$   do
10.                         if   $Merge(c(v_{s+1}, v_{m-1}, k), c(v_m, v_m, k), c(v_{m+1}, v_{t-1}, k)) = c(v_{s+1}, v_{t-1}, k)$ then
11.                             compute $c(v_0, v_m, k)$ and $c(v_m, v_{n-1}, k)$ by merging
12.                             if $c(v_0, v_m, k)$ is valid && $c(v_m, v_{n-1}, k)$ is valid
13.                             && no more than $k$ nodes in $c(v_0, v_s, k)$, $c(v_{s+1}, v_{m-1}, k)$, $c(v_{m+1}, v_{t-1}, k)$ and $c(v_t, v_{n-1}, k)$ that interfere with $v_m$
14.                             && $F^*(s,m,k)$ && $F^*(m,t,k)$ then
15.                                 return $F^* = true$
16. return $F^* = false$   /* no topology on $\overline{v_s v_t}$ to satisfy the 6 conditions*/
```

To compute $G^*(s,t,k)$, we actually assume there is no path from v_s to v_t before adding edges to the nodes inside $\overline{v_s v_t}$. In order to satisfy the condition 6), there must be a node v_m inside $\overline{v_s v_t}$ that is adjacent to v_s. Similarly, with ensuring $RI(v_m) \leq k$, $G^*(s,t,k)$ is divided to check $F^*(s,m,k)$ and $G^*(m,t,k)$. Algorithm 5 gives a detailed description on how to compute $G^*(s,t,k)$.

By calling $F^*(s,t,k)$ and $G^*(s,t,k)$, we design the main function, $FindMinMax(V)$, to find the minimum maximum interference k. From 1 to $n-1$. We check and return k immediately when a spanning tree with the maximum interference of k is found. Algorithm 6 illustrates how to compute the function $FindMinMax(V)$ by calling $G^*(0, n-1, k)$, where we only consider the cases when $|V| = n > 2$.

After computing $FindMinMax(V)$, we can do traceback and construct the optimal spanning tree with the minimum maximum interference by adding exactly $n-1$ edges. To save space, we omit the traceback function here.

Algorithm 5. Compute *boolean* $G^*(s, t, k)$

```
1.  if$|v_s - v_t| \le min(r_{v_s}, r_{v_t})$ && $F^*(s,t,k) = true$ then
2.      return $G^* = true$        /* assume adding an edge $(v_s, v_t)$ */
3.  $S = \{v_s, v_{s+1}, ..., v_t\}$
4.  for each $v_m \in S/\{v_s, v_t\}$ do
5.  $R(v_m, S) = \{|u - v_m||u \in N(v_m) \bigcap S\}$
6.      for each $r_{v_m} \in R(v_m, S)$ do
7.          if $|v_s - v_m| \le min(r_{v_m}, r_{v_s})$ then        /* assume adding an edge $(v_s, v_m)$ */
8.              for each $c(v_{s+1}, v_{m-1}, k) \in C(v_{s+1}, v_{m-1}, k)$ do
9.                  for each $c(v_{m+1}, v_{t-1}, k) \in C(v_{m+1}, v_{t-1}, k)$ do
10.                     if $Merge(c(v_{s+1}, v_{m-1}, k), c(v_m, v_m, k), c(v_{m+1}, v_{t-1}, k)) =$
    $c(v_{s+1}, v_{t-1}, k)$ then
11.                         compute $c(v_0, v_m, k)$ and $c(v_m, v_{n-1}, k)$ by merging
12.                         if $c(v_0, v_m, k)$ is valid && $c(v_m, v_{n-1}, k)$ is valid
13.                         && no more than $k$ nodes in $c(v_0, v_s, k)$, $c(v_{s+1}, v_{m-1}, k)$,
    $c(v_{m+1}, v_{t-1}, k)$ and $c(v_t, v_{n-1}, k)$ that interfere with $v_m$
14.                         && $F^*(s, m, k)$ && $G^*(m, t, k)$ then
15.                             return $G^* = true$
16. return $G^* = false$    /* no topology on $\overline{v_s v_t}$ to satisfy the 6 conditions*/
```

Algorithm 6. Compute $FindMinMax(V)$, and return the minimum maximum interference

```
1.  $k = 1$
2.  while $k \le n - 1$ do
3.      for each $r_{v_0} \in R(v_0)$ do
4.          for each $r_{v_{n-1}} \in R(v_{n-1})$ do
5.              for each $c(v_1, v_{n-2}, k) \in C(v_1, v_{n-2}, k)$
6.                  if no more than $k$ nodes in $c(v_1, v_{n-2}, k)$ and $v_{n-1}$ that interfere with $v_0$
7.                  && no more than $k$ nodes in $c(v_1, v_{n-2}, k)$ and $v_0$ that interfere with
8.                  $v_{n-1}$ && $G^*(0, n-1, k)$ then
9.                      return k
10.     k=k+1
11. End While
12. return $+\infty$    / *no connected topology on $V$ with the constraint of $r_{max}$ */
```

4.3 Analysis

Correctness. The method has been verified through comparing our optimal topologies with the outputs generated by the brute-force search running in time $O(n^{\Delta})$. Moreover, our algorithms can find all the topologies of minimum maximum interference without crosses. For example, we can find all the 17 optimal topologies without a cross for the 6-node chain (Figure 5).

Time Complexity. Firstly, we analyze the size of the set $C(v_s, v_t, k)$ when $s > 0$ and $t < n-1$. The nodes in each element $c(v_s, v_t, k)$ can be divided as the node sets CL and CR which contain the nodes on $\overline{v_s v_t}$ that interfere with v_{s-1} and v_{t+1} respectively. As the maximum interference is k, we get $|CL| \le k \le \Delta$. Each node has different transmission radii of Δ as many as possible. Thus, the number of combinations for the nodes in CL and their transmission radii is

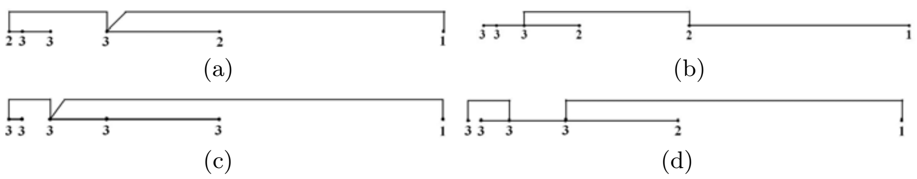


Fig. 5. 4 different optimal spanning trees for the 6-node exponential chain with the minimum maximum interference

$$\binom{\Delta}{0} + \binom{\Delta}{1} \times \Delta + ... + \binom{\Delta}{k} \times \Delta^k = O(\Delta^{2k}). \tag{7}$$

A similar result can be obtained for CR. Therefore, the size of $C(v_s, v_t, k)$ is $O(\Delta^{4k})$. Further, the total amount of the functions $F^*(s,t,k)$ and $G^*(s,t,k)$ is $O(n^2\Delta^{O(k)})$. For each function, the computing time is $O(n\Delta^{O(k)})$. As there are no functions being repeatedly computed, the time to finish $FindMinMax(V)$ will be $O(n^3\Delta^{O(k)})$. To construct the optimal spanning tree, the main time is to compute k by $FindMinMax(V)$. Thus, the time complexity to construct the spanning tree with the minimum maximum interference is $O(n^3\Delta^{O(k)})$. Since $\Delta \leq n-1$ and $k = O(\sqrt{\Delta})$ [14], the time is sub-exponential. However, when Δ is small, which means a low maximum node degree, our algorithm is fast.

Space Complexity. The space is mainly for storing the functions F^* and G^* as well as the sets $C(v_s, v_t, k)$. Therefore, the space complexity is $O(n^2\Delta^{O(k)})$.

5 Conclusion

In this paper, we study the problem to minimize the receiver-centric interference for the highway model. Based on the no-cross property and using dynamic programming, the first polynomial-time exact algorithm for constructing a connected topology with minimum average interference is proposed. We give also the first sub-exponential-time exact algorithm for constructing a connected topology while minimizing the maximum interference. The question of whether it is NP-hard to minimize the maximum interference for the highway model is still open. Related open problems include how to design an approximation with a ratio better than $O(\sqrt[4]{\Delta})$ for the highway model, how to design efficient approximations to minimize the maximum interference in 2D networks, and how to combine the interference minimization with other network properties.

References

1. Banner, R., Orda, A.: Multi-Objective Topology Control in Wireless Networks. In: INFOCOM 2008, pp. 448–456 (2008)
2. Buchin, K.: Minimizing the Maximum Interference is Hard, arXiv: 0802.2134v1 (2008)

3. Burkhart, M., von Rickenbach, P., Wattenhofer, R., Zollinger, A.: Does Topology Control Reduce Interference? In: MobiHoc 2004, pp. 9–19 (2004)
4. Moaveni-Nejad, K., Li, X.-Y.: Low-interference topology control for wireless ad hoc networks. Int. J. of Ad Hoc & Sensor Wireless Networks 1, 41–64 (2005)
5. Wu, K.-D., Liao, W.: On Constructing Low Interference Topology in Multihop Wireless Networks. Int. J. of Sensor Networks 2, 321–330 (2007)
6. Johansson, T., Carr-Motyčková, L.: Reducing Interference in Ad Hoc Networks Through Topology Control. In: DIALM-POMC 2005, pp. 17–23 (2005)
7. Benkert, M., Gudmundsson, J., Haverkort, H., Wolff, A.: Constructing minimum-interference networks. Comp. Geometry 40(3), 179–194 (2008)
8. Damian, M., Javali, N.: Distributed construction of bounded-degree low-interference spanners of low weight. In: MobiHoc 2008, pp. 101–110 (2008)
9. Halldórsson, M., Tokuyama, T.: Minimizing interference of a wireless ad hoc network in a plane. In: Nikoletseas, S.E., Rolim, J.D.P. (eds.) ALGOSENSORS 2006. LNCS, vol. 4240, pp. 71–82. Springer, Heidelberg (2006)
10. Li, X.Y., Song, W.Z., Wang, W.: A Unified Energy Efficient Topology for Unicast and Broadcast. In: MOBICOM 2005, pp. 1–15 (2005)
11. Locher, T., von Rickenbach, P., Wattenhofer, R.: Sensor networks continue to puzzle: Selected open problems. In: Rao, S., Chatterjee, M., Jayanti, P., Murthy, C.S.R., Saha, S.K. (eds.) ICDCN 2008. LNCS, vol. 4904, pp. 25–38. Springer, Heidelberg (2008)
12. Moscibroda, T., Wattenhofer, R.: Minimizing interference in ad hoc and sensor networks. In: DIALM-POMC 2005, pp. 24–33 (2005)
13. Santi, P.: Topology Control in Wireless Ad Hoc and Sensor Networks. Wiley, Chichester (2005)
14. von Rickenbach, P., Schmid, S., Wattenhofer, R., Zollinger, A.: A robust interference model for wireless ad hoc networks. In: WMAN 2005 (2005)
15. von Rickenbach, P., Wattenhofer, R., Zollinger, A.: Algorithmic Models of Interference in Wireless Ad Hoc and Sensor Networks. IEEE/ACM Trans. on Net (TON) 17(1), 172–185 (2009)

Join-Queries between Two Spatial Datasets Indexed by a Single R*-Tree

Michael Vassilakopoulos[1], Antonio Corral[2,⋆], and Nikitas N. Karanikolas[3]

[1] Dept. of Computer Science and Biomedical Informatics, University of Central Greece, 2-4 Papasiopoulou st., 35100 Lamia, Greece
mvasilako@ucg.gr

[2] Dept. of Languages and Computing, University of Almeria, 04120 Almeria, Spain
acorral@ual.es

[3] Dept. of Informatics, Technological Educational Institute of Athens, Ag. Spyridonos st., 12210 Aigaleo, Greece
nnk@teiath.gr

Abstract. A spatial join, a common query in Spatial Databases and Geographical Information Systems (GIS), consists in testing every possible pair of data elements belonging to two spatial datasets against a spatial predicate. This predicate might be "intersects", "contains", "is enclosed by", "distance", "northwest", "adjacent", "meets", etc. The large size of datasets that appears in industrial and commercial modern applications (e.g. GIS applications, where multiple instances of the datasets are kept) raises the cost of join processing and the importance of the choice of the data indexing method and the query processing technique. The family of R-trees is considered a good choice (especially the R*-tree) for indexing a spatial dataset. When joining two datasets, a common assumption is that each dataset is indexed by a different R*-tree and the join is processed by a synchronous traversal of the two trees. In this paper, we assume that both datasets are indexed by a single R*-tree, so that spatial locality between different datasets is embedded in data indexing, facilitating the evaluation of join queries between the two datasets. We experimentally compare the I/O and Response Time performance of join queries, using this single tree indexing approach against the usual approach of indexing each dataset by a different tree.

Keywords: Spatial Access Methods, R-trees, Query Processing, Joins.

1 Introduction

A Spatial Database is a database that offers spatial data types (for example, types for points, line segments, regions, etc.), a query language with spatial predicates, spatial indexing techniques and efficient processing of spatial queries.

⋆ Supported by the De Qatris iManager a vManager. Estrategias para el almacenamiento y busqueda en colecciones de documentos de video digital project (TIN2008-03063), funded by the Spanish Ministry of Science and Technology.

Among the most frequent queries appearing in spatial databases is the spatial join query: given two collections R and S of spatial objects and a spatial predicate θ, find all pairs of objects (O, O') $\in$ R$\times$S, where θ(O.G, O'.G) evaluates to true (O.G represents the spatial extent of spatial object O). Some examples of the spatial predicate θ are: *intersects, contains, is_enclosed_by, distance, north-west, adjacent, meets*, etc. For spatial predicates such as *contains, encloses*, or *adjacent*, for example, the intersection join is an efficient filter that yields a set of candidate solutions typically much smaller than the Cartesian product R$\times$S. The popularization of technology, as well as, the advancements of the computing and telecommunication devices and infrastructure contribute to the production of large datasets (e.g. datasets of Geographical Information Systems applications, where multiple instances of the positions of mobile phone users are kept, or datasets of traffic control systems). The size of such datasets raises the cost of join processing and, as a consequence, the importance of the choice of the data indexing method and the query processing technique.

The multidimensional access methods belonging to the R-tree family (the R*-tree [1] being the most popular one) are considered good choices for indexing spatial data sets in order to process join queries in Spatial Databases. This is accomplished by branch-and-bound algorithms that employ spatial predicates and pruning heuristics based on MBRs (Minimum Bounding Rectangles), in order to reduce the search space.

When joining two datasets, a common assumption is that each dataset is indexed by a different R*-tree and the join is processed by a synchronous traversal of the two trees. In this paper, we assume that both datasets are indexed by a single R*-tree, so that spatial locality between different datasets is embedded in data indexing, facilitating the evaluation of join queries between the two datasets. We experimentally compare the I/O and Response Time performance of *Intersection Join, K Closest Pair Query (K-CPQ)* and *Buffer Query* (see Section 2), using this single tree indexing approach against the usual approach of indexing each dataset by a different tree.

The contributions of this paper consist in the following:

1. We present an R*-tree variation able to index two datasets simultaneously, taking advantage of the spatial locality between the different datasets.
2. We present a new algorithm for processing join queries on the two datasets R*-tree variation by Breadth-First traversal, where at each level we follow Best-First selection.
3. We present results of extensive comparative experimentation between a pair of one dataset R*-trees and a two dataset R*-tree, when processing *Intersection Join* queries (on non-point datasets), *K-CPQ* and *Buffer Queries* (on point datasets) and conclude about the winner structure, in each case, in terms of disk accesses and response time.

The paper is organized as follows. In Section 2, we review the related literature and motivate the research reported here. In Section 3, a brief description of the R*-tree indexing one dataset is presented. In Section 4, the R*-tree variation indexing two datasets are presented. In Section 5, we present the algorithms that perform the join processing in the two types of R*-trees. In Section 6, a comparative performance study is reported. Finally, in Section 7, conclusions on the contribution of this paper and future work are summarized.

2 Related Work and Motivation

There are numerous papers that study processing of join queries using R-trees. Some of the most characteristic ones are mentioned in the following. Recently, an exhaustive analysis of several techniques used to perform a spatial join taking into account a filter-and-refinement approach has been published in [10]. We can classify the spatial join methods depending on whether the sets of spatial objects involved in the query are indexed or not. When both sets are indexed, the most influential and known algorithm for joining two datasets indexed by R*-trees was presented in [2], where several techniques to improve both CPU and I/O time have been studied. This algorithm follows a Depth-First synchronized tree traversal order. A Breadth-First synchronized tree traversal version to reduce I/O cost was presented in [9].

An *Intersection Join* involves two spatial data sets and discovers the pairs of objects from the two input datasets that intersect each other. An example of this spatial join query is to "find all trails that go through some forest", where the two spatial datasets are trails and forests, and the spatial predicate is *intersects*.

The *K Closest Pairs Query (K-CPQ)*, where each dataset is stored in an R-tree is studied thoroughly in [5,6]. The *K-CPQ* discovers the K pairs of data elements formed from two datasets that have the K smallest distances between them. The *K-CPQ* is a combination of join and nearest neighbor queries. Like a join query, all pairs of objects are candidates for the result. Like a nearest neighbor query, proximity forms the basis for the final ordering.

A class of commonly asked queries in a spatial database is known as *Buffer Queries* and they are studied in [3]. An example of such a query is to "find house-power line pairs that are within 50 meters of each other". A *Buffer Query* involves two spatial data sets and a distance threshold ρ. The answer to this query consists of pairs of objects from the two input sets that are within distance ρ of each other.

A specialized index for speeding up processing of joins in spatial databases has been presented in [15]. This is called a join-index. Join-indices use pre-computation techniques to speed up online query processing and are useful for datasets which are updated infrequently. The I/O cost of join computation using a join-index with limited buffer space depends primarily on the page access sequence used to fetch the pages of the base relations. In [15], methods based on clustering to compute the joins are presented.

The idea of using one data structure for two datasets has been presented in [16], where the Multi-Layer Quadtree (ML-Quadtree), a data structure that allows the storage and processing of several layers at the same time, is proposed. This structure is based on the PM-Quadtree (Polygonal Map Quadtree, a structure for storing points and edges), which allows the storage of only a single layer map. The aim of the ML-Quadtree is to be able to manage, store and perform queries among multiple layers simultaneously.

In this paper, we examine the effect of storing both datasets in a single R*-tree (so that spatial locality between different datasets is inherent in data indexing) on join processing (*Intersection Join, K-CPQ* and *Buffer Query*).

3 Background

R-trees [7,8] are hierarchical, height balanced multidimensional data structures, designed for use in secondary storage, and are generalizations of B-trees [4] for multidimensional data spaces. They are used for indexing or both for indexing and data storage of d-dimensional objects represented by their Minimum Bounding d-dimensional hyper-Rectangle (MBRs). An MBR is determined by two d-dimensional points that belong to its faces, one that has the d minimum and one that has the d maximum coordinates (these are the endpoints of one of the diagonals of the MBR). Each R-tree node corresponds to the MBR that contains its children. The tree leaves contain pointers to the objects of the database, instead of pointers to children nodes. The nodes are implemented as disk pages. The rules obeyed by an R-tree are as follows: (1) Leaves reside on the same level. (2) Each leaf node contains entries of the form (MBR, Oid), such that MBR is the minimum bounding rectangle that encloses the spatial object determined by the identifier Oid. (3) Every other node (internal) contains entries of the form (MBR, Addr), where Addr is the address of the child node and MBR is the minimum bounding rectangle that encloses MBRs of all entries in that child node. (4) An R-tree of class (m, M) has the characteristic that every node, except possibly for the root, contains between m and M entries, where m $\leq \lceil$ M/2 $\rceil$ (M and m are also called maximum and minimum branching factor or fan-out). The root contains at least two entries, if it is not a leaf.

Like other tree-like index structures, an R-tree index partitions the multidimensional space by grouping objects in a hierarchical manner. A subspace occupied by a tree node in an R-tree is always contained in the subspace of its parent node, i.e. the *MBR enclosure property.* According to this property, an MBR of an R-tree node (at any level, except at the leaf level) always encloses the MBRs of its descendent R-tree nodes. This characteristic of spatial containment between MBRs of R-tree nodes is commonly used by spatial join algorithms as well as distance-based query algorithms.

Many variations of R-trees have appeared in the literature (exhaustive surveys can be found in [7,12,13]). One of the most popular and efficient variations is the R*tree [1]. The R*-tree added two major enhancements to the original R-tree, when a node overflow is caused. First, rather than just considering the area, the node-splitting algorithm in R*-trees also minimizes the perimeter and overlap enlargement of the MBRs. Second, an over flown node is not split immediately, but a portion of entries of the node is reinserted from the top of the R*-tree (forced reinsertion) [1]. In the rest of this paper, we use R*-trees.

4 The R*-Tree Indexing Two Spatial Datasets

So far in the literature, the option to index two datasets using R*-trees was to use a separate (isolated) tree for each dataset. In the rest of the paper, we call this option 2D2T (2 Datasets in 2 Trees).

In order to examine the effect of storing both datasets in a single (common to both datasets) R*-tree on join processing, an option called 2D1T (2 Datasets in 1 Tree), we use one common R*-tree for datasets D1 and D2, where there is no distinction between D1 data and D2 data during insertions. This means that each datum to be inserted in the tree (point, or object) has a flag at each MBR (so-called *dataset*) showing if it belongs to D1 (flag 'a') or D2 (flag 'b'), or both of them (flag 'c'), but this flag does not affect the placement of this datum in the R*-tree. For example, in Figure 1 a 2D1T R*-tree is depicted (for simplicity, only the *dataset* flag of each MBR and not the MBR itself is depicted). At internal nodes, flag 'a' ('b') determines that the internal node is the root of a subtree with data elements belonging to dataset D1 (D2), only. Flag 'c' determines that the internal node is the root of a subtree with some data elements belonging to dataset D1 and some data elements belonging to dataset D2. At leaf nodes, flag 'a' ('b') determines a data element belonging to dataset D1 (D2), while the numeric subscript of 'a' or 'b' is the identifier of the data element, or pointer to the detailed geometry of the data element.

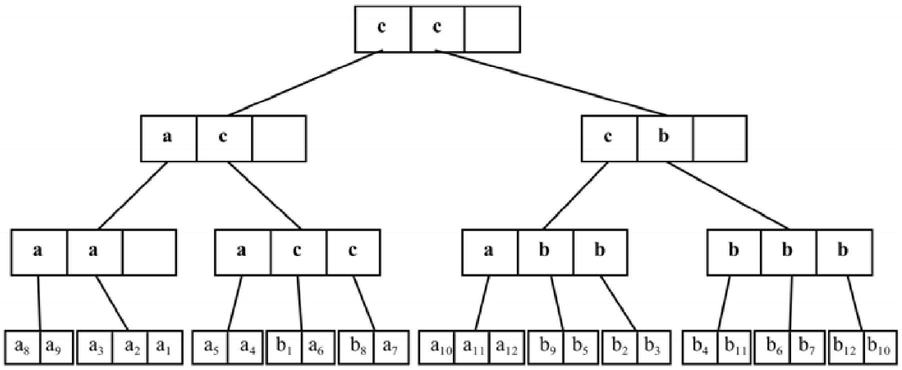


Fig. 1. An example of a 2D1T

5 Algorithms for Processing Join-Queries

In the algorithms that are presented in the following, plane-sweep, a common technique for computing intersections [14], is utilized to save CPU cost. The basic idea is to move a line, the so-called sweep-line, perpendicular to one of the dimensions, e.g. X-axis, from left to right. We apply this technique for restricting all possible combinations of pairs of MBRs, taking into account the minimum distance between MBRs (or whether both MBRs intersect) and excluding from consideration several pairs. If this technique is not used, all possible combinations of pairs of MBRs or spatial objects that result at every step of an algorithm must be processed with quadratic cost. For distance join queries (*K-CPQ and Buffer Query*) a variant of this technique has been developed in [5,6], and it is called distance-based plane-sweep.

When the R*-trees storing the two spatial datasets have different heights, the algorithms are slightly more complicated. There are two approaches for treating different heights: fix-at-leaves and fix-at-roots [5,6] and one of them can be adopted. In our implementations we are used fix-at-leaves.

It is well known that search algorithms (Best-First, Depth-First and Breadth-First Search) can be applied on query processing over tree-like structures in spatial databases, e.g. spatial query algorithms over R-trees [12]. We have implemented all of them for all the three spatial join queries (*Intersection Join, K-CPQ* and *Buffer Query*) that we study on 2D2T and 2D1T (in 2D1T, since the tree is only one, Self-Join variations of the algorithms are utilized). Due to limited space, we review only one search algorithm for each spatial join query.

For processing *Intersection Join* in 2D2T, the Breadth-First Traversal algorithm works as follows [9]. BFT synchronously traverses both R-trees in breadth-first order while processing join computation one level at a time. At each level, BFT creates an intermediate join index (IJI) based on the intermediate join results and deploys global optimization strategies (e.g. orderings) to improve the join computation at the next level.

For processing *K-CPQ* in 2D2T, the Depth-First Traversal algorithm works as follows [5,6]. The DFT algorithm (K=1) visits the roots of the two R*-trees and recursively follow the pair of entries, whose *mindist* (minimum distance between two MBRs) is the minimum among all pairs. The process is repeated recursively until the leaf levels are reached, where a potential closest pair is found. During backtracking to the upper levels, the algorithm only visits pairs of entries whose *mindist* is smaller than the distance of the closest pair found so far.

For processing the *Buffer Query* in 2D2T, the Best-First Traversal algorithm works as follows (rho is the distance threshold) [3]. This query algorithm keeps a heap (minimum binary heap) with the pairs of entries from the pairs of nodes visited so far. The heap contains tuples of the form <Ei.mbr, Ej.mbr, *mindist*(Ei.mbr, Ej.mbr)> and the entry pair with the minimum *mindist* is visited first. The corresponding tuple is replaced in the heap with new tuples (with the same form) for each entry (mbr) in the node pointed by Ei.mbr and each entry (mbr) in the node pointed by Ej.mbr (this action is usually called node expansion). The process is repeated while the heap is not empty and *mindist* of the selected pairs of entries is less than of equal to rho.

5.1 The New Algorithm for 2D1T

To compute the *Intersection Join, K-CPQ* and *Buffer Query* on 2D1T we have devised a Breadth-First Traversal (BFT) algorithm, called *New.* It synchronously traverses the R-tree in breadth-first order, while processing the join condition one level at a time. At each level, *New* creates a list with the entries that satisfy the spatial predicate, named *Intermediate Candidate Entry (ICE)*, which will be accessed at the next level. The condition of the *Intersection Join* depends on the value of the *dataSet* flag (‘a’, ‘b’ or ‘c’) and the intersection of the MBRs, in order to read the nodes to the next level. When the leaf level is reached (through ICE[1]), verifying the join condition, two separate lists (V1 and V2) are created, one from each dataset. And finally, the intersection plane-sweep technique is applied to both lists (according to the spatial predicate) in order to get the final result. Global optimization has been applied level by level as in [9]. The pseudocode of this algorithm for the case of *Intersection Join* follows.

```
New (rootAddr, height)
// ICE intermediate candidate entry at level i. Entry's format is (mbr, addr, read)
// S1, S2, S3 are vectors of entries with the following format (index, mbr, read)
// V1, V2 are vectors of MBRs

node = readNode(rootAddr);
level = height;
for each entry Ei in node do
    ICE[level - 1].add(Ei.mbr, Ei.addr, false);
level = level - 1;
while (level > 0) do
    for each entry Ei in ICE[level] do
        if (ICE[level][i].mbr.dataSet == 'a') S1.add(i, Ei.mbr, false);
        if (ICE[level][i].mbr.dataSet == 'b') S2.add(i, Ei.mbr, false);
        if (ICE[level][i].mbr.dataSet == 'c')
            S3.add(i, Ei.mbr, false);
            ICE[level][i].read = true;
    SelectEntryToRead(&ICE[level], S1, S2, S3);
    for each entry Ei in ICE[level] do
        if(ICE[level][i].read == true)
            node = readNode(ICE[level][i].addr);
            if (level != 1) // Internal Nodes
                for each entry Ej in node do
                    ICE[level - 1].add(Ej.mbr, Ej.addr, false);
            else // Leaf Nodes
                for each entry Ej in node do
                    if (Ej.mbr.dataSet == 'a') V1.add(Ej.mbr);
                    if (Ej.mbr.dataSet == 'b') V2.add(Ej.mbr);
    level = level - 1;
if (level == 0) // intersection plane-sweep technique is applied
    for each pair of entries (Ei, Ej) in V1 and V2 do
        if (intersect(Ei.mbr, Ej.mbr))
            output(Ei.mbr, Ej.mbr);
```

```
SelectEntryToRead(ICE[level], S1, S2, S3)

for each entry Ei in S1 do
    if (Ei overlaps with any entry of S2 or S3)
        ICE[level][S1[i].index].read = true;
for each entry Ei in S2 do
    if (Ei overlaps with any entry of S1 or S3)
        ICE[level][S2[i].index].read = true;
```

6 Experimentation

In order to evaluate the behavior of the join algorithms on 2D2T and 2D1T, we have used four large real spatial datasets of North America, representing roads (NArd) consisting of 569,120 line-segments, and railroads (NArr) consisting of 191,637 line-segments. In order to have large datasets of points, we have transformed the MBRs of line-segments from NArd and NArr into points by taking the center of each MBR. We have also used datasets of Greece, representing rivers (Grri) consisting of 24,650 line-segments and roads (Grrd) consisting of 23,268 line-segments. When using NArd and NArr, the page size of 2D2T and 2D1T was set to 4 Kbytes (M = 204, m = 81), while when using the other datasets (that have lower cardinalities) the page size of 2D2T and 2D1T was set to 1 Kbyte (M = 50, m = 20). All data are available through the R-tree portal (http://www.rtreeportal.org/spatial.html). All experiments were performed on a Mac BookPro (Intel Core 2 Duo, 2.4 GHz) with 4 GB RAM and several GBs of secondary storage. The operating system supported was Mac OS X 10.5.8, Leopard (version of kernel Darwin 9.8.0). The programs were created using the GNU C++ compiler (gcc). The performance measurements are: (1) the number of page accesses and (2) the response time (elapsed time) reported in seconds.

First of all, results we gathered during the creation of the trees show that the 2D1T is slightly smaller than the sum of the two R*-trees that make up 2D2T (in terms of number of nodes). For example, for pairs of line segments (point) datasets for NArd and NArr the 2D1T nodes are 5393 (5490), while the sum of the 2D2T nodes are 5543 (5697).

Second, we studied the *Intersection Join* using as the two datasets NArd and NArr and also using as the two datasets Grrd and Grri. The results are depicted in Figure 2. The algorithm used for 2D2T was BFT, while the algorithms used for 2D1T were Self-BFT and New. As Figure 2 shows, the New algorithm (applied on 2D1T) is 3.7 to 4.6 times better in I/O than the BFT (applied on 2D2T), while it is 11.7 to 8.2 times worse than the BFT in response time. The Self-BFT (applied on 2D1T) is more than 10 times worse than the New algorithm in I/O and more than 3 times better than the New algorithm in response time.

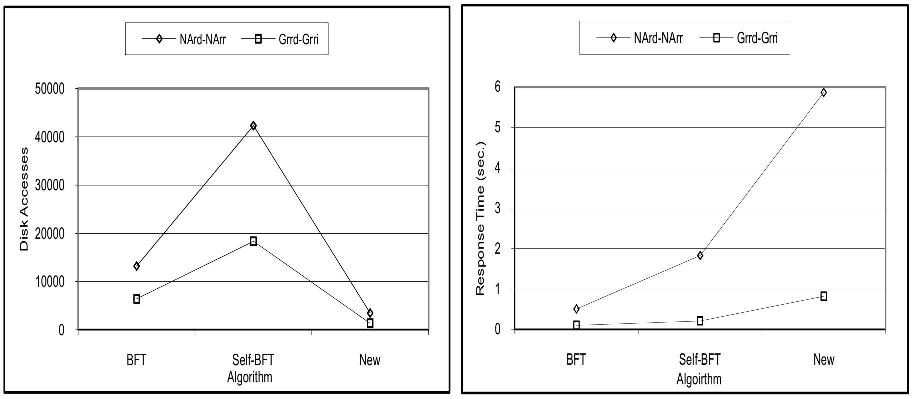


Fig. 2. Processing of the *Intersection Join*

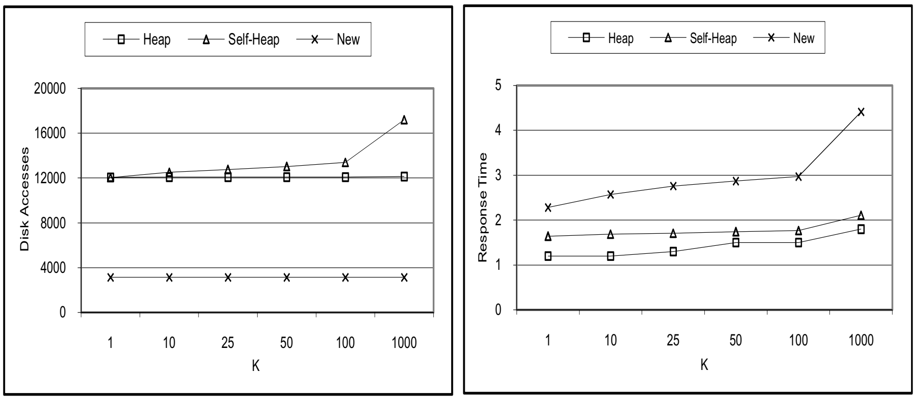


Fig. 3. Processing of the *K-CPQ*

Third, we studied the *K-CPQ* using as the two sets of points NArd and NArr (taking the center of each MBR), for K values ranging from 1 to 1000. The results are depicted in Figure 3. The algorithm used for 2D2T was Heap, while the algorithms used for 2D1T were Self-Heap and New. As Figure 3 shows, the New algorithm (applied on 2D1T) is 3.8 times better in I/O than the Heap and 2 times worse in response time. The Self-Heap (applied on 2D1T) is more than 3.8 times worse than the New algorithm in I/O and more than 1.4 times better than the New algorithm in response time.

Forth, we studied the *Buffer Query* using as the two datasets of points NArd and NArr (taking the center of each MBR), for rho (distance threshold) values ranging from 0.01 to 0.1. The results are depicted in Figure 4. The algorithm used for 2D2T was Heap, while the algorithms used for 2D1T were Self-Heap and New. As Figure 4 shows, the New algorithm (applied on 2D1T) 3.6 to 4.3 times better in I/O than the Heap algorithm and 4.2 to 5.2 times in response

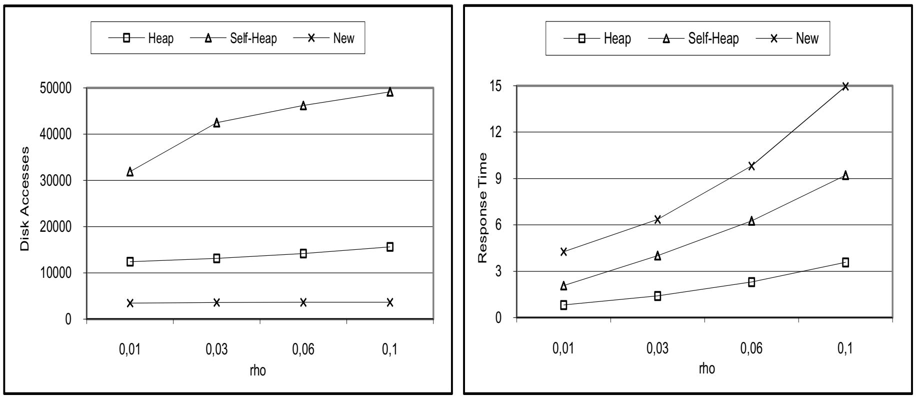


Fig. 4. Processing of the *Buffer Query*

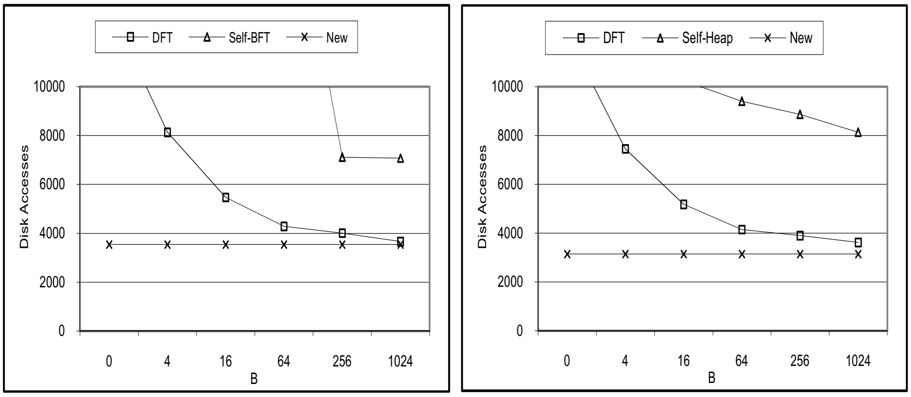


Fig. 5. The effect of the use of a variable LRU buffer

time. The Self-Heap (applied on 2D1T) is more than 9 times worse than the New algorithm in I/O and more than 1.6 times better than the New algorithm in response time.

Fifth, we studied the effect of using an LRU buffer on I/O performance on 2D2T and 2D1T join algorithms. We performed experiments of computing *Intersection Join* and *K-CPQ* by using DFT (that exhibits the best I/O performance in combination with an LRU buffer, according to [6]) for 2D2T and by using Self-BFT and New (*Intersection Join*) and Self-Heap and New (*K-CPQ*) for 2D1T. The results are depicted in Figure 5 (the left chart is for *Intersection Join* and the right chart is for *K-CPQ*, where K = 100) for NArd and NArr datasets, varying the buffer size (B = 0, 4, 16, 64, 256, 1024). It is very interesting to observe that both charts follow the same trend for these two different types of spatial queries, which indicates that the combination of the traversal order of the indexes and the increment of the buffer size affect in the same way these two spatial queries. We can notice that the I/O performance of the New algorithm was always the best and invariant to the buffer size (contrary to DFT that improves its I/O performance, up to a buffer size and then exhibits stable I/O performance). The Self-BFT and Self-Heap had the worst behaviour in presence of an LRU buffer, mainly due to the special treatment at the leaf level and the traversal order. In summary, this excellent behaviour of the New algorithm is a clear indication that it accesses only once the disk pages that are necessary to answer the queries and does not make unnecessary accesses.

The above experimental results show that, for several join queries (*Intersection Join, K-CPQ, Buffer Query*), by using a computer with a faster CPU and RAM, the New algorithm will probably become better in response time then the algorithms applied on 2D2T. In any case, these results show that it is possible to solve these queries with fewer disk accesses. We plan to study the CPU cost of the new algorithm and focus on reducing its CPU cost. An initial examination of the algorithm shows that the most time consuming stage is processing at the leaf level, where a large number of MBR combinations are performed.

7 Conclusions and Future Work

The large size of datasets that appears in industrial and commercial modern applications raises the importance of the cost of query processing and, as a consequence, the choice of the data indexing method and the query processing technique. In this paper, we present an R*-tree variation able to index two datasets simultaneously, taking advantage of the spatial locality between the different datasets and a new algorithm for join queries (based on Breadth-First traversal, where at each level we follow Best-First selection) that is applied on this variation. Moreover, we presented results of extensive comparative experimentation between the one dataset R*-tree and the two dataset R*-trees, when processing the *Intersection Join* (on non-point datasets), *K-CPQ* and *Buffer Query* (on point datasets). Our experimentation shows that the tree storing two datasets simultaneously exhibits a much better I/O performance, while independent structures storing separate datasets exhibit better CPU performance. The winner depends on the balance between CPU power and I/O efficiency of the computing system used.

In the future, we plan to consider 2D1T variants of structures with non-overlapping nodes, such as R^+-trees, or Quadtrees, as they have been used in [11] for studying k nearest neighbour and distance join queries. Variations of such structures are promising. Note that in [11] it is concluded that "an often dismissed index structure (the Quadtree) can be a better choice than the widely used R*-tree for index-based kNN query and distance join algorithms when indices are constructed dynamically".

References

1. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The R*-tree: an Efficient and Robust Access Method for Points and Rectangles. In: SIGMOD Conference, pp. 322–331 (1990)
2. Brinkhoff, T., Kriegel, H.P., Seeger, B.: Efficient Processing of Spatial Joins Using R-trees. In: SIGMOD Conference, pp. 237–246 (1993)
3. Chan, E.: Buffer Queries. IEEE Transactions on Knowledge and Data Engineering 15(4), 895–910 (2003)
4. Comer, D.: The Ubiquitous B-tree. ACM Computing Surveys 11(2), 121–137 (1979)
5. Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M.: Closest Pair Queries in Spatial Databases. In: SIGMOD Conference, pp. 189–200 (2000)
6. Corral, A., Manolopoulos, Y., Theodoridis, Y., Vassilakopoulos, M.: Algorithms for Processing K-Closest-Pair Queries in Spatial Databases. Data & Knowledge Engineering 49(1), 67–104 (2004)
7. Gaede, V., Günther, O.: Multidimensional Access Methods. ACM Computing Surveys 30(2), 170–231 (1998)
8. Guttman, A.: R-trees: A Dynamic Index Structure for Spatial Searching. In: Proc. SIGMOD Conference, pp. 47–57 (1984)
9. Huang, Y.M., Jing, N., Rundensteiner, E.: Spatial Joins Using R-trees: Breadth-First Traversal with Global Optimizations. In: VLDB Conference, pp. 395–405 (1997)
10. Jacox, E.H., Samet, H.: Spatial Join Techniques. TODS 32(1) article 7, 1–44 (2007)
11. Kim, Y.J., Patel, J.: Performance Comparison of the R*-tree and the Quadtree for kNN and Distance Join Queries. IEEE Transactions on Knowledge and Data Engineering 22(7), 1014–1027 (2010)
12. Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A.N., Theodoridis, Y.: R-Trees: Theory and Applications. Springer, Heidelberg (2006)
13. Manolopoulos, Y., Theodoridis, Y., Tsotras, V.: Advanced Database Indexing. Kluwer Academic Publishers, Boston (1999)
14. Preparata, F.P., Shamos, M.I.: Computational Geometry: An Introduction. Springer, Heidelberg (1985)
15. Shekhar, S., Lu, C.T., Chawla, S., Ravada, S.: Efficient Join Index Based Join Processing; A Clustering Approach. IEEE Transactions on Knowledge and Data Engineering 14(6), 1400–1421 (2002)
16. Touir, A.: ML-Quadtree: The Design of an Efficient Access Method for Spatial Database Systems. Journal of King Saud Univ. 17, 43–60 (2004)

Information Leakage Analysis by Abstract Interpretation

Matteo Zanioli[1,2] and Agostino Cortesi[1]

[1] Università Ca' Foscari Venezia
[2] Université Paris Diderot (Paris 7)

Abstract. Protecting the confidentiality of information stored in a computer system or transmitted over a public network is a relevant problem in computer security. The approach of information flow analysis involves performing a static analysis of the program with the aim of proving that there will not be leaks of sensitive information. In this paper we propose a new domain that combines variable dependency analysis, based on propositional formulas, and variables' value analysis, based on polyhedra. The resulting analysis is strictly more accurate than the state of the art abstract interpretation based analyses for information leakage detection. Its modular construction allows to deal with the tradeoff between efficiency and accuracy by tuning the granularity of the abstraction and the complexity of the abstract operators.

1 Introduction

Protecting the confidentiality of information stored in a computer system or transmitted over a public network is a relevant problem in computer security.

Any information flow analysis involves performing a static analysis of the program with the aim of proving that there will not be leaks of sensitive information. There is an information flow from object x to object y whenever the information stored in x is transferred to, or used to derive information transferred to, object y. Flows would be explicit or implicit. An explicit flow occurs whenever the operations generating it are independent of the value of x; whereas an implicit flow occurs whenever a statement specifies a flow from some arbitrary z to y, but the execution depends on the value of x. The starting point in secure information flow analysis is the classification of program variables into different security levels. In the simplest case, two levels are used: public (or low, L) and secret (or high, H). The main purpose is to prevent leak of sensitive information from an hight variable to a lower one. More generally, we might work with a lattice of security levels, and we would aim to ensure that sensitive information flows only upwards in the lattice [8].

In 1982 Goguen and Meseguer introduced in [10] the notion of non-interference: *"one group of users/processes/variables, using a certain set of commands, is non-interfering with another group of users if what the first group does with those commands has no effect on what the second group of users/processes/variables* *can see"*. The idea behind non-interference is that someone observing the final values of public variables cannot conclude anything about the initial values of secret variables [14].

There is a widespread literature on methods and techniques for checking secure information flows in software: from standard control flow analysis to type inference. In a security-typed language Volpano, Irvine and Smith [16] were the first to develop a type system to enforce information flow policies, where a type is inductively associated at compile-time with program statements in such a way that well-typed programs satisfy the non-interference property. Moreover, the same problem was handled also in different situation, for example with multi-threaded programs or with programs that employ explicit cryptographic operations.

A different approach is the use of standard control flow analysis to detect information leakage, e.g. [2]. Some of these works are applied to specific system, e.g. mobile ambients [3], or to specific programs, e.g. written in VHDL [15], where the analysis of information flow is closely related to the context.

The use of abstract interpretation in language-based security is not new, even though there aren't many work that use the lattice of abstract interpretations for evaluating the security of programs. Giacobazzi and Mastroeni in [9] generalize the notion of non-interference making it parametric relatively to what an attacker can observe and use it to model attackers as abstract interpretations.

In this paper we present an information flow analysis by abstract interpretation through a combination of two analysis: a syntactic variable dependency analysis, based on propositional formulas domain, and a variable value dependency using a Polyhedra analysis. An interesting aspect is that the polyhedra analysis can be replaced with other kinds of analysis which use different domains to represent the relations among variables' values.

The idea is to use logic formulas to represent dependency between variables, refine the analysis in order to reduce as much as possible "false alarms" and detect information leakages evaluating formulas on truth-assignment functions.

The analysis of a program involves the following steps:

- For each program instruction construct: a propositional formula (ϕ_i), through a fixpoint algorithm, which show an over-approximation of dependencies that occur between variables, and a polyhedron ($\mathcal{P}_i$) which represent an over-approximation of dependencies among variables value, through a classical polyhedra analysis.
- Refine each propositional formulas ϕ_i through the information in $\mathcal{P}_i$.
- Consider the public/private partitions of variables and the truth-assignment function $\overline{\Upsilon}$, that assigns to a propositional variable the value T (true) or the value F (false) if the corresponding variable is respectively private or public. If $\overline{\Upsilon}$ does not satisfy ϕ_i, there could be some information leakages.

In order to better understand how our new dependency analysis works, consider the following example.

Example 1. When a credit card PIN reaches the issuing bank, its correspondence with the validation data (i.e. the user PAN, and possibly other public data, such as the card expiration date or the customer name) is checked via a verification API. Consider the case study showed in [4], where a strict subset of the real PIN verification function named Encrypted_PIN_Verify is considered.

This function checks the equality of the actual user PIN and the trial PIN inserted at the ATM, and it returns either the result of the verification or an error code. The former PIN is derived through the PIN derivation key *pdk* and from the public data $offset$, $vdata$, $dectab$, while the latter comes encrypted under the key k as EPB (Encrypted PIN Block). Variable *counter* counts the number of executed test. Note that the two keys are pre-loaded in the HSM (Hardware Security Module), and they are never exposed to the untrusted external environment.

```
PIN_V ( $PAN, EPB, len, offset, vdata, dectab, counter$ )  {
    $x_1 := enc_{pdk}(vdata);$
    $x_2 := left(len, x_1);$
    $x_3 := decimalize(dectab, x_2);$
    $x_4 := sum\_mod10(x_3, offset);$
    $x_5 := dec_k(EPB);$
    $x_6 := fcheck(x_5);$
    if  $(x_4 = x_6)${
        $counter := counter + 1;$  $result := {}''PIN\ correct'';$
    }else{
        $counter := counter + 1;$  $result := {}''PIN\ wrong'';$
    }
    return  $result;$
}
```

When we apply our analysis we obtain, at the end of the program, the following formula, where the implication symbol can be read as possible information flow dependency.

$$\begin{aligned}\phi =&(\overline{vdata} \rightarrow \overline{x_1}) \wedge (\overline{len} \rightarrow \overline{x_2}) \wedge (\overline{x_1} \rightarrow \overline{x_2}) \wedge (\overline{dectab} \rightarrow \overline{x_3}) \wedge (\overline{x_2} \rightarrow \overline{x_3}) \wedge \\ &(\overline{offset} \rightarrow \overline{x_4}) \wedge (\overline{x_3} \rightarrow \overline{x_4}) \wedge (\overline{EPB} \rightarrow \overline{x_5}) \wedge (\overline{x_5} \rightarrow \overline{x_6}) \wedge \\ &(\overline{x_6} \rightarrow \overline{result}) \wedge (\overline{x_4} \rightarrow \overline{result}) \wedge (\overline{x_6} \rightarrow \overline{result})\end{aligned}$$

Let $\Upsilon : V \rightarrow \{L, H\}$ be a function that assign "L" class to *counter* variable and "H" class to other variables. Through a traditional information flow analysis we would find as a false positive a warning of information leakage from variables x_4 and x_6 to variable *counter*, whereas with our analysis we obtain that $\overline{\Upsilon}$, the correspondent truth-assignment function, satisfies ϕ. In fact there is no information leakage in the program: the final value of variable *counter* is independent from the value of variables x_4 and x_6.

The rest of this paper is organized as follows. In the next two sections (Section 2 and 3) we define the concrete and the abstract domain, respectively, and in Section 4 we introduce our information leakage analysis. Finally, Section 5 concludes.

2 Concrete Domain

2.1 Sintax

We consider a simple imperative language where programs are labelled commands with the following syntax:

$$C ::= {}^{\ell}\, skip \mid {}^{\ell}\, x := E \mid C_1; C_2 \mid if\ {}^{\ell}B\ then\ C_1\ else\ C_2\ {}^{\ell'} endif \mid while\ {}^{\ell}B\ do\ C\ {}^{\ell'} done$$

With E denoting expressions evaluated in the set of values $\mathcal{V}$ with standard operations, i.e. if $\mathcal{V} = \mathbb{N}$ then E can be any arithmetical expression. In the following we will denote by $\mathbb{V}[\![P]\!]$ the set of variables of the program P, by $i[\![C]\!]$ and $f[\![C]\!]$ the initial and final label of a command C respectively and by $\mathbb{A} = \{{}^{\ell}skip, {}^{\ell}\, x := E, {}^{\ell}\, B, {}^{\ell}\, notB, {}^{\ell}\, endif, {}^{\ell}\, done\}$, the set of actions.

2.2 Semantics

An environment $\rho \in \mathcal{E}$ is a function with signature: $\mathbb{V} \to \mathcal{V}$. A state $\sigma \in \Sigma \equiv \mathbb{L} \times \mathcal{E}$ is a pair $\langle \ell, \rho \rangle$ where the environment $\rho \in \mathcal{E}$ defines the current value $\rho(x)$ of the program variable $x \in \mathbb{V}[\![C]\!]$ and the program label $\ell \in \mathbb{L}$ specifies which part of the program remains to be executed. We denote by $\mathtt{E}[\![E]\!]\rho$ and $\mathtt{B}[\![B]\!]\rho$ the expression and the condition evaluation of $E \in \mathbb{E}$ and $B \in \mathbb{B}$, respectively.

The execution of a program starts at its initial label with any possible value of the variables. Therefore, the set $\mathcal{I}$ of possible initial states of a program P is $\mathcal{I}[\![P]\!] \equiv \{\langle i[\![P]\!], \rho \rangle \mid \rho \in \mathcal{E}\}$. In the same way, we can define $\mathcal{F}[\![P]\!]$ as the set of possible final state of P: $\mathcal{F}[\![P]\!] \equiv \{\langle f[\![P]\!], \rho \rangle \mid \rho \in \mathcal{E}\}$.

The labelled transition semantics $T^{\ell}[\![C]\!]$ of a command C in a program P is a set of transitions $\langle \sigma_1, A, \sigma_2 \rangle$ between a state σ_1 and its next states σ_2 by action A, satisfying the transition rule $\sigma_1 \xrightarrow{A} \sigma_2$.

$$T^{\ell}[\![{}^{\ell}skip]\!] \equiv \{\langle \ell, \rho \rangle \xrightarrow{{}^{\ell}skip} \langle f[\![{}^{\ell}skip]\!], \rho \rangle \mid \rho \in \mathcal{E}\}$$

$$T^{\ell}[\![{}^{\ell}X := E]\!] \equiv \{\langle \ell, \rho \rangle \xrightarrow{{}^{\ell}X:=E} \langle f[\![{}^{\ell}X := E]\!], \rho[X \leftarrow v] \rangle \mid \rho \in \mathcal{E} \wedge v \in \mathtt{E}[\![E]\!]\rho\}$$

$$T^{\ell}[\![if\ {}^{\ell}B\ then\ C_1\ else\ C_2\ {}^{\ell'} endif]\!] \equiv T^{\ell}[\![C_1]\!] \cup T^{\ell}[\![C_2]\!] \cup$$

$$\{\langle \ell, \rho \rangle \xrightarrow{{}^{\ell}B} \langle i[\![C_1]\!], \rho \rangle \mid \rho \in \mathcal{E} \wedge\ true\ \in \mathtt{B}[\![B]\!]\rho\} \cup$$

$$\{\langle \ell', \rho \rangle \xrightarrow{{}^{\ell}endif} \langle f[\![if\ {}^{\ell}B\ then\ C_1\ else\ C_2\ {}^{\ell'} endif]\!], \rho \rangle \mid \rho \in \mathcal{E}\}$$

$$T^{\ell}[\![C_1; C_2]\!] \equiv T^{\ell}[\![C_1]\!] \cup T^{\ell}[\![C_2]\!]$$

$$T^{\ell}[\![while\ {}^{\ell}B\ do\ C\ {}^{\ell'} done]\!] \equiv \{\langle \ell, \rho \rangle \xrightarrow{{}^{\ell}not\ B} \langle \ell', \rho \rangle \mid \rho \in \mathcal{E} \wedge false \in \mathtt{B}[\![B]\!]\rho\} \cup$$

$$\{\langle \ell, \rho \rangle \xrightarrow{{}^{\ell}B} \langle i[\![C]\!], \rho \rangle \mid \rho \in \mathcal{E} \wedge true \in \mathtt{B}[\![B]\!]\rho\} \cup T^{\ell}[\![C]\!] \cup$$

$$\{\langle \ell', \rho \rangle \xrightarrow{{}^{\ell}done} \langle f[\![while\ {}^{\ell}B\ do\ C\ {}^{\ell'} done]\!], \rho \rangle \mid \rho \in \mathcal{E}\}$$

2.3 Concrete Domain

A labelled transition system is a tuple $\langle \Sigma, \mathcal{I}, \mathcal{F}, \mathbb{A}, T^\ell \rangle$, where Σ is a nonempty set of states, $\mathcal{I} \subseteq \Sigma$ is a nonempty set of initial states, $\mathcal{F} \subseteq \Sigma$ is a set of final states, $\mathbb{A}$ is a nonempty set of actions, and $T^\ell \in \wp(\Sigma \times \mathbb{A} \times \Sigma)$ is the labelled transition relation.

We define the partial trace semantics of a transition system as the set of all possible traces, denoted by $\Sigma^\star$, recording the observation of an execution during a finite time, starting from an initial state and possibly reaching a final state.

$$\Sigma^\star \in \wp(\Sigma \times \mathbb{A} \times \Sigma)$$

$$\Sigma^\star = \{\sigma_0 \xrightarrow{A_0} \ldots \xrightarrow{A_{n-1}} \sigma_n \mid n \geq 1 \wedge \sigma_0 \in \mathcal{I} \wedge \forall i \in [0, n-1] : \sigma_i \xrightarrow{A_i} \sigma_{i+1} \in T^\ell\}$$

This set, with order relation "$\preceq$" and meet operator"$\curlywedge$", forms the meet semi lattice $(\Sigma^\star, \preceq, \curlywedge)$. Let $\pi_0, \pi_1 \in \Sigma^\star$ be two partial traces, $\pi_0 \preceq \pi_1$ if and only if π_0 is a subtrace of π_1 and $\pi_0 \curlywedge \pi_1 = \pi$ such that $(\pi \preceq \pi_1) \wedge (\pi \preceq \pi_2)$ and $(\forall \pi' : (\pi' \preceq \pi_1) \wedge (\pi' \preceq \pi_2)).\pi' \preceq \pi$.
This partial trace semantics can be expressed also in fixpoint form.

$$\Sigma^\star = \mathrm{lfp}^{\subseteq} F^t \text{ where}$$
$$F \in \wp(\Sigma \times \mathbb{A} \times \Sigma) \rightarrow \wp(\Sigma \times \mathbb{A} \times \Sigma)$$

Where

$$F(X) \equiv \{\sigma \xrightarrow{A'} \sigma' \in T^\ell \mid \sigma \in \mathcal{I}\} \cup$$
$$\{\sigma_0 \xrightarrow{A_0} \ldots \xrightarrow{A_{n-2}} \sigma_{n-1} \xrightarrow{A_{n-1}} \sigma_n \,|\, \sigma_0 \xrightarrow{A_0} \ldots \xrightarrow{A_{n-2}} \sigma_{n-1} \in X \wedge \sigma_{n-1} \xrightarrow{A_{n-1}} \sigma_n \in T^\ell\}$$

Let $\langle \wp(\Sigma^\star), \sqsubseteq, \emptyset, \Sigma^\star, \sqcap, \sqcup \rangle$ be a complete lattice of partial execution traces, where "$\sqsubseteq$", "$\sqcap$" and "$\sqcup$" are defined as follows. Consider $\Pi^0, \Pi^1 \in \wp(\Sigma^\star)$, $\Pi^0 \sqsubseteq \Pi^1$ if and only if $\forall \pi^0 \in \Pi^0 . \exists \pi^1 \in \Pi^1$ such that $\pi^0 \preceq \pi^1$; $\Pi^0 \sqcup \Pi^1 = \{\pi \in \Pi^0 \cup \Pi^1 \mid \forall \pi' \in \Pi^0 \cup \Pi^1, \pi' \preceq \pi\}$ and $\Pi^0 \sqcap \Pi^1 = \{\pi \mid (\exists \pi^0 \in \Pi^0 \wedge \pi^1 \in \Pi^1).\pi = \pi^0 \curlywedge \pi^1 \wedge (\forall \pi' : (\pi' \preceq \pi^0) \wedge (\pi' \preceq \pi^1)).\pi' \preceq \pi\}$.

3 Abstract Domain

Our information leakage analysis combines a variable dependency analysis, based on propositional formulas, and variables' value analysis, based on polyhedra, through the reduced product of the corresponding representation domains.

3.1 Variables Dependency Analysis

Propositional Formulas. Let $\mathbb{V}_p = \{\overline{x}, \overline{y}, \overline{z}, \ldots\}$ be a countably infinite set of propositional variables and let $FP(\mathbb{V}_p)$ be the set of finite subset of variables of $\mathbb{V}_p$. The set of propositional formulas constructed over the variables of $\mathbb{V}_p$ and the logical connectives in $\Gamma \subseteq \{\wedge, \vee, \rightarrow, \neg\}$ is denoted by $\Omega(\Gamma)$. For any $U \in FP(\mathbb{V}_p)$, $\Omega_U(\Gamma)$ consists of formulas using only the variables of U and the connectives of Γ.

A truth-assignment is a function $r : \mathbb{V}_p \rightarrow \{true,\ false\}$. Given a formula $f \in \Omega(\{\wedge, \vee, \rightarrow, \neg\})$, $r \vDash f$ means that r satisfies f, and $f_1 \vDash f_2$ is a shorthand for "$r \vDash f_1$ implies $r \vDash f_2$". $\Omega(\{\wedge, \vee, \rightarrow, \neg\})$ is ordered by $f_1 \trianglelefteq f_2$ if $f_2 \vDash f_1$. Two formulas f_1 and f_2 are logically equivalent, denoted $f_1 \equiv f_2$ if $f_1 \vDash f_2$ and $f_2 \vDash f_1$.

The unit assignment u is defined by $u(\overline{x}) = true$ for all $\overline{x} \in \mathbb{V}_p$. We define the set of positive formulas by: $Pos = \{f \in \Omega(\{\wedge, \vee, \rightarrow, \neg\}) \mid u \models f\}$, as in [5]. Some obvious examples: $T, \overline{x}_1 \in Pos$ and $F, \neg\overline{x}_1 \notin Pos$.

We can consider the propositional formula ϕ as a conjunction of subformulas $(\zeta_0 \wedge \ldots \wedge \zeta_n)$. We denote the set of subformulas of ϕ as Sub_ϕ. Let $\triangledown$ be least upper bound operator on propositional formula, $\triangledown\{\phi_0, \ldots, \phi_n\} = \bigwedge\{Sub_{\phi_0}, \ldots, Sub_{\phi_n}\}$. Therefore $(Pos, \trianglelefteq, \triangledown)$ is a join semi lattice. Moreover, consider $\ominus : Pos \times Pos \rightarrow Pos$: a binary operator defined as subtraction between two propositional formulas: $\phi_0 \ominus \phi_1 = \bigwedge(Sub_{\phi_0} \setminus Sub_{\phi_1})$.

Abstract Domain. An abstract state $\sigma^\sharp \in \Sigma^\sharp \equiv \mathbb{L} \times Pos$ is a pair $\langle \ell, \phi \rangle$ which denotes the dependencies that occur among program variables, up to label $\ell \in \mathbb{L}$, expressed by the propositional formula $\phi \in Pos$. Given a pair $\sigma^\sharp = \langle \ell, \phi \rangle$, we define $l(\sigma^\sharp) = \ell$ and $r(\sigma^\sharp) = \phi$. Notice that the propositional variables are denoted by $\bar{\square}$. Let $BV(C)$ be the set of bound variables of command C.

$$\begin{aligned}
BV({}^\ell skip) &= \{\emptyset\} \\
BV({}^\ell x := E) &= \{\overline{x}\} \\
BV(C_0; C_1) &= BV(C_0) \cup BV(C_1) \\
BV(if\ {}^\ell B\ then\ C_0\ else\ C_1\ {}^{\ell'} endif) &= BV(C_0) \cup BV(C_1) \\
BV(while\ {}^\ell B\ do\ C\ {}^{\ell'} done) &= BV(C)
\end{aligned}$$

The abstract transition semantics $\overline{T^\ell}[\![C]\!]$ of a command C is a set of transition $\langle \sigma_1^\sharp, \sigma_2^\sharp \rangle$ between abstract states $\sigma_1^\sharp$ and $\sigma_2^\sharp$. Similarly to the concrete domain we denote this transition by $\sigma_1^\sharp \rightarrow \sigma_2^\sharp$.

$$\begin{aligned}
&\overline{T^\ell}[\![{}^\ell skip]\!] = \{\langle \ell, \phi \rangle \rightarrow \langle f[\![{}^\ell skip]\!], \phi \rangle\} \\
&\overline{T^\ell}[\![{}^\ell x := E]\!] = \{\langle \ell, \phi \rangle \rightarrow \langle f[\![{}^\ell x := E]\!], \phi' \rangle\} \\
&\overline{T^\ell}[\![C_0; C_1]\!] = \overline{T^\ell}[\![C_0]\!] \cup \overline{T^\ell}[\![C_1]\!] \\
&\overline{T^\ell}[\![if\ {}^\ell B\ then\ C_0\ else\ C_1\ {}^{\ell'} endif]\!] = \overline{T^\ell}[\![C_0]\!] \cup \overline{T^\ell}[\![C_1]\!] \cup \\
&\quad \{\langle \ell, \phi \rangle \rightarrow \langle i[\![C_0]\!], \phi \rangle\} \cup \{\langle \ell, \phi \rangle \rightarrow \langle i[\![C_1]\!], \phi \rangle\} \cup \\
&\quad \{\langle \ell', \phi \rangle \rightarrow \langle f[\![if\ {}^\ell B\ then\ C_0\ else\ C_1\ {}^{\ell'} endif]\!], \phi''_{C_0} \rangle\} \\
&\quad \{\langle \ell', \phi \rangle \rightarrow \langle f[\![if\ {}^\ell B\ then\ C_0\ else\ C_1\ {}^{\ell'} endif]\!], \phi''_{C_1} \rangle\} \\
&\overline{T^\ell}[\![while\ {}^\ell B\ do\ C\ {}^{\ell'} done]\!] = \overline{T^\ell}[\![C]\!] \cup \{\langle \ell, \phi \rangle \rightarrow \langle i[\![C]\!], \phi \rangle\} \cup \\
&\quad \{\langle \ell', \phi \rangle \rightarrow \langle f[\![while\ {}^\ell B\ do\ C\ {}^{\ell'} done]\!], \phi''' \rangle\}
\end{aligned}$$

where:

$$\begin{aligned}
\phi' &= \bigwedge\{\overline{y} \to \overline{x} \mid \overline{y} \in \mathbb{V}_p[\![E]\!] \wedge \overline{y} \neq \overline{x}\} \wedge (\phi \ominus \bigwedge\{\overline{y} \to \overline{x} \mid \overline{y} \in \mathbb{V}_p \wedge \overline{x} \notin \mathbb{V}_p[\![E]\!]\})\\
\phi''_{C_0} &= \bigwedge\{y \to x \mid y \in \mathbb{V}_p[\![B]\!] \wedge x \in BV(C_0) \wedge y \neq x\} \wedge \phi\\
\phi''_{C_1} &= \bigwedge\{y \to x \mid y \in \mathbb{V}_p[\![B]\!] \wedge x \in BV(C_1) \wedge y \neq x\} \wedge \phi\\
\phi''' &= \bigwedge\{\overline{y} \to \overline{x} \mid \overline{y} \in \mathbb{V}_p[\![B]\!] \wedge \overline{x} \in BV(C) \wedge \overline{y} \neq \overline{x}\} \wedge \phi
\end{aligned}$$

Consider the set $\wp(\Sigma^\sharp)$, S_1 and S_2, two sets of abstract state, such that

$$S_1 = \{\langle \ell_0^1, \phi_0^1\rangle, \ldots, \langle \ell_n^1, \phi_n^1\rangle\}\ S_2 = \{\langle \ell_0^2, \phi_0^2\rangle, \ldots, \langle \ell_m^2, \phi_m^2\rangle\}$$

and $S_1 \sqsubseteq^\sharp S_2$ if and only if $n \leq m$, $\forall i \in [0,n], \ell_i^1 = \ell_i^2$ and $\forall i \in [0,n], \phi_i^1 \trianglelefteq \phi_i^2$.

We can define a join and a meet operation on this set. Let $S_0, \ldots S_n \in \wp(\Sigma^\sharp)$ be sets of abstract states, the join operation "$\sqcup^\sharp$" is defined as:

$$\begin{aligned}
\sqcup^\sharp\{S_0,\ldots,S_n\} = &\bigcup(S_0,\ldots,S_n)\\
&\cup \{\langle \ell, \phi\rangle \mid \phi = \triangledown\{\phi' \mid \langle \ell, \phi'\rangle \in \bigcup(S_0,\ldots,S_n)\}\}\\
&\setminus \{\langle \ell, \phi\rangle \in \bigcup(S_0,\ldots,S_n) \mid \exists \langle \ell, \phi'\rangle \in \bigcup(S_0,\ldots,S_n) \wedge \phi \neq \phi'\}
\end{aligned}$$

and the meet operation "$\sqcap^\sharp$":

$$\begin{aligned}
\sqcap^\sharp\{S_0,\ldots,S_n\} =&\{\langle \ell, \phi\rangle \in S' \mid S' \in \{S_0,\ldots,S_n\} \wedge\\
&\forall i \in [0,n].\exists \langle \ell, \phi_i'\rangle \in S_i \wedge \phi \trianglelefteq \phi_i'\}
\end{aligned}$$

Therefore, $\langle \wp(\Sigma^\sharp), \sqsubseteq^\sharp, \emptyset, \Sigma^\sharp, \sqcap^\sharp, \sqcup^\sharp\rangle$ is a complete lattice.

Let $\mathcal{I}^\sharp[\![P]\!] = \{\langle i[\![P]\!], \phi\rangle \mid \phi \in Pos\}$ be the set of possible initial abstract state of program P. We define the abstract semantics as the set of all finite sets of abstract states, denoted by $\Sigma^{\star\sharp}$, which can occur during one or more execution, in a finite time. For each element $\mathbb{S} \in \Sigma^{\star\sharp}$ we can denote by $\mathbb{S}^\dashv$ the set of terminal states, defined as $\mathbb{S}^\dashv = \{\sigma_0^\sharp \mid \nexists \sigma_1^\sharp \in \mathbb{S}.\sigma_0^\sharp \to \sigma_1^\sharp \in \overline{T^\ell}\}$ and by $\ell(\mathbb{S})$ all labels of the set $\mathbb{S}$. Letting $\mathbb{S}_{\sigma_0^\sharp,\sigma_n^\sharp}$ denote a set of states, called abstract sequence, that contains a starting state $\sigma_0^\sharp$ and an ending state $\sigma_n^\sharp$ such that $\forall i \in [0, n-1], \sigma_i^\sharp \to \sigma_{i+1}^\sharp \in \overline{T^\ell}$. Notice that $\mathbb{S}^\dashv_{\sigma_0^\sharp,\sigma_n^\sharp} = \{\sigma_n^\sharp\}$.

We express the abstract semantics in fixpoint form.

$$\begin{aligned}
&\Sigma^{\star\sharp} = lfp^{\sqsubseteq} F^\sharp \text{where}\\
&F^\sharp \in \Sigma^{\star\sharp} \to \Sigma^{\star\sharp}
\end{aligned}$$

where

$$\begin{aligned}
F^\sharp(X) \equiv&\{\sigma^\sharp \mid \sigma^\sharp \in \mathcal{I}^\sharp\}\cup\\
&\{\mathbb{S}_{\sigma_0^\sharp,\sigma_n^\sharp} \mid n \geq 1 \wedge \sigma_0^\sharp \in \mathcal{I}^\sharp \wedge \mathbb{S}_{\sigma_0^\sharp,\sigma_{n-1}^\sharp} \in X \wedge \sigma_{n-1}^\sharp \to \sigma_n^\sharp \in \overline{T^\ell}\}\cup\\
&\{\sqcup^\sharp\{\mathbb{S}_{\sigma_0^\sharp,\sigma_n^\sharp} \mid \mathbb{S}_{\sigma_0^\sharp,\sigma_n^\sharp} \in X\}\}
\end{aligned}$$

Let $\langle \Sigma^{\star\sharp}, \sqsubseteq^\sharp, \emptyset, \Sigma^\sharp, \sqcap^\sharp, \sqcup^\sharp\rangle$ be a lattice of abstract state sets, our abstract domain.

To simplify the definition of Galois connection we present another domain, isomorphic to the concrete domain. Let $\sigma^\diamond \in \Sigma^\diamond \equiv \mathbb{L} \times \mathbb{A}$ be a pair $\langle \ell, A\rangle$

where A is the action which occur at program label $\ell \in \mathbb{L}$. Consider the set $\Sigma^{\star\diamond}$ which contains all the possible sequence of $\sigma^{\diamond}$ that can occur during a finite computation, and the lattice $\langle \wp(\Sigma^{\star\diamond}), \sqsubseteq^{\diamond}, \emptyset, \Sigma^{\star\diamond}, \cap, \cup \rangle$. We define for $\Pi_0^{\diamond}, \Pi_1^{\diamond} \in \wp(\Sigma^{\star\diamond})$, $\Pi_0^{\diamond} \sqsubseteq^{\diamond} \Pi_1^{\diamond}$ if and only if for each $\pi_0^{\diamond} \in \Pi_0^{\diamond}$ exists $\pi_1^{\diamond} \in \Pi_1^{\diamond}$ such that $\pi_0^{\diamond} \preceq^{\diamond} \pi_1^{\diamond}$, while $\pi_0^{\diamond} \preceq^{\diamond} \pi_1^{\diamond}$ if and only if $\pi_0^{\diamond}$ is a subsequence of $\pi_1^{\diamond}$. Moreover, we denote by $\pi^{\diamond\dashv}$ the last state of the sequence.

We can express the relation between $\wp(\Sigma^{\star})$ and $\wp(\Sigma^{\star\diamond})$ by two funcions: the abstraction function, $\alpha^{\diamond} \in \wp(\Sigma^{\star}) \to \wp(\Sigma^{\star\diamond})$, and the concretization function, $\gamma^{\diamond} \in \wp(\Sigma^{\star\diamond}) \to \wp(\Sigma^{\star})$.

Let $X = \{\pi_0, \ldots, \pi_n\} \in \wp(\Sigma^{\star})$ be a set of partial trace and let $Y = \{\pi_0^{\diamond}, \ldots, \pi_n^{\diamond}\} \in \wp(\Sigma^{\star\diamond})$ be a set of sequences of $\sigma^{\diamond}$.

$$\begin{aligned} \alpha^{\diamond}(X) \equiv & \{\langle \ell_0, A_0 \rangle \to \ldots \to \langle \ell_m, A_m \rangle \mid \sigma_0 \xrightarrow{\ell_0\, A_0} \ldots \xrightarrow{\ell_m\, A_m} \sigma_{m+1} \in X\} \\ \gamma^{\diamond}(Y) \equiv & \{\pi \in \wp(\Sigma^{\star}) \mid \alpha^{\diamond}(\{\pi\}) \in Y\} \end{aligned}$$

Notice that $(\gamma^{\diamond}, \wp(\Sigma^{\star}), \wp(\Sigma^{\star\diamond}), \alpha^{\diamond})$ is an isomorphism: in fact it's simple to prove that $\gamma^{\diamond} \circ \alpha^{\diamond} = \alpha^{\diamond} \circ \gamma^{\diamond} = id$, where id is the identity function.

Then, we can define the relation between $\wp(\Sigma^{\star\diamond})$ and $\Sigma^{\star\sharp}$ by the abstraction function $\alpha^{\sharp}$ and $\gamma^{\sharp}$. Let $X \in \wp(\Sigma^{\star\diamond})$ be a set of sequences of $\sigma^{\diamond}$, $\alpha^{\sharp} : \wp(\Sigma^{\star\diamond}) \to \Sigma^{\star\sharp}$ is defined as $\alpha^{\sharp}(X) = \sqcup^{\sharp}\{\theta(\pi^{\diamond}) \mid \pi^{\diamond} \in X\}$, where $\theta : \Sigma^{\star\diamond} \to \Sigma^{\star\sharp}$ is defined ad follows.

$$\begin{aligned} \theta(X) = & \{\langle \ell, \phi \rangle \mid \forall \pi \in X. \forall \pi' = \langle \ell_0, A_0 \rangle \to \langle \ell_m, A_m \rangle \preceq^{\diamond} \pi : \\ & m \geq 0 \land \ell = \ell_m \land \phi = f_0 \land \ldots \land f_n\} \end{aligned}$$

such that:

- $(\forall \langle \ell, x := E \rangle \in \pi' : \forall \langle \ell', x := E' \rangle \in \pi'. \ell' \leq \ell). \exists f_i = \overline{y} \to \overline{x} : \overline{y} \in \mathbb{V}_p[\![E]\!]$
- $\forall((\langle \ell_i, B \rangle \to \ldots \to \langle \ell_j, \mathit{endif} \rangle) \lor (\langle \ell_i, \mathit{not}\ B \rangle \to \ldots \to \langle \ell_j, \mathit{endif} \rangle)) \preceq^{\diamond} \pi^{\diamond}$ which represents an *if* statement and $\forall \langle \ell_k, x := E_k \rangle : i < k < j$ exists $f_h = \overline{y} \to \overline{x}$ such that $\overline{y} \in \mathbb{V}_p[\![B]\!]$.
- $\forall((\langle \ell_i, B \rangle \to \ldots \to \langle \ell_j, \mathit{done} \rangle) \lor (\langle \ell_i, \mathit{not}\ B \rangle \to \ldots \to \langle \ell_j, \mathit{done} \rangle)) \preceq^{\diamond} \pi^{\diamond}$ which represents an *while* statement and $\forall \langle \ell_k, x := E_k \rangle : i < k < j$ exists $f_h = \overline{y} \to \overline{x}$ such that $\overline{y} \in \mathbb{V}_p[\![B]\!]$.

Notice that $\langle \ell_i, B \rangle \to \ldots \to \langle \ell_j, \mathit{endif} \rangle$ (or $\langle \ell_i, \mathit{not}\ B \rangle \to \ldots \to \langle \ell_j, \mathit{endif} \rangle$) represents an *if* statement if and only if $\forall(\langle \ell_p, B \rangle \lor \langle \ell_p, \mathit{not}\ B \rangle) : i < p < j. \exists(\langle \ell_q, \mathit{endif} \rangle \lor \langle \ell_q, \mathit{done} \rangle) : p < q < j$ and $\forall(\langle \ell_q, \mathit{endif} \rangle \lor \langle \ell_q, \mathit{done} \rangle) : i < q < j. \exists(\ell_p B \lor \langle \ell_p, \mathit{not}\ B \rangle) : i < p < q$. Similarly for *while* statement.

On the other hand, the concretization function $\gamma^{\sharp} : \Sigma^{\star\sharp} \to \wp(\Sigma^{\star\diamond})$ is defined as follows. Let $Y \in \Sigma^{\star\sharp}$:

$$\gamma^{\sharp}(Y) = \{\pi^{\diamond} \in \Sigma^{\star\diamond} \mid \theta(\pi^{\diamond}) \sqsubseteq^{\sharp} Y \land l(\pi^{\diamond\dashv}) \in \ell(Y^{\dashv})\}$$

It's simple to prove that $\gamma^{\sharp}$ and $\alpha^{\sharp}$ are monotone, $\gamma^{\sharp} \circ \alpha^{\sharp}$ is extensive, $\alpha^{\sharp} \circ \gamma^{\sharp}$ is equivalent to the identity and that $(\gamma^{\sharp}, \wp(\Sigma^{\star\diamond}), \Sigma^{\star\sharp}, \alpha^{\sharp})$ is a Galois insertion.

Finally, we can express the relation between $\wp(\Sigma^{\star})$ and $\Sigma^{\star\sharp}$ by the composition of above functions, $\alpha = \alpha^{\sharp} \circ \alpha^{\diamond}$ and $\gamma = \gamma^{\diamond} \circ \gamma^{\sharp}$.

By property of function composition we can assert that $(\gamma, \wp(\Sigma^{\star}), \Sigma^{\star\sharp}, \alpha)$ is a Galois insertion.

3.2 Polyhedra Analysis

Convex polyhedra are regions of some n-dimensional space that are bounded by a finite set of hyperplanes. A convex polyhedron in $\mathbb{R}^n$ describes a relation between n quantities. In the seminal work [7], P. Cousot and N. Halbwachs applied the theory of abstract interpretation to the static determination of linear equality and inequality relations among program variables and introduced the use of convex polyhedra as a domain of descriptions to solve a number of important data-flow analysis problems.

There are many works in literature on the use of the polyhedra domain and relative Galois connection and on its implementations [1,12]. Therefore we can omit a comprehensive presentation of polyhedra analysis and provide only some basic notions.

For $n > 0$ we denote by $\mathbf{v} = (v_0, \dots v_{n-1}) \in \mathbb{R}^n$ an n-tuple (vector) of real numbers; $\mathbf{v} \cdot \mathbf{w}$ denotes the scalar product of vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$; the vector $\mathbf{0} \in \mathbb{R}^n$ has all components equal to zero. Let $\mathbf{x}$ be a n-tuple of distinct variable. Then $\beta = (\mathbf{a} \cdot \mathbf{x} \bowtie b)$ denotes a linear equality and inequality constraint, for each vector $\mathbf{a} \in \mathbb{R}^n$, where $\mathbf{a} \neq \mathbf{0}$, each scalar $b \in \mathbb{R}$ and $\bowtie = \{=, \geq, >\}$. A linear inequality constraint β defines an affine half-space of $\mathbb{R}^n$, denoted by $con(\{\beta\})$.

A set $\mathcal{P} \in \mathbb{R}^n$ is a (convex) polyhedron if and only if $\mathcal{P}$ can be expressed as the intersection of a finite number of affine half-spaces of $\mathbb{R}^n$, i.e. as the solution $\mathcal{P} = con(\mathcal{C})$ of a finite set of linear inequality constraints $\mathcal{C}$. The set of all polyhedra on the vector space $\mathbb{R}^n$ is denoted as $\mathbb{P}_n$. Let $\langle \mathbb{P}_n, \subseteq, \emptyset, \mathbb{R}^n, \uplus, \cap \rangle$ be a lattice of convex polyhedra, where "$\subseteq$" is the set-inclusion, the empty set and $\mathbb{R}^n$ as the bottom and top elements, respectively; the binary meet operation, returning the greatest polyhedron smaller than or equal to the two arguments, correspond to set-intersection and "$\uplus$" is the binary join operation and return the least polyhedron greater than or equal to the two arguments, called convex polyhedral hull. Moreover let $G_{\wp(\Sigma^\star), \mathbb{P}_n} = (\gamma_{\mathbb{P}_n, \wp(\Sigma^\star)}, \wp(\Sigma^\star), \mathbb{P}_n, \alpha_{\wp(\Sigma^\star), \mathbb{P}_n})$ be a Galois connection between the concrete domain $\wp(\Sigma^\star)$ and abstract domain $\mathbb{P}_n$.

3.3 Reduced Product

We combine the abstract domains $\langle \Sigma^{\star\sharp}, \sqsubseteq^\sharp, \emptyset, \Sigma^{\star\sharp}, \sqcap^\sharp, \sqcup^\sharp \rangle$ and $\langle \mathbb{P}_n, \subseteq, \emptyset, \mathbb{R}^n, \uplus, \cap \rangle$ through a reduced product operator [6].

Let $G_{\wp(\Sigma^\star), \Sigma^{\star\sharp}}$ and $G_{\wp(\Sigma^\star), \mathbb{P}_n}$ be Galois connection and let $\varrho : \Sigma^{\star\sharp} \times \mathbb{P}_n \rightarrow \Sigma^{\star\sharp} \times \mathbb{P}_n$ be a reduction operator defined as follows: let $X \in \Sigma^{\star\sharp}$ be a set of partial traces and let $\mathcal{P} \in \mathbb{P}_n$ be a polyhedra.

$$\varrho(\langle X, \mathcal{P} \rangle) = \langle X', \mathcal{P} \rangle$$

such that

$$\begin{aligned} X' = &\{\sigma^\sharp_{new} \mid \forall \sigma^\sharp \in X. l(\sigma^\sharp_{new}) = l(\sigma^\sharp) \wedge \\ &\wedge r(\sigma^\sharp_{new}) = (r(\sigma^\sharp) \ominus \{x \rightarrow y \mid y = z \in \mathcal{P}, z \in \mathbb{V}_p \cup \mathbb{Z} \wedge z \neq x\})\} \end{aligned}$$

Then, the reduced product $D^\natural$ is defined as follows:

$$D^\natural = \{\varrho(\langle X, \mathcal{P} \rangle) \mid X \in \Sigma^{\star\sharp}, \mathcal{P} \in \mathbb{P}_n\}$$

Consider $X_0, X_1 \in \Sigma^{\star\sharp}, \mathcal{P}_0, \mathcal{P}_1 \in \mathbb{P}_n$ and $\langle X_0, \mathcal{P}_0\rangle, \langle X_1, \mathcal{P}_1\rangle \in D^\natural$: $\langle X_0, \mathcal{P}_0\rangle \sqsubseteq^\natural \langle X_1, \mathcal{P}_1\rangle$ if and only if $X_0 \sqsubseteq^\sharp X_1$ and $\mathcal{P}_0 \subseteq \mathcal{P}_1$. Let $\sqcup^\natural : D^\natural \to D^\natural$ and $\sqcap^\natural : D^\natural \to D^\natural$ be the least upper bound and greatest lower bound operator, respectively, defined as $\langle X_0, \mathcal{P}_0\rangle \sqcup^\natural \langle X_1, \mathcal{P}_1\rangle = \langle X_0 \sqcup^\sharp X_1, \mathcal{P}_0 \uplus \mathcal{P}_1\rangle$ and $\langle X_0, \mathcal{P}_0\rangle \sqcap^\natural \langle X_1, \mathcal{P}_1\rangle = \langle X_0 \sqcap^\sharp X_1, \mathcal{P}_0 \cap \mathcal{P}_1\rangle$.

Therefore $\langle D^\natural, \sqsubseteq^\natural, \emptyset, \varrho(\langle \Sigma^{\star\sharp}, \mathbb{R}^n\rangle), \sqcup^\natural, \sqcap^\natural\rangle$ is a complete lattice.

The reduce operator showed above is aimed at excluding pointless dependencies for all variables which have the same value during the execution, without the loss of purposeful relations (by the condition "$x \neq z$"). In order to better understand the improvements yielded by the combination of the two domains consider the following example.

Example 2.

```
$foo()\{$
    $^0n = 0;\ ^1x = 1;\ ^2i = 0;\ ^3y = x - 1;\ ^4sum = p;$
    $while(^5i \leq k)\ do$
        $if(^6n\%2 == 0)\ then$
            $^7sum = y + p;\ ^8n = n + 1;$
        $else$
            $^9sum = x + (p - 1);\ ^{10}n = n + 3;$
        $^{11}endif$
        $^{12}i = i + 1;$
    $^{13}done$
$\}^{14}$
```

For the sake of simplicity, we show a partial representations through propositional formula and polyhedra of the variables dependency. In particular we take in account the labels 4, 5, 8, 10, 12 and 14.

	Polyhedra
4	$n = 0; x - 1 = 0; i = 0; y = 0$
5	$-p + sum = 0; y = 0; x - 1 = 0; -i + n \geq 0; 3i - n \geq 0;$
8	$-p + sum = 0; y = 0; x - 1 = 0; -i + n \geq 0; -i + k \geq 0; 3i - n \geq 0;$
10	$-p + sum = 0; y = 0; x - 1 = 0; -i + n \geq 0; -i + k \geq 0; 3i - n \geq 0;$
12	$-p + sum = 0; y = 0; x - 1 = 0; -i + n - 1 \geq 0; -i + k \geq 0;$
	$i \geq 0; 3i - n + 3 \geq 0;$
14	$-p + sum = 0; y = 0; x - 1 = 0; -i + n \geq 0; -i + k - 1 \geq 0; 3i - n \geq 0;$

	Propositional formula
4	$x \to y$
5	$p \to sum$
8	$(x \to y) \wedge (p \to sum) \wedge (y \to sum)$
10	$(x \to y) \wedge (p \to sum) \wedge (x \to sum)$
12	$(x \to y) \wedge (p \to sum) \wedge (x \to sum) \wedge (y \to sum)$
14	$(x \to y) \wedge (p \to sum) \wedge (x \to sum) \wedge (y \to sum) \wedge (n \to sum) \wedge$
	$(i \to sum) \wedge (i \to n) \wedge (k \to sum) \wedge (k \to n)$

When we apply the reduce operator defined above we obtain the following propositional formulas:

4	T
5	$p \rightarrow sum$
8	$p \rightarrow sum$
10	$p \rightarrow sum$
12	$p \rightarrow sum$
14	$(p \rightarrow sum) \wedge (i \rightarrow n) \wedge (k \rightarrow n)$

By using the reduce operator we simplified the propositional formulas, removing some implication which could in fact generate false alarms when using the direct product of the domains instead of the reduced product.

4 Analysis

An information flow analysis can be carried out by considering different attacker abilities. In this paper, we consider two scenarios: when the attacker can read public variables only at the beginning and at the end of the computation, and when the attacker can read public variables after each step of the computation. Consider that the attacker, in both cases, knows the source code of the program.

Let $\Upsilon_P : \mathbb{V} \rightarrow \{L, H\}$ be a function which assigns to each variable of program P a security class: public (L) or private (H). We say that program P is secure if and only if it does not contain any information leakage with respect to the function Υ_P, i.e. there is no information that moves from private to public variables.

Let $\overline{\Upsilon_P} : \mathbb{V}_p \rightarrow \{\mathtt{T}, \mathtt{F}\}$ be the truth-assignment function associated with Υ_P. $\overline{\Upsilon_P}(\overline{x})$ assigns to $\overline{x}$ the value T or F if the security class, assigned by Υ_P, is H or L respectively.

The aim of this analysis is to formally verify if the program is secure, therefore we look for all subsets of the prefix partial trace that do not have any information leakage. For the first case, in which the attacker can read public variables only at the beginning and at the end of the computation, the property is $\chi^1 = \{\mathbb{S} \in \Sigma^{\star\sharp} \mid \overline{\Upsilon_p} \vDash r(\mathbb{S}^{\dashv})\}$ which is all abstract sequences in $\Sigma^{\star\sharp}$ such that the propositional formula of the last abstract state is satisfied by truth-assignment function $\overline{\Upsilon_p}$. Whereas in the second case, when the attacker can read public variables at each step of the computation, the property is $\chi^2 = \{\mathbb{S} \in \Sigma^{\star\sharp} \mid \forall \sigma^\sharp \in \mathbb{S}. \overline{\Upsilon_p} \vDash r(\sigma^\sharp)\}$ the set of all abstract sequences in $\Sigma^{\star\sharp}$ which contains all the abstract states that are satisfied by the truth-assignment function $\overline{\Upsilon_p}$.

4.1 Complexity

In order to evaluate the efficiency of our analysis, we have to consider the two main components: *Pos* formulas and polyhedra.

Variables dependency analysis through propositional formulas involves two different aspects: on the one hand, the logical equivalence of two boolean expressions, a co-NP-complete problem, and on the other hand the fact that the complexity of *Pos* domain is bounded because we work only with the variables appearing in the program, whose number is, in the practice, reasonably small [11]. Moreover, it is possible to reduce the complexity using the ordered binary decision diagrams (BDDs) to provide a compact representation of many boolean functions and by using algorithms based on that.

About polyhedra analysis, the complexity is well and completely treated in many works [1] and heavily depends on its implementation.

For example many implementation, e.g. Polylib and New Polka, use matrices of coefficients, that cannot grow dynamically, and the worst case space complexity of the methods employed is exponential. In PPL library, instead, all data structures are fully dynamic and automatically expand (in amortized constant time) ensuring the best use of available memory. Comparing the efficiency of the polyhedra libraries is not simple, the pay-off depends on the targeted applications: in [1] the authors presented many test results about it.

In this paper we considered the polyhedra analysis, but, as already observed, the modular construction allows to tune efficiency and accuracy changing the domain which represents the relations among variables' values.

For instance, we can use the analysis proposed by Karr in [13]: in this way we may get a loss of precision, as this domain represnts only linear combination of the variables, but we achieve an improvement of the computational cost of the analysis (it is polynomial).

The complexity of reduced product, and more precisely of reduction operator ϱ, is strictly connected with the complexity of the operations on the domains we combine.

5 Conclusions

In this paper, we presented an information flow analysis through abstract interpretation based on a new domain that combines variable dependency analysis, based on the reduced product of a propositional formulas domain, and a polyhedra domain. The techniques used in our analysis are frequently applied in the context of generic code analysis; the main contribution of our work consists in combining these techniques in a novel way to solve a specific security problem: the detection of information leakages.

Moreover, the structure of our analysis allow easily some generalization, e.g. for multi-level security policies or for other kind of analysis (for example, knowing which variables depends on others, regardless of the security classes).

Aknowledgments

Work partially supported by RAS project "TESLA - Tecniche di enforcement per la sicurezza dei linguaggi e delle applicazioni".

References

1. Bagnara, R., Hill, P.M., Zaffanella, E.: The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Science of Computer Programming 72(1–2), 3–21 (2008)

2. Bodei, C., Degano, P., Nielson, F., Riis Nielson, H.: Static analysis for secrecy and non-interference in networks of processes. In: Malyshkin, V.E. (ed.) PaCT 2001. LNCS, vol. 2127, pp. 27–41. Springer, Heidelberg (2001)
3. Braghin, C., Cortesi, A., Focardi, R.: Information flow security in boundary ambients. Inf. Comput. 206(2-4), 460–489 (2008)
4. Centenaro, M., Focardi, R., Luccio, F.L., Steel, G.: Type-based analysis of pin processing apis. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 53–68. Springer, Heidelberg (2009)
5. Cortesi, A., File, G., Winsborough, W.: Optimal groundness analysis using propositional logic. The Journal of Logic Programming 27(2), 137–167 (1996)
6. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Conference Record of the Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 269–282. ACM Press, New York (1979)
7. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 84–97. ACM Press, New York (1978)
8. Denning, D.E.: A lattice model of secure information flow. Commun. ACM 19(5), 236–243 (1976)
9. Giacobazzi, R., Mastroeni, I.: Abstract non-interference: parameterizing non-interference by abstract interpretation. In: Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages, POPL 2004, pp. 186–197. ACM, New York (2004)
10. Goguen, J.A., Meseguer, J.: Security policies and security models. In: IEEE Symposium on Security and Privacy, vol. 0, p. 11 (1982)
11. Van Hentenryck, P., Cortesi, A., Le Charlier, B.: Evaluation of the domain prop. The Journal of Logic Programming 23(3), 237–278 (1995)
12. Jeannet, B.: Convex Polyhedra Library, release 1.1.3c edn., Documentation of the "New Polka" library (March 2002), `http://www.irisa.fr/prive/Bertrand.Jeannet/newpolka.html`
13. Karr, M.: Affine relationships among variables of a program. Acta Inf. 6, 133–151 (1976)
14. Smith, G.: Principles of secure information flow analysis. In: Christodorescu, M., Jha, S., Maughan, D., Song, D., Wang, C. (eds.) Malware Detection. Advances in Information Security, vol. 27, pp. 291–307. Springer, Heidelberg (2007)
15. Tolstrup, T.K., Nielson, F., Nielson, H.R.: Information flow analysis for vhdl. In: PaCT, pp. 79–98 (2005)
16. Volpano, D., Irvine, C., Smith, G.: A sound type system for secure flow analysis. J. Comput. Secur. 4(2-3), 167–187 (1996)

Partition into Triangles on Bounded Degree Graphs

Johan M.M. van Rooij, Marcel E. van Kooten Niekerk, and Hans L. Bodlaender

Department of Information and Computing Sciences, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
jmmrooij@cs.uu.nl, markonie@xs4all.nl, hansb@cs.uu.nl

Abstract. We consider the Partition Into Triangles problem on bounded degree graphs. We show that this problem is polynomial time solvable on graphs of maximum degree three by giving a linear time algorithm. We also show that this problem becomes $\mathcal{NP}$-complete on graphs of maximum degree four. Moreover, we show that there is no subexponential time algorithm for this problem on maximum degree four graphs unless the Exponential Time Hypothesis fails. However, the partition into triangles problem for graphs of maximum degree at most four is in many cases practically solvable as we give an algorithm for this problem that runs in $\mathcal{O}(1.02220^n)$ time and linear space. In this extended abstract, we will only give an $\mathcal{O}(1.02445^n)$ time algorithm.

1 Introduction

In his weblog of February 2009 [8], R. J. Lipton quotes Alan J. Perlis, the first Turing Award winner:

> For every polynomial-time algorithm you have, there is an exponential algorithm that I would rather run.

His point is simple: if your algorithm runs in n^4 time, then an algorithm that runs in $n2^{n/10}$ time (alternatively denoted as $n1.07178^n$ time) is faster if for example $n = 100$ (this holds for all $n \leq 236$).

The same observation for $\mathcal{NP}$-hard problems instead of polynomial time solvable problems was made by Woeginger in his well known survey on exact exponential time algorithms [12]. Woeginger considers the fact that algorithms for $\mathcal{NP}$-hard problems with exponential running times may actually lead to practical algorithms: he compares $\mathcal{O}(n^4)$ with $\mathcal{O}(1.01^n)$. We, however, are not aware of any papers on natural[1] $\mathcal{NP}$-hard problems with exponential time algorithms with running times anywhere near $\mathcal{O}(1.01^n)$ without involving huge polynomial factors (either visible, hidden in the notation, or hidden in the decimal rounding of the exponent in the big-$\mathcal{O}$). In this extended abstract, we will give such an

[1] I.e., without making artificial constructions like Independent Set restricted to graphs in which 99% of the vertices have degree at most two.

algorithm running in time $\mathcal{O}(1.02445^n)$ or $\mathcal{O}(2^{n/28.69})$ for the Partition Into Triangles problem restricted to maximum degree four graphs. In the full version [10], we show through some case analysis that there also is an $\mathcal{O}(1.02220^n)$ or $\mathcal{O}(2^{n/31.58})$ time algorithm for this problem.

The Partition Into Triangles problem a classical $\mathcal{NP}$-complete problem [4]. In this paper, we study this problem restricted to bounded degree graphs and obtain a series of results. On graphs of maximum degree three, we show that the problem is linear time solvable. On graphs of maximum degree four, we show that there exists a strong and interesting relation between Partition Into Triangles and the Exact 3-Satisfiability problem. We exploit this relation in several ways. First, we use it to show that the Partition Into Triangles problem becomes $\mathcal{NP}$-complete on graphs of maximum degree four. Second, we use it to show that there exists no subexponential time algorithm for Partition Into Triangles on maximum degree four graphs unless the Exponential Time Hypothesis [5] fails. Thus it seems that Partition Into Triangles restricted to graphs of maximum degree four is a hard problem. However, as a third application of the relation to Exact 3-Satisfiability, we give an $\mathcal{O}(1.02445^n)$ time algorithm for our problem by combining the fastest known algorithms for Exact Satisfiability [3] and Exact 3-Satisfiability [11]. The faster algorithm in the full version [10], is also based on this relation. We note that the running times of these algorithms involve no large hidden polynomial factors, which makes them effective in practice.

On general graphs, the Partition Into Triangles problem can be solved using set partitioning via inclusion-exclusion [2] in $\mathcal{O}(2^n n^{\mathcal{O}(1)})$ time and polynomial space. Better running times for Partition Into Triangles can be obtained as a side result of two recent papers. Koivisto [7] has given a general covering algorithm that can be used to solve the problem in $\mathcal{O}(1.7693^n)$ time and space. And, Björklund [1] has given a general randomised partitioning algorithm that can be used to solve the problem in $\mathcal{O}(1.496^n)$ time and polynomial space while having a probability of failure that is exponentially small in n. On bounded degree graphs, we are unfamiliar with any results besides the hardness result of Kann: he proved that the optimisation variant (cover by maximum number of triangles) is *Max-$\mathcal{SNP}$*-complete on graphs of maximum degree at least six [6].

Notation and definitions. We assume the reader to be familiar with standard notation and terminology from graph theory and computer science related logic.

A *triangle* is a collection of three vertices in G in which each pair is joined by an edge. A *triangle partition* of G is a partitioning of V in $n/3$ disjoint subsets such that each such subset forms a triangle. This paper considers the problem Partition Into Triangles: given a graph G, does G have a triangle partition?

We will often use reasoning involving the Exact 3-Satisfiability problem (X3SAT) while the 3-Satisfiability problem (3SAT) is also used. To avoid confusion, in the X3SAT problem *exactly* one variable in each clause must be set to $True$, while in the 3SAT problem *at least* one variable in each clause must be set to $True$. When there is the possibility of confusion, we denote a 3SAT clause by SAT(x, y, z) and an X3SAT clause by XSAT(x, y, z).

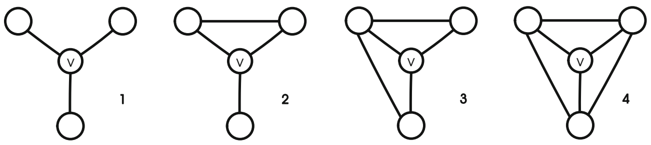


Fig. 1. Possible edges within the local neighbourhood of a vertex in a cubic graph

We denote the frequency of a variable x by $f(x)$, that is, $f(x)$ is the number of occurrences of the literals x and $\neg x$.

Paper organisation. We first give a linear time algorithm for Partition Into Triangles on graphs of maximum degree three in Section 2. Then, we relate the problem on graphs of maximum degree four to Exact 3-Satisfiability in Section 3. We use this relation to prove our hardness results in Section 4. In this extended abstract we give a simple $\mathcal{O}(1.02445^n)$ time algorithm in Section 5. In the same section, an $\mathcal{O}(1.02220^n)$ time algorithm is claimed without proof. The proof can be found in the full paper [10]. Finally, some conclusions are given in Section 6.

2 A Linear Time Algorithm on Graphs with $\Delta(G) \leq 3$

We begin by considering Partition Into Triangles on graphs of maximum degree three. We will prove that this problem is polynomial time solvable on this graph class by giving a linear time algorithm: Algorithm 1.

Algorithm 1. A linear time algorithm for graphs of maximum degree three

```
Input: A graph $G = (V, E)$ of maximum degree three.
Output: A triangle partition $T$ of $G$ or No if no such partition exists.
 1: if $|V|$ is not a multiple of three then return No
 2: while $G$ is non-empty do
 3:    Take any vertex $v \in V$.
 4:    if $N[v]$ contains a vertex of degree at most two then
 5:       Reduce the graph using Lemma 1. If a triangle is selected, then add it to $T$.
 6:    else if $N[v]$ corresponds to cases 1, 3, or 4 of Fig. 1 then
 7:       return No
 8:    else //case 2 of Fig. 1
 9:       Add the triangle in $N[v]$ to $T$ and remove its vertices from $G$.
10: return $T$
```

Lemma 1. *Let $G = (V, E)$ be an instance of* Partition Into Triangles *restricted to graphs of maximum degree d containing a vertex v of degree at most two. In constant time, we can either decide that G is a* No*-instance, or we can transform G into an equivalent smaller instance.*

Proof. If v has degree at most one, then this vertex cannot be in any triangle and the instance is a No-instance. Otherwise, let u, w be the neighbours of v. As G is of constant maximum degree, we can test in constant time whether $(u, w) \in E$. If $(u, w) \in E$, then $\{u, v, w\}$ is the unique triangle containing v, and we remove this triangle from G to obtain a smaller equivalent instance. If $(u, w) \notin E$, then v is not part of any triangle, and we again have a No-instance. □

Theorem 1. *Algorithm 1 solves* Partition Into Triangles *on graphs of maximum degree three in linear time.*

Proof. For correctness, we note that the number of vertices must be a multiple three in order to partition G into triangles. Furthermore, correctness of the first case in the main loop follows from Lemma 1. For the other two cases, we observe that any local neighbourhood of v must equal one of the four cases in Fig. 1. In case 1, no triangle containing v exists, and, in cases 3 and 4, the fact that G is cubic results in that removing any triangle would lead to vertices of degree at most 1 which can no longer be in a triangle. Hence, these are all No-instances. In case 2, v can only be part of one triangle, which Algorithm 1 selects.

Each iteration of the main loop requires constant time, since inspecting a neighbourhood in a cubic graph can be done in constant time. In each iteration, Algorithm 1 either terminates, or removes three vertices from G. Hence, there are at most a linear number of iterations and Algorithm 1 runs in linear time. □

3 The Relation between Partition into Triangles on Graphs with $\Delta(G) \leq 4$ and Exact 3-Satisfiability

When we restrict the Partition Into Triangles problem to graphs of maximum degree four, an interesting relation with Exact 3-Satisfiability emerges. This relation will be the topic of this section.

We will first give three lemmas used to either decide that an instance of Partition Into Triangles on maximum degree four graphs is a No-instance, or that it can be reduced to an equivalent smaller instance. These lemmas will apply to any instance unless all vertices in the instance have a local neighbourhood that is identical to one of two possible options. If we cannot reduce an instance in this way, connected series of one of the remaining locals neighbourhoods can be interpreted as a variable that can be set to true or false depending on in which of the two possible ways it will be partitioned into triangles. Under this interpretation, the other possible local neighbourhood can be interpreted as a clause of size three in which exactly one variable must be set to true. In this way, remaining instances can be interpreted as an Exact 3-Satisfiability instance.

Lemma 2. *Let G be an instance of* Partition Into Triangles *of maximum degree four with a vertex v of degree at most three. In constant time, we can either decide that G is a* No*-instance, or obtain an equivalent smaller instance.*

Proof. We can assume that v has degree three: otherwise we apply Lemma 1.

Similar to in the proof of Theorem 1, the local neighbourhood of v corresponds to one of the four cases in Fig. 1. If this neighbourhood corresponds to case 1, then all edges incident to v are not part of any triangle. If this neighbourhood corresponds to case 2, then the edge between v and the bottom vertex is not part of any triangle. In these two cases, we remove these edges and apply Lemma 1 to v, which now has degree at most two. If this neighbourhood corresponds to case 4, then, since G is of maximum degree four, selecting any triangle in the solution results in the creation of a vertex of degree at most one: we can conclude that we have a NO-instance. The same holds for case 3 unless the vertices a and b (see Fig. 2) are of degree four.

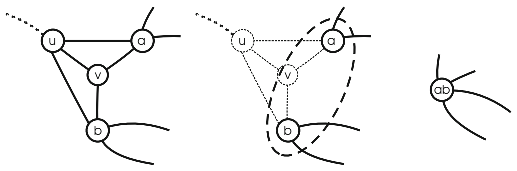


Fig. 2. Reducing an instance with a degree three vertex by merging its neighbours

In this last case, we reduce the graph as in Fig. 2. Either vertex a or vertex b must be in a triangle with u and v. Thus, the other vertex from a and b must be in a triangle with its other two neighbours. We distinguish three subcases depending on the number of common neighbours of a and b.

Let a and b have no other common neighbours than u and v. Observe that an edge between a neighbour of a and a neighbour of b outside the shown part of the graph cannot be in a triangle in any solution: we remove these if any exist. Next, we merge the vertices a and b to a single vertex and remove both u and v. Now, the new vertex is part of only two different triangles, and both possibilities corresponds to taking one of the two possible triangles containing v in the original graph. Also, no extra triangles are introduced as we have removed the edge between the neighbours of the merged vertices. We conclude that the new smaller instance is equivalent.

Alternatively, let a and b have a third common neighbour, say w. Since we must pick a triangle with u, v and either a or b, we can remove the two edges incident to a and not incident to u or v if they are not on a common triangle together. If we do so, we obtain a vertex of degree two and can apply Lemma 1. The same goes for the two edges incident to b and not incident to u or v. Hence, we can assume that both a and b lie on a triangle with their third common neighbour w. Moreover, depending on which vertex from a and b we pick in a triangle with u and v, the other must be in a triangle with w. Now, we remove u and v and merge a and b to a single vertex and remove double edges. In the new instance, the edge between ab and w can be in two triangles and the choice corresponds directly to either taking the triangle u, v, a and the triangle with b and w, or taking the triangle u, v, b and the triangle with a and w.

Finally, let a and b have four common neighbour called u, v, w and x. Again, the two pairs of edges incident to a and b not incident to u and v must be pairwise in triangles or we can remove them and apply Lemma 1. In the remaining case, each of these pairs of edges forms a triangle with the edge between w and x. Now, we must either pick the triangles u, v, a and b, w, x or we must pick u, v, b, and a, w, x. Both options involve the same vertices, hence we can remove these to obtain an equivalent smaller instance. □

As a result, we can reduce any non 4-regular instance. In a 4-regular graph, a vertex v can have a number of possible local neighbourhoods, all shown in Fig. 3. We will now show that we can reduce any instance having a vertex whose local neighbourhood does not correspond to cases 2b or 3a in Fig. 3.

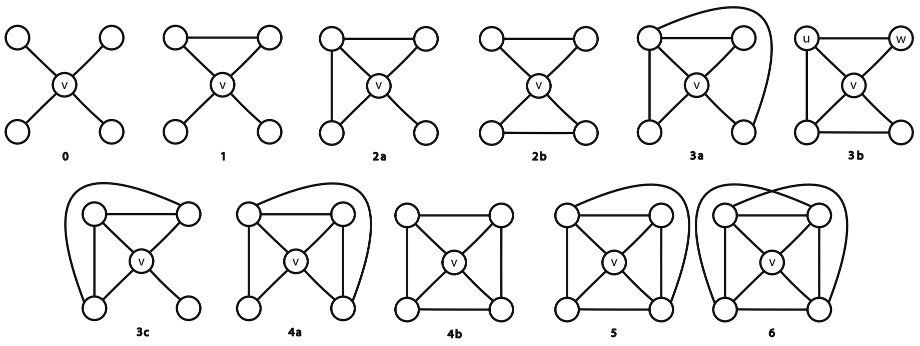


Fig. 3. Possible edges within the local neighbourhood of a degree four vertex. Notice that the numbering corresponds to the number of edges between the neighbours of v.

Lemma 3. *Let $G = (V, E)$ be a 4-regular instance of* Partition Into Triangles *containing a vertex v whose local neighbourhood is different from cases 2b, 3a and 3b in Fig. 3. In constant time, we can either decide that G is a* No*-instance, or we can transform G into an equivalent smaller instance.*

Proof. Consider the possible local neighbourhoods around v shown in Fig. 3.

If the local neighbourhood of v equals case 0, 1, 2a, or 3c, then v is incident to an edge that is not part of any triangle in G since both endpoints do not have a common neighbour. For these cases, we remove the edge and apply Lemma 2 to v. If this local neighbourhood equals case 5 or 6, then we have a No-instances since picking any triangle containing v results in a vertex of degree at most one.

To complete the proof, we consider the remaining two cases: 4a, and 4b.

Case 4a: Consider the edge from the top left vertex to the bottom right vertex. This edge is part of two triangles, one with the centre vertex v and one with the top right vertex. If we would take any of these two triangles in the solution, a vertex of degree at most one remains. Hence, this edge cannot be part of a triangle in the solution and we can apply Lemma 2 after removing this edge.

Case 4b: Consider one of the four edges in $N[v]$ not incident to v, say the edge between the top two vertices. This edge is part of one or two triangles, one with v, and one with a possible third vertex outside of $N[v]$. Assume that we take the triangle with this edge and v in a solution, then the remaining two vertices will get degree two and thus they can only be in a triangle together and with a common neighbour. Hence, for each of the four edges in $N[v]$, we remove it if the endpoints of both the edge and the opposite edge (edge between the other two vertices in $N[v] \setminus \{v\}$) have no common neighbour except for v.

Note that there is no instance in which all four edges remain since each of the four corner vertices has only one neighbour outside of $N[v]$. Hence there can be at most two such common neighbours, and if there are two then they must involve the endpoints of opposite edges. We can now apply Lemma 2. □

Having reduced the number of possible local neighbourhoods of a vertex in an instance to three, we now remove one more such possibility.

Lemma 4. *Let G be a 4-regular instance of* Partition Into Triangles *in which the local neighbourhood of each vertex equals case 2b, 3a or 3b in Fig. 3. Then, vertices whose local neighbourhood equal case 3b form separate connected components in G. We can either decide that G is a* No*-instance, or remove these components to obtain an equivalent smaller instance in linear time.*

Proof. Let v be a vertex whose local neighbourhood corresponds to case 3b of Fig. 3. Let u be the top left vertex in this picture and consider the local neighbourhood of u. This neighbourhood cannot equal case 2b of Fig. 3 as it contains one vertex adjacent to two other vertices in the neighbourhood. The neighbourhood can also not equal case 3a, since v is of degree four and thus cannot have an extra edge to the neighbour of u outside $N[v]$. We conclude that the local neighbourhood of u must equal that of case 3b in Fig. 3. Thus, the top two vertices have a common neighbour outside $N[v]$.

We can repeat this argument and apply it to u to conclude that the top right vertex in the picture w also has the same local neighbourhood. This shows that w and the new vertex created in the previous step must have another common neighbour. In this way, we conclude that every vertex in the connected component containing v has this local neighbourhood. Moreover, this connected component consists of a circular chain of these configurations as shown in Fig. 4.

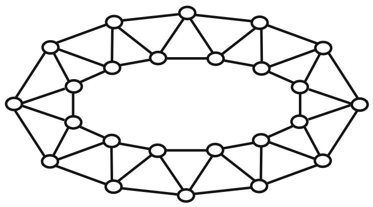


Fig. 4. A connected component with all local neighbourhoods equal to case 3b of Fig. 3

It is not hard to see that such a connected component can be partitioned into triangles if and only if its number of vertices is a multiple of three. Therefore, we can decide that we have a No-instance if this is not the case, and otherwise we can remove it in linear time to obtain an equivalent smaller instance. □

Let a *reduced instance* of Partition Into Triangles on maximum degree four graphs be an instance to which Lemmas 2, 3 and 4 do not apply, i.e., an instance in which each local neighbourhood corresponds to case 2b or 3a in Fig. 3.

Let v be a vertex in a reduced instance whose neighbourhood equals case 3a in Fig. 3. Note that v has one neighbour with the same neighbourhood and it has three neighbours which neighbourhoods are equal to case 2b. We refer to a pair of two vertices with neighbourhood 3a as a *fan*. And, we refer to adjacent series of vertices with the other local neighbourhood as a *cloud* of triangles. See Fig. 5.

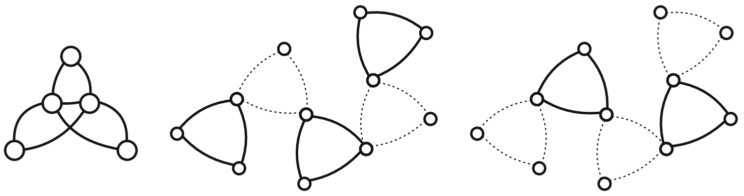


Fig. 5. A fan and a cloud with the two ways in which it can be partitioned into triangles

Observe how these reduced instances can be partitioned into triangles. In a fan, we must select a triangle containing the middle two vertices and exactly one of the three vertices on the boundary. And in a cloud, each triangle is either selected or all its neighbouring (cloud or fan) triangles are selected. Hence, adjacent triangles will alternate between being selected and not being selected in a triangle partition of a cloud; see Fig. 5. As a result, an instance with a cloud that contains a cycle of triangles of odd length is a No-instance since there cannot be such an alternating cycle. Every other cloud has two groups of boundary vertices connecting it to fans: in any solution all fan triangles connected to one group will be selected and all fan triangles connected to the other group will not (see also Fig. 5). The only exception to this is the single vertex cloud that directly connects two fans; here the single vertex is in both groups of endpoints.

Now, the relation between Partition Into Triangles on graphs of maximum degree four and Exact 3-Satisfiability emerges. Namely, we can interpret a reduced instance as an X3SAT instance. We interpret a fan as a clause containing three literals represented by its adjacent clouds: exactly one fan triangle must be selected and this choice determines exactly which triangles in the adjacent clouds will be selected. In this way, we interpret a cloud as a variable that can be set to true or false. Both truth assignments correspond to one of the two possible ways to partition the cloud into triangles. The two groups of vertices on the boundary of a cloud then form the positive and the negative literals; these are contained in the clauses represented by adjacent fans. It is not hard to see that this X3SAT interpretation of a reduced instance is satisfiable if and only if the partition into triangles instance has a solution.

Note that an Exact 3-Satisfiability instance obtained in this way can have multiple identical clauses. Such an instance also satisfies Property 1.

Property 1. For any variable x, the number of positive $f_+(x)$ and negative $f_-(x)$ literals differ a multiple of three.

Proposition 1. *An* Exact 3-Satisfiability *instance obtained in the above way from an instance of* Partition Into Triangles *satisfies Property 1.*

Proof. Let t_+, t_- be the number of triangles selected within the cloud representing x when x is set to true or false, respectively. A cloud has a fixed number of vertices and for each corresponding truth assignment each vertex is either selected in a triangle or part of a corresponding literal, thus: $3t_+ + f_+(x) = 3t_- + f_-(x)$. Hence, $f_+(x) \equiv f_-(x) \pmod 3$. □

The following lemma shows how we can model instances of X3SAT by reduced instances of Partition Into Triangles of maximum degree four.

Lemma 5. *Any variable satisfying Property 1 can be represented by a cloud. Such a cloud consists of* $2f(x) - 3$ *vertices.*

Proof. Consider a cloud representing a variable x, without considering its adjacent fans. Note that we can increase $f_+(x)$ or $f_-(x)$ by 3 in the following way: take a chain of three triangles connected by common endpoints and identify the loose endpoint of the middle triangle with a vertex representing a literal of x.

Without loss of generality let $f_+(x) > 0$. It is not hard to see that in this way we can create any combination $F(x) = (f_+(x), f_-(x))$ given that $f_+(x) \equiv f_-(x) \pmod 3$ by starting from the single vertex cloud with $F(x) = (1,1)$, a single triangle with $F(x) = (3,0)$, two adjacent triangles with $F(x) = (2,2)$, or a chain of four triangles with $F(x) = (3,3)$.

One easily checks that the statement on the number of vertices holds for the initial cases and is maintained every time three triangles are added. □

We conclude by formally expressing the relation between Partition Into Triangles on graphs of maximum degree four and Exact 3-Satisfiability. The proof of the resulting theorem directly follows from the above discussion.

Theorem 2. *There exist linear time transformations between instances of* Partition Into Triangles *on graphs of maximum degree four and instances of* Exact 3-Satisfiability *satisfying Property 1 such that the following holds:*

1. *A given instance is equivalent to its transformed instance.*
2. *An* Exact 3-Satisfiability *instance with variable set X and clause set $\mathcal{C}$ obtained form an n-vertex* Partition Into Triangles *instance of maximum degree four satisfies:* $2|\mathcal{C}| + \sum_{x \in X} (2f(x) - 3) \leq n$.
3. *A* Partition Into Triangles *instance on n vertices obtained form an* Exact 3-Satisfiability *instance satisfying Property 1 with variable set X and clauses set $\mathcal{C}$ satisfies:* $2|\mathcal{C}| + \sum_{x \in X} (2f(x) - 3) = n$.

4 Hardness Results on Graphs of Maximum Degree Four

Having formalised the relation between Partition Into Triangles on graphs of maximum degree four and Exact 3-Satisfiability in the previous section, we are now ready to prove some hardness results. In this section, we will show that Partition Into Triangles on graphs of maximum degree four is $\mathcal{NP}$-complete, and that no subexponential algorithm for this problem exists unless the Exponential Time Hypothesis (ETH) [5] fails.

Theorem 3. Partition Into Triangles *on graphs of maximum degree four is $\mathcal{NP}$-complete.*

Proof. Clearly, the problem is in $\mathcal{NP}$. For hardness, we reduce from the $\mathcal{NP}$-complete problem Exact 3-Satisfiability [4]. Given an instance of Exact 3-Satisfiability, we enforce Property 1 by making three copies of each clause. Then, the result follows from Theorem 2. □

Next, we show that no subexpoential time algorithm for our problem exists. We note that although we prove that, under the ETH, no algorithm subexponential in n exists, this also implies that no algorithm subexponential in m exists as $m = \mathcal{O}(n)$ on bounded degree graphs.

Theorem 4. *Under the ETH, there exists no algorithm for* Partition Into Triangles *on graphs of maximum degree four with a running time subexponential in n.*

Proof. Consider an arbitrary 3SAT instance with m clauses. We create an equivalent X3SAT instance with $4m$ clauses by using an equivalence from [9].

$$\mathrm{SAT}(x,y,z) \Longleftrightarrow \mathrm{XSAT}(x,v_1,v_2) \wedge \mathrm{XSAT}(y,v_2,v_3) \wedge \mathrm{XSAT}(v_1,v_3,v_4) \wedge \mathrm{XSAT}(\neg z,v_2,v_5)$$

We then transform this Exact 3-Satisfiability instance into an equivalent instance of Partition Into Triangles of maximum degree four using the construction in the proof of Theorem 3. This construction triples the number of clauses to $12m$, and thus the total sum of the number of literal occurrences is at most $36m$. By Lemma 5, variables x can be represented by clouds using less than $2f(x)$ vertices each. This gives at most $96m$ vertices: $72m$ for the variables and another $24m$ for the two vertices of a fan for each clause.

Suppose there exists a subexponential time algorithm for Partition Into Triangles on graphs of maximum degree four, i.e, an $\mathcal{O}(2^{\delta n})$ time algorithm for all $\delta > 0$. Then, this algorithm solves 3SAT problems in $\mathcal{O}(2^{\epsilon m})$ for all $\epsilon > 0$ using the above construction and $\delta = \epsilon/96$. However, the sparsification lemma in [5] shows that, assuming the ETH, no such algorithm can exist. □

5 Exponential Time Algorithms

In the previous section, we have given two hardness results for Partition Into Triangles on graphs of maximum degree four. Despite these results, this problems seems to admit very fast, though exponential time, algorithms.

In this extended abstract, we give a simple $\mathcal{O}(1.02445^n)$ time algorithm for this problem based on the algorithm for EXACT SATISFIABILITY by Byskov et al. [3] and the algorithm for EXACT 3-SATISFIABILITY from Wahlström's PhD thesis [11]. We also claim a faster $\mathcal{O}(1.02220^n)$ time algorithm. This algorithm is based on the same principles combined with an extensive analysis; it is given in the full paper [10].

Proposition 2. *There exists an $\mathcal{O}(1.02445^n)$ time algorithm algorithm for* PARTITION INTO TRIANGLES *on graphs of maximum degree four.*

Proof. Use Theorem 2 to obtain an instance of X3SAT with variable set X and clause set $\mathcal{C}$ satisfying $n \geq 2|\mathcal{C}| + \sum_{x \in X} (2f(x) - 3)$. Let γ_1 be the number of variables x with $f_+(x) = f_-(x) = 1$ and let γ_3 be the number of variables x with $f(x) \geq 3$; by Property 1 the total number of variables γ equals $\gamma_1 + \gamma_3$. Since clauses have size three, we find that $n \geq 2(2\gamma_1 + 3\gamma_3)/3 + \gamma_1 + 3\gamma_3 = 2\frac{1}{3}\gamma_1 + 5\gamma_3$.

If $\gamma_1 \leq 0.10746n$, then apply the $\mathcal{O}(1.0984^\gamma)$ algorithm for X3SAT from Wahlström's PhD thesis [11]. Now, $\gamma = \gamma_1 + \gamma_3 \leq 0.10746n + (n - 2\frac{1}{3} \times 0.10746n)/5 < 0.25732n$ by basic calculus. Therefore, the problem is solved in $\mathcal{O}(1.0984^{0.25732n}) = \mathcal{O}(1.02445^n)$ time.

Otherwise $\gamma_1 > 0.10746n$. Then, we apply the $\mathcal{O}(2^{0.2325\gamma})$ XSAT algorithm from Byskov et al. [3]. This algorithm first reduces the instance in polynomial time removing, among others, variables x with $f_+(x) = f_-(x) = 1$, see [3] for details. Hence, the algorithm solves our instance in $\mathcal{O}(2^{0.2325\gamma_3}) = \mathcal{O}(1.02445^n)$ time as $\gamma_3 \leq (n - 2\frac{1}{3} \times 0.10746n)/5 < 0.14986n$ by basic calculus. □

Theorem 5 ([10]). *There exists an $\mathcal{O}(1.02220^n)$ time algorithm algorithm for* PARTITION INTO TRIANGLES *on graphs of maximum degree four.*

6 Conclusion

We have shown that the PARTITION INTO TRIANGLES problem is linear time solvable on graphs of maximum degree three, that it is $\mathcal{NP}$-complete on graphs of maximum degree at least four, and that no subexponential time algorithm for this last problem exists unless the Exponential Time Hypothesis fails. For this seemingly hard problem on graphs of maximum degree four, we have given an efficient $\mathcal{O}(1.0222^n)$ time algorithm using only linear space, and without any large hidden polynomial factors in the running time. In practical situations with reasonable input sizes, this would mean that our algorithm will probably be faster than polynomial time algorithms for the same problem on, for example, graphs whose treewidth is bounded by 10. We would be interested to find more problems on which such fast, yet exponential time, algorithms exists.

We have used an interesting relation between PARTITION INTO TRIANGLES on graphs of maximum degree four and EXACT 3-SATISFIABILITY to obtain these results. This relationship emerges by reducing PARTITION INTO TRIANGLES instances of maximum degree four until each vertex can have only two different local neighbourhoods. Connected series of vertices with one of these local neighbourhoods then form the variables of an EXACT 3-SATISFIABILITY instance and pairs vertices with the other local neighbourhood form the clauses of this Exact 3-Satisfiability instance. Since such a structure seems to disappear on graphs with a higher degree bound, we wonder whether similar ideas could be used for triangle packing or triangle covering.

References

1. Björklund, A.: Exact covers via determinants. In: Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010 (2010) (to appear)
2. Björklund, A., Husfeldt, T., Koivisto, M.: Set partitioning via inclusion-exclusion. SIAM Journal of Computing 39(2), 546–563 (2009)
3. Byskov, J.M., Madsen, B.A., Skjernaa, B.: New algorithms for exact satisfiability. Theoretical Computer Science 332(1-3), 515–541 (2005)
4. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, New York (1979)
5. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? Journal of Computer and System Sciences 63(4), 512–530 (2001)
6. Kann, V.: Maximum bounded 3-dimensional matching is MAX SNP-complete. Information Processing Letters 37(1), 27–35 (1991)
7. Koivisto, M.: Partitioning into sets of bounded cardinality. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 258–263. Springer, Heidelberg (2009)
8. Lipton, R.J.: Gödel's lost letter and P=NP, fast exponential algorithms. Weblog, http://rjlipton.wordpress.com/2009/02/13/polynomial-vs-exponential-time/
9. Schaefer, T.J.: The complexity of satisfiability problems. In: Proceedings of the 10th Annual ACM Symposium on Theory of Computing, STOC 1978, pp. 216–226 (1978)
10. van Rooij, J.M.M., van Kooten Niekerk, M.E., Bodlaender, H.L.: Partitioning sparse graphs into triangles: Relations to exact satisfiability and very fast exponential time algorithms. Technical Report UU-CS-2010-005, Department of Information and Computing Sciences, Utrecht University (2010)
11. Wahlström, M.: Algorithms, measures, and upper bounds for satisfiability and related problems. PhD thesis, Linköping University (2007)
12. Woeginger, G.J.: Exact algorithms for NP-hard problems: A survey. In: Jünger, M., Reinelt, G., Rinaldi, G. (eds.) Combinatorial Optimization - Eureka, You Shrink! LNCS, vol. 2570, pp. 185–207. Springer, Heidelberg (2003)

Author Index