Jan Kratochvíl
Angsheng Li
Jiří Fiala
Petr Kolman (Eds.)

LNCS 6108

Theory and Applications of Models of Computation

7th Annual Conference, TAMC 2010
Prague, Czech Republic, June 2010
Proceedings

Springer

Lecture Notes in Computer Science 6108

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

Editorial Board

David Hutchison
Lancaster University, UK
Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA
Josef Kittler
University of Surrey, Guildford, UK
Jon M. Kleinberg
Cornell University, Ithaca, NY, USA
Alfred Kobsa
University of California, Irvine, CA, USA
Friedemann Mattern
ETH Zurich, Switzerland
John C. Mitchell
Stanford University, CA, USA
Moni Naor
Weizmann Institute of Science, Rehovot, Israel
Oscar Nierstrasz
University of Bern, Switzerland
C. Pandu Rangan
Indian Institute of Technology, Madras, India
Bernhard Steffen
TU Dortmund University, Germany
Madhu Sudan
Microsoft Research, Cambridge, MA, USA
Demetri Terzopoulos
University of California, Los Angeles, CA, USA
Doug Tygar
University of California, Berkeley, CA, USA
Gerhard Weikum
Max-Planck Institute of Computer Science, Saarbruecken, Germany

Jan Kratochvíl Angsheng Li
Jiří Fiala Petr Kolman (Eds.)

Theory and Applications of Models of Computation

7th Annual Conference, TAMC 2010
Prague, Czech Republic, June 7-11, 2010
Proceedings

Springer

Volume Editors

Jan Kratochvíl
Jiří Fiala
Petr Kolman
Charles University
Malostranské nám. 25
11800 Praha 1, Czech Republic
E-mail: {honza; fiala; kolman}@kam.mff.cuni.cz

Angsheng Li
State Key Lab. of Computer Science
Institute of Software
Chinese Academy of Sciences
Beijing 100190, P.R. China
E-mail: angsheng@gcl.iscas.ac.cn

Library of Congress Control Number: 2010927811

CR Subject Classification (1998): F.2, F.3, F.4, G.2.2, H.1.1, E.1, G.4, I.1

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

ISSN	0302-9743
ISBN-10	3-642-13561-7 Springer Berlin Heidelberg New York
ISBN-13	978-3-642-13561-3 Springer Berlin Heidelberg New York

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

springer.com

© Springer-Verlag Berlin Heidelberg 2010
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

The 7th Annual Conference on Theory and Applications of Models of Computation was held during June 7–11, 2010 in Prague. After six successful conferences held in 2004–2009 in China (Beijing, Kunming, Beijing, Shanghai, Xi'an, and ChangSha) TAMC left Asia for the first time, aiming at the "heart of Europe." Changing the geographical location did not bring any changes to the scope of the conference. Its three main themes continued to be Computability, Complexity, and Algorithms. The conference aims to bring together researchers from all over the world with interests in theoretical computer science, algorithmic mathematics, and applications to the physical sciences. This year we saw more participants from Europe and the Americas, but we were very happy that we could also welcome to Prague traditional participants from Asia (China, Japan, and India) to continue enhancing the collaboration among the theoretical computer science communities of these continents.

After hard work the Program Committee decided to accept 35 papers out of 76 submitted to TAMC 2010. Each submission was reviewed by at least three, Program Committee members. All actions of the Program Committee were coordinated via flawlessly and efficiently running EasyChair. We congratulate the authors of accepted contributions and thank all authors who submitted their papers. They all contributed to a successful event.

We extend our thanks to the distinguished plenary speakers who accepted our invitation to deliver plenary talks – John Hopcroft from Cornell University and Shang-Hua Teng from University of Southern California. Their talks "New Research Directions in the Information Age" and "The Laplacian Paradigm: Emerging Algorithms for Massive Graph" were highlights of the conference. We surely do not host and listen to Turing and Harry H. Goode Memorial Awards or Fulkerson and Gödel Prize recipients every day.

It has become a tradition of TAMC conferences to organize special sessions. We would like to thank two prominent Czech mathematicians, Jan Krajíček and Jiří Matoušek, for organizing special sessions on Proof Complexity and Computational Geometry and thus bringing to Prague well-known experts in these areas. And we thank the invited speakers – Olaf Beyersdorff, Stefan Dantchev, Edward A. Hirsch, Sebastian Muller, and Iddo Tzameret (Proof Complexity), and Xavier Goaoc, Christian Knauer, Anastasios Sidiropoulos, and Takeshi Tokuyama (Computational Geometry)—for accepting the invitation to come to TAMC 2010 and share their expertise with the participants of the conference.

We are very grateful to the members of the Program Committee, the external referees, and the members of the Organizing Committee for all the work they did. While all committee members worked well as a team, some names must be singled out: Special thanks go to Ondřej Pangrác for designing the logo of TAMC 2010 and for taking care of the website of the conference, to Tomáš

Dvořák, Petr Gregor, and Martin Mareš for logistic help, and to Jan Kratochvíl Jr. for designing the TAMC 2010 poster. Most of all, we thank Anna Kotěšovcová for running the conference so smoothly.

Finally, we thank the editors at Springer for the cooperation throughout the preparation of this volume. And last but not least we gratefully acknowledge the sponsorship of DIMATIA Charles University and of Czech research grants ITI-1M0545 and MSM0021620838.

June 2010

Jan Kratochvíl
Angsheng Li
Jiří Fiala
Petr Kolman

Conference Organization

Program Co-chairs

Jan Kratochvíl	Charles University, Prague, Czech Republic
Angsheng Li	ISCAS, Beijing, China

Program Committee

Hans L. Bodlaender	Utrecht University, The Netherlands
Wei Chen	Microsoft Research, Beijing, China
Jianer Chen	Texas A&M University, USA
S. Barry Cooper	University of Leeds, UK
Josep Diaz	Universitat Politecnica de Catalunya, Barcelona, Spain
Zhiming Ding	ISCAS, Beijing, China
Rod Downey	Victoria University, Wellington, New Zealand
Jiří Fiala	Charles University, Prague, Czech Republic
Fedor Fomin	University of Bergen, Norway
Lane A. Hemaspaandra	University of Rochester, USA
Dieter Kratsch	University of Metz, France
Antonín Kučera	Masaryk University, Brno, Czech Republic
Antonín Kučera	Charles University, Prague, Czech Republic
Zhyong Liu	ICTCAS, Beijing, China
Mitsunori Ogihara	University of Miami, USA
Sang-il Oum	KAIST, Daejeon, Korea
Andrzej Proskurowski	University of Oregon, Eugene, USA
Xiaoming Sun	Tsinghua University, China
Jun Tarui	University of Electro-Comm, Tokyo, Japan
Chris Umans	Caltech, USA
Peter van Emde Boas	University of Amsterdam, The Netherlands
Lusheng Wang	City University of Hong Kong
Osamu Watanabe	Tokyo Institute of Technology, Japan
Binhai Zhu	Montana State University, USA

Local Organization

Tomáš Dvořák
Jiří Fiala, Chair
Petr Gregor
Petr Kolman
Anna Kotěšovcová
Martin Mareš
Ondřej Pangrác

TAMC Steering Committee

Manindra Agarwal	Indian Institute of Technology, Kanpur, India
Jin-Yi Cai	University of Wisconsin-Madison, USA
S. Barry Cooper	University of Leeds, UK
Angsheng Li	Chinese Academy of Sciences, Beijing, China

External Reviewers

Farid Ablayev
Joergen Bang-Jensen
Ivona Bezáková
Peter Boothe
Tomáš Brázdil
Cyril Brom
Dave Buchfuhrer
Dave Buchfuhrer
Barbora Buhnová
Catherine McCartin
Marek Chrobak
Florin Constantin
Andrea Corradini
Decheng Dai
Lars Eirik Danielsen
Shahar Dobzinski
Yong Dou
Michael Drmota
Martin Dyer
Shahram Esmaeilsabzali
Liya Fan
Arthur Farley
Michael Fellows
Stefan Felsner
Dengguo Feng
Henning Fernau
Felix Fischer
Rusins Freivalds
Serge Gaspers
Qi Ge
Ioannis Giotis
Petr Golovach
Petr Gregor
Alexander Grigoriev
Gregory Gutin
Pinar Heggernes
Danny Hermelin
Nao Hirokawa
Dieter Hofbauer
Clemens Huemer
Tsuyoshi Ito
Akinori Kawachi
Julia Kempe
Torleiv Kløve
Martin Klazar
Takeshi Koshiba
Daniel Král'
Stefan Kratsch
Petr Kučera
Michal Kunc
Steffen Lempp
Angsheng Li
Mathieu Liedloff
Chuang Lin
Lin Liu
Vadim Lozin
Pinyan Lu
Anna Lubiw
Joan Lucas
Jack H. Lutz
Johann Makowsky
Yishay Mansour
Elvira Mayordomo
Kazuyuki Miura
Hiroki Morizumi
Georg Moser
Miguel Mosteiro
Ben Moszkowski
Haiko Müller
Shin-Ichi Nakano
Gonzalo Navarro
Shinya Nishibata

Harumichi Nishimura
Jan Obdržálek
Yoshio Okamoto
Alexander Okhotin
Mikael Onsjö
Martin Otto
Daniël Paulusma
Radek Pelánek
Wlodek Proskurowski
Aleš Pultr
Jimmy Qiao
Stanislaw P. Radziszowski
Julian Rathke
Vojtěch Řehák
Jeffrey Remmel
Peter Rossmanith
Alexis Saurin
Michael Schapira
Jiří Sgall
Jeffrey Shallit
Takeya Shigezumi
Libor Škarvada
Ulrike Stege
Leen Stougie
Martin Strauss
Ondřej Suchý
Pavel Surynek
Guangming Tan
Kazuyuki Tanaka
Gerard Tel
Milan Vlach
Jan Vondrák
Prudence W.H. Wong
Guoping Wang
Yajun Wang
Mathias Weller
Alexander Wilkie
Philipp Woelfel
Jie Wu
Ge Xia
Mingji Xia
Hao Yin
Wei Yu
Janez Žerovnik
Peng Zhang
Shengyu Zhang
Weimin Zheng
Xizhong Zheng

Table of Contents

New Research Directions in the Information Age

John E. Hopcroft

Cornell University, Ithaca
jeh@cs.cornell.edu

Abstract. Computer Science has expanded in may new directions giving rise to a large number of interesting and important research problems. This talk will explore a number of new areas and open problems in these areas that need attention. Areas include tracking the flow of ideas in scientific literature, identifying key papers and how a discipline evolved, extracting information from unstructured data sources, the definition of communities in different types of graphs and the structure and evolution of social networks.

© Springer-Verlag Berlin Heidelberg 2010

The Laplacian Paradigm: Emerging Algorithms for Massive Graphs[⋆]

Shang-Hua Teng

Computer Science Department, University of Southern California
shanghua@usc.edu

Abstract. This presentation describes an emerging paradigm for the design of efficient algorithms for massive graphs. This paradigm, which we will refer to as the *Laplacian Paradigm*, is built on a recent suite of nearly-linear time primitives in spectral graph theory developed by Spielman and Teng, especially their solver for linear systems $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A}$ is the Laplacian matrix of a weighted, undirected n-vertex graph and $\mathbf{b}$ is an n-place vector.

In the Laplacian Paradigm for solving a problem (on a massive graph), we reduce the optimization or computational problem to one or multiple linear algebraic problems that can be solved efficiently by applying the nearly-linear time Laplacian solver. So far, the Laplacian paradigm already has some successes. It has been applied to obtain nearly-linear-time algorithms for applications in semi-supervised learning, image process, web-spam detection, eigenvalue approximation, and for solving elliptic finite element systems. It has also been used to design faster algorithms for generalized lossy flow computation and for random sampling of spanning trees.

The goal of this presentation is to encourage more researchers to consider the use of the Laplacian Paradigm to develop faster algorithms for solving fundamental problems in combinatorial optimization (e.g., the computation of matchings, flows and cuts), in scientific computing (e.g., spectral approximation), in machine learning and data analysis (such as for web-spam detection and social network analysis), and in other applications that involve massive graphs.

1 Nearly-Linear Time Laplacian Primitive

A matrix $\mathbf{L} = (l_{i,j})$ is a *Laplacian matrix* if (1) it is symmetric, i.e., $l_{i,j} = l_{j,i}$ for all i, j, (2) $l_{i,j} < 0$ for all $i \neq j$, and (3) $l_{i,i} = -\sum_{j \neq i} l_{i,j}$ for all i. We can view an $n \times n$ Laplacian matrix as a weighted undirected graph over n vertices.

Let $G = (V, E)$ be a graph with n vertices $V = \{1, ..., n\}$. The *adjacency matrix*, $A(G)$, of a graph $G = (V, E)$ is the $n \times n$ matrix whose (i, j)-th entry is

[⋆] Most materials in this presentation are joint work with Dan Spielman. This presentation also summarizes a recent NSF proposal of the author titled "Nearly-Linear-Time Algorithms for Massive Graphs: Spectral Graph Theory Approach." Therefore, some discussions of this presentation are about proposed work rather than completed work. This research is supported by an NSF grant.

© Springer-Verlag Berlin Heidelberg 2010

1 if $(i, j) \in E$ and 0 otherwise, and the diagonal entries are defined to be 0. Let D be the $n \times n$ *diagonal matrix* with entries $D_{i,i} = d_i$, where d_i is the degree of the ith vertex of G. The *Laplacian*, $L(G)$, of the graph G is defined to be $L(G) = D - A$.

In general, suppose $G = (V, E, w)$ is a weighted undirected n-vertex graph where each edge in $e \in E$ has a weight $w(e) > 0$ and for each $e \notin E$, $w(e) = 0$. Sometime we say w defines the *affinity* between each pair of vertices. We can extend the notion of adjacency matrix $A(G)$, diagonal matrix $D(G)$ and Laplacian matrix $L(G)$ to weighted graphs as following: $A_{i,j}(G) = w(i, j)$ and $D_{i,i}(G) = \sum_{j \neq i} w(i, j)$ and $L(G) = D(G) - A(G)$.

1.1 The Laplacian Primitive and Its Solver

A fundamental problem in numerical analysis and scientific computing is to find a solution to a linear system. Mathematically, we are given an $n \times n$ matrix $\mathbf{A}$ and an n-place vector $\mathbf{b}$ (in the span of $\mathbf{A}$), and are asked to find a vector $\mathbf{x}$ such that $A\mathbf{x} = \mathbf{b}$. In practice, we are often allowed to have a small degree of imprecision. For example, given a precision parameter ϵ, we are asked to produce an $\tilde{\mathbf{x}}$ such that $\left\|\tilde{\mathbf{x}} - \mathbf{A}^{\dagger}\mathbf{b}\right\|_2 \leq \epsilon \left\|\mathbf{A}^{\dagger}\mathbf{b}\right\|_2$, where $\mathbf{A}^{\dagger}$ denotes the Moore-Penrose pseudo-inverse of $\mathbf{A}$—that is the matrix with the same nullspace as $\mathbf{A}$ that acts as the inverse of $\mathbf{A}$ on its image.

We will call the computational problem of solving a linear system defined by a Laplacian matrix the *Laplacian Primitive*.

Definition 1 (Laplacian Primitive). *This primitive concerns linear systems defined by Laplacian matrices.*

> `Input`*: a Laplacian matrix* $\mathbf{L}$ *of dimension* n*, an* n*-place vector* $\mathbf{b} = (b_1, ..., b_n)^T$ *such that* $\sum_i b_i = 0$*, and a precision parameter* $\epsilon > 0$*.*
>
> `Output`*: an* n*-place vector* $\tilde{\mathbf{x}}$ *such that* $\left\|\tilde{\mathbf{x}} - \mathbf{L}^{\dagger}\mathbf{b}\right\|_L \leq \epsilon \left\|\mathbf{L}^{\dagger}\mathbf{b}\right\|_{\mathbf{L}}$*, where for an* n*-place vector* $\mathbf{z}$*, its* $\mathbf{L}$ norm *is defined as* $\left\|\mathbf{z}^T\mathbf{L}\mathbf{z}\right\|_L$*.*

The starting point of the Laplacian Paradigm to be discussed in the next section is the following algorithmic result of Spielman and Teng [50] for solving Laplacian linear systems. In particular, they prove that:

Theorem 1 (Spielman-Teng). *There is a randomized algorithm for the Laplacian primitive that runs in expected time* $m \log^{O(1)} n \log(1/\epsilon)$*, where* n *is the dimension of the Laplacian matrix,* m *is the number of non-zero entries in the Laplacian matrix, and* ϵ *is the precision parameter.*

Note that this result makes no assumption on the structure of the non-zero entries. In fact, the solver of Spielman and Teng applies to every linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ where $\mathbf{A}$ is a symmetric, weakly diagonally dominant matrix. A matrix is *weakly diagonally dominant* if the diagonal entry of each row is at least the 1-norm of the off-diagonal entries of that row.

The Laplacian solver applies the combinatorial preconditioning technique introduced in the pioneering work of Vaidya [55]. It also uses insights and results in the work of Joshi, Reif, Gremban, Miller, Boman, Hendrickson, Maggs, Parekh, Ravi, Woo, Bern, Gilbert, Chen, Nguyen, Toledo [15,14,30,42,28,38].

1.2 A Suite of Nearly-Linear-Time Spectral Algorithms

In the process of developing the nearly linear-time algorithm for the Laplacian primitive, Spielman and Teng and their collaborators designed a suite of nearly linear-time graph algorithms. Most of these algorithms concern the spectral property of graphs. We include these nearly linear-time spectral algorithms as part of the algorithmic primitives in the Laplacian Paradigm.

Clustering and Partitioning: The first family of their spectral algorithms is for clustering and partitioning. A *cluster* of $G = (V, E, w)$ is a subset of V that is richly intra-connected but sparsely connected with the rest of the graph. The quality of a cluster can be measured by its *conductance*, the ratio of the number of its external connections to the number of its total connections.

We let $d(i) = D_{i,i}(G)$, the *weighted degree* of vertex i. For $S \subseteq V$, we define $\mu(S) = \sum_{i \in S} d(i)$. So, $\mu(V) = 2|E|$ if the weights of all edges are equal to 1. Let $E(S, V - S)$ be the set of edges connecting a vertex in S with a vertex in $V - S$. We define the *conductance* of a set of vertices S, written $\Phi(S)$, and the *conductance* of G, respectively by

$$\Phi(S) \stackrel{\text{def}}{=} \frac{|E(S, V - S)|}{\min(\mu(S), \mu(V - S))}, \quad \text{and} \quad \Phi_G \stackrel{\text{def}}{=} \min_{S \subset V} \Phi(S).$$

We also refer to a subset S of V as a *cut* of G and refer to $(S, V - S)$ as a *partition* of G. The *balance* of a cut S or a partition $(S, V - S)$ is then equal to $\mathbf{bal}(S) = \min(\mu(S), \mu(V - S))/\mu(V)$. We call S a *sparsest cut* of G if $\Phi(S) = \Phi_G$ and $\mu(S)/\mu(V) \le 1/2$.

The clustering problem has centered around the following combinatorial optimization problem: Given an undirected graph G and a conductance parameter, find a cluster C such that $\Phi(C) \le \phi$, or determine no such cluster exists. The problem is NP-complete (see, for example [36]). But, approximation algorithms exist. Leighton and Rao [36] used linear programming to obtain $O(\log n)$-approximations of the sparsest cut. Arora, Rao and Vazirani [10] improved this to $O(\sqrt{\log n})$ through semi-definite programming. Faster algorithms obtaining similar guarantees have been constructed by Arora, Hazan and Kale [8], Khandekar, Rao and Vazirani [31], Arora and Kale [9], and Orecchia, Schulman, Vazirani, and Vishnoi [41].

The algorithmic kernel of the Laplacian solver of Spielman and Teng is a local-clustering algorithm, called `Nibble`, for weighted graphs, based on the random walk distributions [49]. The running time of this algorithm is almost linear in the size of the cluster it produces, and is almost independent of the size of the original graph. Although the algorithm may not find a local cluster for some input vertices, it is usually successful:

Theorem 2 (Local Clustering). *There exists a constant $\alpha > 0$ such that for any target conductance ϕ and any cluster C_0 of conductance at most $\alpha{\cdot}\phi^2/\log^3 n$, when given a random vertex v sampled according to degree inside C_0,* `Nibble` *will return a cluster C mostly inside C_0 and with conductance at most ϕ, with probability at least $1/2$.*

Using `Nibble` as a subroutine, Spielman and Teng [49] developed an algorithm called `Partition` and prove the following statement.

Theorem 3 (Nearly Linear-Time Partitioning). *There exists a constant $\alpha > 0$ such that for any graph $G = (V, E)$ that has a cut S of sparsity $\alpha \cdot \theta^2/\log^3 n$ and balance $b \leq 1/2$, with high probability,* `Partition` *finds a cut D with $\Phi_V(D) \leq \theta$ and $\boldsymbol{bal}(D) \geq b/2$. Actually,* `Partition` *satisfies an even stronger guarantee: with high probability either the cut it outputs is well balanced,*

$$\frac{1}{4}\mu(V) \leq \mu(D) \leq \frac{3}{4}\mu(V),$$

or touches most of the edges touching S,

$$\mu(D \cap S) \geq \frac{1}{2}\mu(S).$$

The expected running time of `Partition` *is $O(m\log^7 n/\phi^4)$. Thus, it can be used to quickly find crude cuts.*

Spectral Graph Sparsification: One of the major conceptual developments in the work of [48,51] is a new notion of graph sparsification based on the spectral similarity of graph Laplacians. Let $\mathbf{L}$ be an $n \times n$ a Laplacian matrix. An n-dimensional vector $\mathbf{x} = (x_1, ..., x_n)^T$ is an *eigenvector* of $\mathbf{L}$ if there is a scalar λ such that $\mathbf{L}\mathbf{x} = \lambda\mathbf{x}$. λ is the *eigenvalue* of $\mathbf{L}$ corresponding to the eigenvector $\mathbf{x}$. Because $\mathbf{L}$ is a symmetric matrix, all of its n eigenvalues are real. Notice that the all-1's vector is an eigenvector of any Laplacian matrix and that its associated eigenvalue is 0. Because Laplacian matrices are positive semidefinite, all the other eigenvalues must be non-negative. An important property of weighted Laplacian is:

$$\mathbf{x}^T\mathbf{L}\mathbf{x} = \sum_{i,j} l_{i,j}(x_i - x_j)^2.$$

Graph sparsification is the task of approximating a graph by a sparse graph, and is often useful in the design of efficient approximation algorithms. Several notions of graph sparsification have been proposed. For example, Chew [19] was motivated by proximity problems in computational geometry to introduce graph spanners. Spanners are defined in terms of the *distance similarity* of two graphs: A spanner is a sparse graph where the shortest-path distance between between every pair of vertices is approximately the same in the original graph as in the sparsifier. Motivated by cut problems, Benczur and Karger [13], introduced a notion of sparsification that requires that for every set of vertices, the weight of the edges leaving that set should be approximately the same in the original graph as in the sparsifier.

Motivated by constructing preconditioners, Spielman and Teng introduce a new notion of sparsification called *spectral sparsification* [51]. A *spectral sparsifier* is a subgraph of the original whose Laplacian quadratic form is approximately the same as that of the original graph on all real vector inputs. We say that $\widetilde{G}$ is a *σ-spectral approximation* of G if for all $\mathbf{x} \in \mathbb{R}^V$

$$\frac{1}{\sigma}\mathbf{x}^T L(\widetilde{G})x \leq \mathbf{x}^T L(G)x \leq \sigma \mathbf{x}^T L(\widetilde{G})\mathbf{x}. \tag{1}$$

This notion of sparsification captures the *spectral similarity* between a graph and its sparsifiers. It is a stronger notion than the cut sparsification of Benczur and Karger: the cut-sparsifiers constructed by Benczur and Karger [13] are only required to satisfy these inequalities for all $\mathbf{x} \in \{0,1\}^V$.

In [51], Spielman and Teng prove the following theorem about spectral sparsification with a nearly-linear-time algorithm.

Theorem 4 (Spectral Sparsification). *Given $\epsilon \in (1/n, 1/3)$, $p \in (0, 1/2)$ and a weighted graph G and with n vertices, in expected time $m \log(1/p) \log^{O(1)} n$, one can produce a weighted graph $\widetilde{G}$ that satisfies the following properties:*

a. The edges of $\widetilde{G}$ are a subset of the edges of G; and
b. with probability at least $1-p$, (b.1) $\widetilde{G}$ is a $(1+\epsilon)$-approximation of G, and (b.2) $\widetilde{G}$ has at most $\epsilon^{-2} n \log^{O(1)}(n/p)$ edges.

Low Stretch Spanning Trees: An important discrete mathematical concept in building preconditioners is the low-stretch spanning tree introduced by Alon, Karp, Peleg, and West [2]: Suppose T is spanning tree of $G = (V, E, w)$. For any edge $e \in E$, let $e_1, \cdots, e_k \in F$ be the edges on the unique path in T connecting the endpoints of e. The *stretch* of e w.r.t T is given by $\mathrm{st}_T(e) = w(e)(\sum_{i=1}^{k} \frac{1}{w(e_i)})$. The *average stretch* of the graph G with respect to T is defined by $\mathrm{st}_T(G) = \sum_{e \in E} \mathrm{st}_T(e)/|E|$. Alon *et al* proved that every weighted graph has a spanning tree with average stretch $O(n^{o(1)})$. Elkin, Emek, Spielman, and Teng [23], improved the average stretch to $O(\log^2 n \log\log n)$ with a nearly linear-time construction.

2 The Laplacian Paradigm for Massive Graphs

2.1 Massive Data and Efficient Algorithm Design

In light of the explosive growth in the amount of data and the diversity of computing applications, efficient algorithms are needed now more than ever. We may need to deal with equations and mathematical programming that involve hundreds of millions of variables [43]. We may need to analyze data and graphs such as web logs, social networks, and web graphs that are massive (e.g., of hundreds billions of nodes [29]), complex, and dynamic. As a result of this rapid growth in problem size, what used to be considered an efficient algorithm, such as a $O(n^{1.5})$-time algorithm, may no longer be adequate for solving problems of these scales. Space complexity poses an even greater problem. Thus, the need to design algorithms whose running time is linear or nearly linear in the input size has become increasingly critical.

Linear and Nearly Linear-Time Graph Primitives: Many basic graph-theoretic problems such as connectivity and topological sorting can be solved in linear or nearly-linear time. The efficient algorithms for these problems are built on linear-time frameworks such as Breadth-First-Search (BFS) and Depth-First-Search (DFS). Minimum Spanning Trees (MST), Shortest-Path Trees, and sorting are examples of other commonly used nearly linear-time primitives. However, not every graph problem can be reduced to these primitives in linear or nearly linear time.

Efficient Algorithmic Paradigms: Over the last half century, several algorithmic paradigms have been developed and applied to various problems and applications. Some of these paradigms such as divide-and-conquer, dynamic programming, greedy and local search, linear and convex programming, randomization, and branch-and-bound are commonly covered by textbooks on algorithm design and analysis [20], while some less theoretically-covered paradigms such as the multilevel method, simulated annealing, and the genetic algorithm, are also widely used in practice.

The paradigms such as dynamic programming, linear/convex programming, and branch-and-bound yield polynomial-time algorithms whose complexity is normally not linear or nearly linear — their running time is typically quadratic or cubic or of even higher order — in the input size. But the algorithmic paradigms such as greedy and divide-and-conquer, when they can be successfully applied, usually leads linear- or nearly-linear-time algorithms. In graph theory, several previously-known divide-and-conquer algorithms, run in nearly linear time or use only linear space [24,35]. Their success critically uses the fact that the underlying graphs have a balanced separator that can be found in linear time. Thus, these algorithms can only be applied to special families of graphs, for example planar graphs [35] and nearest neighborhood graphs [40]. However, most graphs such as web graphs and social network graphs simply do not have a balanced separator with the desired quality.

While paradigms such as the multilevel method usually lead to nearly linear-time algorithms in practice, their theoretical behaviors remain widely open and are subjects for excellent research projects.

2.2 The Laplacian Paradigm

The **thesis** behind the Laplacian Paradigm is that the Laplacian primitive, which was not available for previous algorithmic paradigms for graphs, could be a very powerful primitive for combinatorial optimization and numerical analysis.

Unlike the separator-based divide-and-conquer paradigm, this primitive makes no assumption on the structure of the graph. Its complexity depends only on the number of vertices and edges in the underlying graph and the desired precision of the solution. Moreover, its complexity is logarithmic in the reciprocal of the precision.

We conjecture that more graph-theoretical problems can be solved in nearly-linear time using this primitive.

Schematically, to apply the Laplacian Paradigm to solve a problem defined on a graph $G = (V, E, w)$ or a matrix $\mathbf{A}$, we reduce the computational and optimization problem to one or more linear algebraic or spectral graph-theoretic problems whose matrices are Laplacian or Laplacian-like. The nearly-linear-time Laplacian primitive or the primitives from the suite of the spectral algorithms of Section 1.2 is then used to solve these algebraic and spectral problems.

Similar to other algorithmic paradigms, the details of the reduction and resulting algorithm depend on the structure of the application/problem that we need to solve. We now give a two examples of the use of the Laplacian Paradigm.

Example I (Spectral Approximation): Our first example is to approximate the Fiedler value of a weighted graph. Recall that the Fielder value of a weighted graph $G = (V, E, w)$ is the second smallest eigenvalue of $L(G)$.

Definition 2 (Approximate Fiedler Vector and Fiedler Value). *For a Laplacian matrix* $\mathbf{L}$, $\mathbf{v}$ *is an* ϵ*-approximate Fiedler vector if* $\mathbf{v}$ *is orthogonal to the all-1's vector and*

$$\lambda_2(\mathbf{L}) \leq \lambda(\mathbf{v}) = \frac{\mathbf{v}^T\mathbf{L}\mathbf{v}}{\mathbf{v}^T\mathbf{v}} \leq (1+\epsilon)\lambda_2(\mathbf{L}),$$

where $\lambda_2(\mathbf{L})$ *is the Fiedler value of the graph of* $\mathbf{L}$.

To apply the Laplacian Paradigm for computing an approximate Fiedler vector, we use the classic inverse power method. Assume the eigenvalues of $\mathbf{L}$, from the smallest to the largest, are $\lambda_1 = 0$, λ_2, ...,λ_n. Let $\mathbf{v}_i$ be the eigenvector of λ_i. Note $\mathbf{v}_1$ is the all-1's vector.

We choose a unit random vector $\mathbf{r}$ such that $\mathbf{v}_1^T\mathbf{r} = 0$. We can write $\mathbf{r}$ as $\mathbf{r} = \sum_{i=2}^n c_i\mathbf{v}_i$. Note that $\mathbf{L}^\dagger\mathbf{r} = \sum_{i=1}^n c_i\lambda_i^{-1}\mathbf{v}_i$. In general, for positive integer $t \geq 1$, $(\mathbf{L}^\dagger)^t\mathbf{r} = \sum_{i=2}^n c_i\lambda_i^{-t}\mathbf{v}_i$. Therefore, if c_2 is not too small, by choosing $t = \Theta(\log(n/\epsilon)/\epsilon$, assuming we can compute $\mathbf{L}^\dagger\mathbf{r}$ efficiently, we can compute an ϵ-approximate Fiedler vector using the inverse power method.

We can use the nearly-linear-time Laplacian primitive to approximate $\mathbf{L}^\dagger\mathbf{r}$ to a desired precision. With some standard techniques from numerical analysis, one can bound the approximation factor of $(\mathbf{L}^\dagger)^t\mathbf{r}$.

Theorem 5 (Spielman-Teng). *For any* $\epsilon > 0$ *and Laplacian matrix* $\mathbf{L}$, *an* ϵ*-approximate Fiedler vector of* $\mathbf{L}$ *can be computed by a randomized algorithm in time* $m\log^{O(1)} n \log(1/\epsilon)/\epsilon$.

It follows from Mihail [47] that if a graph $G(V, E)$ has a constant maximum degree, then one can obtain a cut of conductance $O(\sqrt{\lambda_2(G)})$ from any approximate Fiedler vector, as guaranteed to exist by Cheeger isoperimetric inequality [18].

Corollary 1 (Cheeger Cut). *If G is a constant-degree graph of n vertices with Fiedler value λ_2, then in nearly linear-time, we can compute a cut of conductance $O(\sqrt{\lambda_2})$.*

Example II (Learning from Labeled Data on a Directed Graph): Our next example is due to Zoom, Huang, and Schölkopf [56]. The problem is to learn from labeled and unlabeled data on a graph: The input of the problem is a strongly connected (aperiodic) directed graph $G = (V, E)$ and a labeling function y that assigns a label from a label set $Y = \{1, -1\}$ to each vertex of a subset $S \subset V$ and 0 to vertices in $V - S$. Let $\mathcal{H}(\mathcal{V})$ be the set of functions of form $V \rightarrow \mathbb{R}$ for labeling vertices in the graph. The mathematical goal of this learning problem is to find a function $f \in \mathcal{H}(\mathcal{V})$ that optimizes the following objective function

$$\textbf{minimize} \ \left(\Omega(f) + \mu||f - y||^2\right), \tag{2}$$

where μ is a constant parameter, and

$$\Omega(f) = \frac{1}{2} \sum_{(u,v)\in E} \pi(u)p(u,v) \left(\frac{f(u)}{\sqrt{\pi(u)}} - \frac{f(v)}{\sqrt{\pi(v)}} \right), \tag{3}$$

and $\pi()$ is the stationary distribution of the random walk on the graph with the transition probability function $p : V \times V \rightarrow \mathbb{R}^+$ defined by the following formula: for each pair $u, v \in V$, if $(u, v) \notin E$, then $p(u, v) = 0$; otherwise $p(u, v) = 1/d^+(u)$ where $d^+(u)$ is the out-degree of u.

Zhou, Huang, and Schölkopf proved that the optimal solution f^* to the mathematical programming defined by (2) is the solution to the following linear system.

$$\left(\Pi - \frac{1}{1+\mu} \frac{\Pi P + P^T \Pi}{2} \right) \left(\Pi^{-1/2} f^* \right) = \left(1 - \frac{1}{1+\mu} \right) \Pi^{1/2} y, \tag{4}$$

where Π the diagonal matrix with $\Pi(v, v) = \pi(v)$ and P is the transition probability matrix defined by $p()$. Using the property that the matrix

$$\mathbf{A} = \left(\Pi - \frac{1}{1+\mu} \frac{\Pi P + P^T \Pi}{2} \right)$$

is symmetric and diagonally dominant, Zhou, Huang, and Schölkopf applied the nearly-linear-time Laplacian primitive to obtain the following result.

Theorem 6 (Zhou, Huang, and Schölkopf). *There exists a randomized algorithm that can solve the graph learning problem given by the mathematical programming defined by (2) in nearly linear time.*

Other Applications of the Laplacian Paradigm: In addition to the two examples given above, the Laplacian paradigm has already been used in several problems in combinatorial optimization and scientific computing.

Boman, Hendrickson, and Vavasis [16] showed that the Laplacian primitive can be used to solve elliptic finite-element systems in nearly linear time. Shklarski-Toledo [44] and Daitch-Spielman [21] extended the solver to systems involving rigidity. Koutis, Miller, and Tolliver [34] presented several applications of the Laplacian Paradigm in vision and image processing. Using the Laplacian primitive. Spielman and Srivastava [46] developed a beautiful nearly linear time algorithm that uses the Laplacian Paradigm to compute the effective resistances in a weighted graph. Using the nearly linear-time Laplacian primitive, Madry and Kelner [39] greatly improved the algorithm for the generation of random spanning trees; Daitch and Spielman [22] gave the fastest known algorithm for computing generalized lossy flows.

2.3 Next Generation Algorithms for Massive Graphs

To support our thesis and excitement that the Laplacian Paradigm may lead to significant advance in graph algorithms, we would like to review the previous linear solvers and their complexity. The straightforward implementation of Gaussian elimination takes $O(n^3)$ time. When m is large relative to n and the matrix is arbitrary, the fastest algorithms for solving linear equations are those based on fast matrix multiplication, which take time approximately $O(n^{2.376})$. The fastest algorithm for solving general sparse positive semi-definite linear systems is the Conjugate Gradient. Used as a direct solver, it runs in time $O(mn)$ (see [54, Theorem 28.3]).

When the linear system is symmetric and sparse, it is standard to represent the non-zero structure of a matrix A by an unweighted graph G_A that has an edge between vertices $i \neq j$ if and only if $A_{i,j}$ is non-zero. If this graph has special structure, there may be elimination orderings that accelerate direct solvers. If A is tri-diagonal, in which case G_A is a path, then a linear system in A can be solved in time $O(n)$. Similarly, when G_A is a tree a linear system in A by be solved in time $O(n)$ If the graph of non-zero entries G_A is planar, one can use Generalized Nested Dissection [25,35,26] to find an elimination ordering under which Cholesky factorization can be performed in time $O(n^{1.5})$ and produces factors with at most $O(n \log n)$ non-zero entries.

For linear equations that arise when solving elliptic partial differential equations, other techniques supply fast algorithms. For example, Multigrid methods may be proved correct when applied to the solution of some of these linear systems [17], and Hierarchical Matrices run in nearly-linear time when the discretization is well-shaped [12].

Before the work of the nearly-linear-time Laplacian primitive, however, no linear solver with complexity better than $O(m^{1.5})$ is known for arbitrary sparse linear systems. So the Laplacian primitive could open a new page for algorithm design.

Then, *which fundamental problems are good candidates for the Laplacian Paradigm?* The family of problems that are closest to the clustering and partitioning problems which are central to the Laplacian primitve and spectral graph sparsification includes matching, s-t flows, and multcommodity flows.

The key step in our attempt to improve the algorithms for these problems is to encode them cleverly by the Laplacian primitive.

3 Final Remarks and Open Questions

Algorithm design is like building a software library. Once we can solve a new problem in linear or nearly linear time, we can add them to our library of efficient algorithms and use them as a subroutine in designing the next wave of algorithms. Because of the several appealing properties of the Laplacian primitive, and our new algorithms for clustering, partitioning, and sparsification, we feel very excited about the new possibilities in the advance of graph algorithms. In fact, each of our algorithms in the suite of the Laplacian Paradigm has been improved since we developed them.

– Using the star-decomposition developed in [23], Abraham, Bartal, and Neiman [1] further improved the average stretch to a quantity smaller than
$$O(\log n O(\log\log n(\log\log\log n)^3)).$$
– The parameters of local clustering algorithm have subsequently been improved by Andersen-Chung-Lang [6] and Andersen and Peres [7]. The former uses personalized Rage-Rank and the latter uses evolving sets to guide the local clustering processing.
– In the original construction, the $O(1)$ in the exponent of $\log^{O(1)}(n/p)$ is quite large (13 for the running time and 29 for the number of edges). Spielman and Srivastava [46] reduced the 29 in the exponent of the number of edges in the spectral sparsifier to 1. The running time of their algorithm, using the Laplacian primitive for computing effective resistances, is nearly linear. Recently, Batson, Spielman, and Srivastava [11] gave a beautiful construction to produce a linearly-sized spectral sparsifier. However, even with polynomial-time complexity, the running time of their algorithm is still far away from being linear.
– Koutis, Miller, and Peng [33] recently improved the running time of the Laplacian solver to $\tilde{O}\left(m\log^2 n\log(1/\epsilon)\right)$.

We would like to conclude this presentation with an exciting algorithmic conjecture and goal that have been driven our research during the last few years.

Conjecture 7 (Laplacian Conjecture: Spielman-Teng). *There exists a (randomized) solver for the Laplacian primitive whose (expected) running time is $O(m\log n\log(1/\epsilon))$.*

The result of Koutis-Miller-Peng is only one $\log n$ factor away from solving this conjecture. In fact, the combination of two recently results, one by Spielman and Woo on preconditioning with low stretch spanning tree, and one by Kolla, Makarychev, Saberi, and Teng [32] implies that after a polynomial-time processing, one can obtain a preconditioner that can solve the Laplacian primitive in time $\tilde{O}(m\log n\log(1/\epsilon)$, where $\tilde{O}$ only hides the ratio of the average stretch of the Abraham-Bartal-Neiman spanning tree to $O(\log n)$, which is $O((\log\log n)^2)$. We are indeed very close to resolve Conjecture 7.

References

1. Abraham, I., Bartal, Y., Neiman, O.: Nearly Tight Low Stretch Spanning Trees. In: FOCS 2008, pp. 781–790 (2008)
2. Alon, N., Karp, R.M., Peleg, D., West, D.: A graph-theoretic game and its application to the k-server problem. SIAM Journal on Computing 24(1), 78–100 (1995)
3. Alpert, C.J., Yao, S.-Z.: Spectral partitioning: the more eigenvectors, the better. In: DAC 1995: Proceedings of the 32nd ACM/IEEE conference on Design automation, pp. 195–200. ACM, New York (1995)
4. Andersen, R., Borgs, C., Chayes, J.T., Hopcroft, J.E., Jain, K., Mirrokni, V.S., Teng, S.-H.: Robust PageRank and locally computable spam detection features. In: Fourth International Workshop on Adversarial Information Retrieval on the Web. ACM International Conference Proceeding Series, pp. 69–76 (2008)
5. Andersen, R., Borgs, C., Chayes, J.T., Hopcraft, J.E., Mirrokni, V.S., Teng, S.-H.: Local computation of pagerank contributions. In: Bonato, A., Chung, F.R.K. (eds.) WAW 2007. LNCS, vol. 4863, pp. 150–165. Springer, Heidelberg (2007)
6. Andersen, R., Chung, F., Lang, K.: Local graph partitioning using pagerank vectors. In: Proceedings: 47th Annual Symposium on Foundations of Computer Science, pp. 475–486 (2006)
7. Andersen, R., Peres, Y.: Finding sparse cuts locally using evolving sets. In: STOC, pp. 235–244 (2009)
8. Arora, S., Hazan, E., Kale, S.: 0(sqrt (log n)) approximation to Sparsest Cut in õ(n^2) time. In: FOCS: IEEE Symposium on Foundations of Computer Science (FOCS), pp. 238–247 (2004)
9. Arora, S., Kale, S.: A combinatorial, primal-dual approach to semidefinite programs. In: STOC 2007: Proceedings of the thirty-ninth annual ACM symposium on Theory of computing, pp. 227–236. ACM, New York (2007)
10. Arora, S., Rao, S., Vazirani, U.: Expander flows, geometric embeddings and graph partitioning. In: STOC 2004: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, pp. 222–231. ACM, New York (2004)
11. Batson, J., Spielman, D.A., Srivastava, N.: Twice-Ramanujan sparsifiers (2008), http://arxiv.org/abs/0808.0163
12. Bebendorf, M., Hackbusch, W.: Existence of H-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients. Numerische Mathematik 95(1), 1–28 (2003)
13. Benczúr, A.A., Karger, D.R.: Approximating s-t minimum cuts in O(n^2) time. In: Proceedings of The Twenty-Eighth Annual ACM Symposium on the Theory of Computing, STOC 1996 (1996)
14. Bern, M., Gilbert, J., Hendrickson, B., Nguyen, N., Toledo, S.: Support-graph preconditioners. SIAM J. Matrix Anal. and Appl. 27(4), 930–951 (2006)
15. Boman, E.G., Hendrickson, B.: Support theory for preconditioning. SIAM Journal on Matrix Analysis and Applications 25(3), 694–717 (2003)
16. Boman, E.G., Hendrickson, B., Vavasis, S.: Solving epplitic finite element systems in nearly-linear time with support preconditioners
17. Briggs, W.L., Henson, V.E., McCormick, S.F.: A Multigrid Tutorial, 2nd edn. SIAM, Philadelphia (2001)
18. Cheeger, J.: A lower bound for smallest eigenvalue of laplacian. In: Gunning, R.C. (ed.) Problems in Analysis, pp. 195–199. Princeton University Press, Princeton (1970)

19. Chew, P.: There is a planar graph almost as good as the complete graph. In: SCG 1986: Proceedings of the second annual symposium on Computational geometry, pp. 169–177. ACM, New York (1986)
20. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 3rd edn.
21. Daitch, S.I., Spielman, D.A.: Support-graph preconditioners for 2-dimensional trusses
22. Daitch, S.I., Spielman, D.A.: Faster approximate lossy generalized flow via interior point algorithms. In: Proceedings of the 40th Annual ACM Symposium on Theory of Computing, pp. 451–460 (2008)
23. Elkin, M., Emek, Y., Spielman, D.A., Teng, S.-H.: Lower-stretch spanning trees. SIAM Journal on Computing 32(2), 608–628 (2008)
24. Frieze, A.M., Miller, G.L., Teng, S.-H.: Separator Based Parallel Divide and Conquer in Computational Geometry. In: SPAA 1992, pp. 420–429 (1992)
25. George, J.A.: Nested dissection of a regular finite element mesh. SIAM J. Numer. Anal. 10, 345–363 (1973)
26. Gilbert, J.R., Tarjan, R.E.: The analysis of a nested dissection algorithm. Numerische Mathematik 50(4), 377–404 (1987)
27. Golub, G.H., Overton, M.: The convergence of inexact Chebychev and Richardson iterative methods for solving linear systems. Numerische Mathematik 53, 571–594 (1988)
28. Gremban, K.: Combinatorial Preconditioners for Sparse, Symmetric, Diagonall y Dominant Linear Systems. PhD thesis, Carnegie Mellon University, CMU-CS-96-123 (1996)
29. Gulli, A., Signorini, A.: The indexable web is more than 11.5 billion pages. In: WWW 2005: Special interest tracks and posters of the 14th international conference on World Wide Web, pp. 902–903. ACM, New York (2005)
30. Joshi, A.: Topics in Optimization and Sparse Linear Systems, Ph.D. thesis, UIUC (1997)
31. Khandekar, R., Rao, S., Vazirani, U.: Graph partitioning using single commodity flows. In: STOC 2006: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing, pp. 385–390. ACM, New York (2006)
32. Kolla, A., Makarychev, Y., Saberi, A., Teng, S.-H.: Subgraph Sparsification. In: STOC 2010 (2010)
33. Koutis, I., Miller, G., Peng, R.: Approaching optimality for solving SDD systems
34. Koutis, I., Miller, G., Tolliver, D.: Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. In: International Symp. of Visual Computing, pp. 1067–1078 (2009)
35. Lipton, R.J., Rose, D.J., Tarjan, R.E.: Generalized nested dissection. SIAM Journal on Numerical Analysis 16(2), 346–358 (1979)
36. Leighton, T., Rao, S.: Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. Journal of the ACM 46(6), 787–832 (1999)
37. Lovasz, L., Simonovits, M.: Random walks in a convex body and an improved volume algorithm. RSA: Random Structures & Algorithms 4, 359–412 (1993)
38. Maggs, B., Miller, G., Parekh, O., Ravi, R., Woo, S.M.: Finding effective support-tree preconditioners. In: ACM SPAA, pp. 176–185 (2005)
39. Madry, A., Kelner, J.: Faster generation of random spanning trees. In: FOCS (2009)
40. Miller, G.L., Teng, S.-H., Thurston, W.P., Vavasis, S.A.: Separators for sphere-packings and nearest neighbor graphs. J. ACM 44(1), 1–29 (1997)

41. Orecchia, L., Schulman, L.J., Vazirani, U.V., Vishnoi, N.K.: On partitioning graphs via single commodity flows. In: STOC 2008: Proceedings of the 40th annual ACM symposium on Theory of computing, pp. 461–470. ACM, New York (2008)
42. Rief, J.: Efficient approximate solution of sparse linear systems. Computer and Mathematics with Applications 36(9), 37–58 (1998)
43. Sharma, A., Liu, X., Miller, P., Nakano, A., Kalia, R.K., Vashishta, P., Zhao, W., Campbell, T.J., Haas, A.: Immersive and interactive exploration of billion-atom systems. In: VR 2002: Proceedings of the IEEE Virtual Reality Conference 2002, p. 217. IEEE Computer Society, Los Alamitos (2002)
44. Shklarski, G., Toledo, S.: Rigidity in finite-element matrices: Sufficient conditions for the rigidity of structures and substructures. SIAM J. Matrix Anal. and Appl. 30(1), 7–40 (2008)
45. Spielman, D.: Graphs and networks: Random walks and spectral graph drawing. Computer Science, Yale (September 18, 2007), `http://www.cs.yale.edu/homes/spielman/462/lect4-07.pdf`
46. Spielman, D.A., Srivastava, N.: Graph sparsification by effective resistances. In: Proceedings of the 40th annual ACM Symposium on Theory of Computing, pp. 563–568 (2008)
47. Spielman, D., Teng, S.-H.: Spectral partitioning works: planar graphs and finite element meshes. In: FOCS 1996: Proceedings of the 37th annual symposium on Foundations of Computer Science, pp. 96–105 (1996)
48. Spielman, D.A., Teng, S.-H.: Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, pp. 81–90 (2003)
49. Spielman, D.A., Teng, S.-H.: A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. CoRR, abs/0809.3232 (2008), `http://arxiv.org/abs/0809.3232`
50. Spielman, D.A., Teng, S.-H.: Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. CoRR, abs/cs/0607105 (2008), `http://www.arxiv.org/abs/cs.NA/0607105`
51. Spielman, D.A., Teng, S.-H.: Spectral sparsification of graphs. CoRR, abs/0808.4134 (2008), `http://arxiv.org/abs/0808.4134`
52. Tolliver, D.A.: Spectral rounding and image segmentation. PhD thesis, Pittsburgh, PA, USA (2006); Adviser-Miller, G.L., Adviser-Collins, R.T.
53. Tolliver, D.A., Miller, G.L.: Graph partitioning by spectral rounding: Applications in image segmentation and clustering. In: CVPR 2006: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 1053–1060. IEEE Computer Society, Los Alamitos (2006)
54. Trefethen, L.N., Bau, D.: Numerical Linear Algebra. SIAM, Philadelphia (1997)
55. Vaidya, P.: Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. An invited at IMA, U. Minnesota (October 1991)
56. Zhou, D., Huang, J., Schölkopf, B.: Learning from Labeled and Unlabeled Data on a Directed Graph. In: The 22nd International Conference on Machine Learning, pp. 1041–1048 (2005)

Proof Complexity of Non-classical Logics

Olaf Beyersdorff

Institute of Computer Science, Humboldt University Berlin, Germany*
beyersdo@informatik.hu-berlin.de

Abstract. Proof complexity is an interdisciplinary area of research utilizing techniques from logic, complexity, and combinatorics towards the main aim of understanding the complexity of theorem proving procedures. Traditionally, propositional proofs have been the main object of investigation in proof complexity. Due their richer expressivity and numerous applications within computer science, also non-classical logics have been intensively studied from a proof complexity perspective in the last decade, and a number of impressive results have been obtained. In this paper we give the first survey of this field concentrating on recent developments in proof complexity of non-classical logics.

1 Propositional Proof Complexity

One of the starting points of propositional proof complexity is the seminal paper of Cook and Reckhow [CR79] where they formalized propositional proof systems as polynomial-time computable functions which have as their range the set of all propositional tautologies. In that paper, Cook and Reckhow also observed a fundamental connection between lengths of proofs and the separation of complexity classes: they showed that there exists a propositional proof system which has polynomial-size proofs for all tautologies (a *polynomially bounded* proof system) if and only if the class NP is closed under complementation. From this observation the so called *Cook-Reckhow programme* was derived which serves as one of the major motivations for propositional proof complexity: to separate NP from coNP (and hence P from NP) it suffices to show super-polynomial lower bounds to the size of proofs in all propositional proof systems.

Although the first super-polynomial lower bound to the lengths of proofs had already been shown by Tseitin in the late 60's for a sub-system of resolution [Tse68], the first major achievement in this programme was made by Haken in 1985 when he showed an exponential lower bound to the proof size in Resolution for a sequence of propositional formulas describing the pigeonhole principle [Hak85]. In the last two decades these lower bounds were extended to a number of further propositional systems such as the Nullstellensatz system [BIK$^+$96], Cutting Planes [BPR97,Pud97], Polynomial Calculus [CEI96,Raz98], or bounded-depth Frege systems [Ajt94,BIK$^+$92,BPI93,KPW95]. For all these

* This paper was written while visiting Sapienza University of Rome under support of DFG grant KO 1053/5-2.

© Springer-Verlag Berlin Heidelberg 2010

proof systems we know exponential lower bounds to the lengths of proofs for concrete sequences of tautologies arising mostly from natural propositional encodings of combinatorial statements.

For proving these lower bounds, a number of generic approaches and general techniques have been developed. Most notably, there is the method of feasible interpolation developed by Krajíček [Kra97], the size-width trade-off introduced by Ben-Sasson and Wigderson [BSW01], and the use of pseudorandom generators in proof complexity [ABSRW04,Kra01,Kra04].

Despite this enormous success many questions still remain open. In particular Frege systems currently form a strong barrier [BBP95], and all current lower bound methods seem to be insufficient for these strong systems. A detailed survey of recent advances in propositional proof complexity is contained in [Seg07].

Let us mention that the separation of complexity classes is not the only motivation for studying lengths of proofs. In particular for strong systems like Frege and its extensions there is a fruitful connection to bounded arithmetic which adds insight to both subjects (cf. [Kra95]). Further, understanding weak systems as Resolution is vital to applications as the design of efficient SAT solvers (see e. g. [PS10] for a more elaborate argument). Last not least, propositional proof complexity has over the years grown into a mature field and many researchers believe that understanding propositional proofs and proving lower bounds—arguably the hardest task in complexity—is a very important and beautiful field of logic which is justified in its own right.

2 Why Non-classical Logics?

Besides the vivid research on propositional proof complexity briefly mentioned in the previous section, the last decade has witnessed intense investigation into the complexity of proofs in non-classical logics. Before describing some of the results, let us comment a bit on the motivation for this research. Arguably, for computer science non-classical logics are even more important than classical logic as they are more expressive and often more suitable for concrete applications. It is therefore quite important to enhance our understanding of theorem proving procedures in these logics, in particular, given the impact that lower bounds to the lengths of proofs have on the performance of proof search algorithms.

Another motivation comes from complexity theory. As non-classical logics are often more expressive than propositional logic, they are usually associated with large complexity classes like PSPACE. The satisfiability problem in the modal logic K was shown to be PSPACE-complete by Ladner [Lad77], and this was subsequently also established for many other modal and intuitionistic logics.[1] Thus, similarly as in the Cook-Reckhow programme mentioned above, proving lower bounds to the lengths of proofs in non-classical logics can be understood

[1] In fact, PSPACE seems to be the "typical" complexity of monomodal logics and similar systems which we will consider here. The complexity often gets higher for logics in richer languages, e. g., PDL or the modal μ-calculus, but I am not aware of any proof complexity research on these, though.

as an attempt to separate complexity classes, but this time we are approaching the NP vs. PSPACE question. Intuitively therefore, lower bounds to the lengths of proofs in non-classical logic should be easier to obtain, as they "only" target at separating NP and PSPACE. In some sense the results of Hrubeš [Hru09] and Jeřábek [Jeř09] on non-classical Frege systems (see Sect. 4) confirm this intuition: they obtain exponential lower bounds for modal and intuitionistic Frege systems (in fact, even extended Frege) whereas to reach such results in propositional proof complexity we have to overcome a strong current barrier [BBP95].

Last not least, research in non-classical proof complexity will also advance our understanding of propositional proofs as we see a number of phenomena which do not appear in classical logic (as e. g. with respect to the question of Frege vs. EF and SF, see Sect. 5). These results are very interesting to contrast with our knowledge on classical Frege as they shed new light on this topic from a different perspective.

3 Proof Systems for Modal and Intuitionistic Logics

We start by introducing some of the relevant proof systems for non-classical logic. While most lower bounds for classical propositional proofs are shown for weak systems like Resolution, Cutting Planes, or Polynomial Calculus, researchers in non-classical logics have mostly investigated Frege style systems. This is quite natural as many modal logics are even defined via derivability in these systems.

Frege systems derive formulas using axioms and rules. In texts on classical logic these systems are usually referred to as Hilbert-style systems, but in proof complexity it has become customary to call them Frege systems [CR79]. A *Frege rule* is a $(k+1)$-tuple $(\varphi_0, \varphi_1, \dots, \varphi_k)$ of formulas such that $\{\varphi_1, \varphi_2, \dots, \varphi_k\} \models \varphi_0$. The standard notation for rules is

$$\frac{\varphi_1 \quad \varphi_2 \quad \dots \quad \varphi_k}{\varphi_0} \ .$$

A Frege rule with $k = 0$ is called a *Frege axiom.* A formula ψ_0 can be derived from formulas $\psi_1, \dots, \psi_k$ by a Frege rule $(\varphi_0, \varphi_1 \dots, \varphi_k)$ if there exists a substitution σ such that $\sigma(\varphi_i) = \psi_i$ for $i = 0, \dots, k$.

Let $\mathcal{F}$ be a finite set of Frege rules. An *$\mathcal{F}$-proof* of a formula φ from a set of formulas Φ is a sequence $\varphi_1, \dots, \varphi_l = \varphi$ of propositional formulas such that for all $i = 1, \dots, l$ one of the following holds:

1. $\varphi_i \in \Phi$ or
2. there exist numbers $1 \le i_1 \le \dots \le i_k < i$ such that φ_i can be derived from $\varphi_{i_1}, \dots, \varphi_{i_k}$ by a Frege rule from $\mathcal{F}$.

A *Frege system* is a set $\mathcal{F}$ of Frege rules which is *implicationally complete*, meaning that for all formulas φ and sets of formulas Φ we have $\Phi \models \varphi$ if and only if there exists an $\mathcal{F}$-proof of φ from Φ.

Every text on classical logic uses its own favourite Frege system, but the actual choice of the rules for the system does not matter (see Sect. 5). Typically, these Frege systems use a number of simple axioms like $p \rightarrow (q \rightarrow p)$ and $(p \rightarrow q) \rightarrow (p \rightarrow (q \rightarrow r)) \rightarrow (p \rightarrow r)$ together with modus ponens

$$\frac{p \quad p \rightarrow q}{q}$$

as its only proper rule.

In addition to the propositional connectives (chosen such that they form a basis for the set of all boolean functions), the *modal language* contains the unary connective $\Box$. As mentioned, non-classical logics are very often defined via an associated Frege system. As an example, a Frege system for the *modal logic* K is obtained by augmenting a propositional Frege system by the modal axiom of distributivity

$$\Box(p \rightarrow q) \rightarrow (\Box p \rightarrow \Box q)$$

and the rule of necessitation

$$\frac{p}{\Box p} \ .$$

The modal logic K can then simply be defined as the set of all modal formulas derivable in this Frege system. Other modal logics can be obtained by adding further axioms, e. g., $K4$ is obtained by adding the axiom $\Box p \rightarrow \Box\Box p$, KB by adding $p \rightarrow \Box\neg\Box\neg p$, and GL by adding $\Box(\Box p \rightarrow p) \rightarrow \Box p$. As two last examples, $S4$ is obtained by extending $K4$ by $\Box p \rightarrow p$ and $S4Grz$ by extending $S4$ by $\Box(\Box(p \rightarrow \Box p) \rightarrow p) \rightarrow \Box p$. For more information on modal logics we refer to [BdRV01] or the thorough introduction in [Jeř09].

While modal logics extend the classical propositional calculus, *intuitionistic logics* are restrictions thereof. We will not define them precisely, but just mention that intuitionistic logic and its superintuitionistic extensions are again defined via Frege systems with a suitable choice of axioms and modus ponens as their only rule (cf. e. g. [Jeř09] for details).

4 Lower Bounds for Modal and Intuitionistic Logics

One of the first topics in proof complexity of non-classical logics was the investigation of the *disjunction property* in intuitionistic logic, stating that if $\varphi \vee \psi$ is an intuitionistic tautology, then either φ or ψ already is. Buss, Mints, and Pudlák [BM99, BP01] showed that this disjunction property even holds in the following feasible form:

Theorem 1 (Buss, Mints, Pudlák [BM99, BP01]). *Intuitionistic logic has the* feasible disjunction property*, i. e., for the standard natural deduction calculus for intuitionistic logic (which is polynomially equivalent to the usual intuitionistic Frege system) there is an algorithm A such that for each proof π of a disjunction $\varphi \vee \psi$, the algorithm A outputs a proof of either φ or ψ in polynomial time in the size of π.*

Subsequently, Ferrari, Fiorentini, and Fiorino [FFF05] extended this result to Frege systems and to further logics such as the modal logic $S4$.

A related property to feasible disjunction is the *feasible interpolation property.* As mentioned in Sect. 1, feasible interpolation is one of the general approaches to lower bounds in proof complexity. This technique was developed by Krajíček [Kra97] and has been successfully applied to show lower bounds for a number of weak systems as Resolution or Cutting Planes (but unfortunately fails for strong systems as Frege systems and their extensions [KP98,BPR00]). For intuitionistic logic, feasible interpolation holds in the following form:

Theorem 2 (Buss, Pudlák [BP01]). *Intuitionistic logic has the* feasible interpolation property*, i. e., from a proof π of an intuitionistic tautology*

$$(p_1 \vee \neg p_1) \wedge \cdots \wedge (p_n \vee \neg p_n) \rightarrow \varphi_0(\bar{p}, \bar{q}) \vee \varphi_1(\bar{p}, \bar{r})$$

using distinct sequences of variables $\bar{p}, \bar{q}, \bar{r}$ (such that $\bar{p} = p_1, \ldots, p_n$ are the common variables of φ_0 and φ_1) we can construct a Boolean circuit C of size $|\pi|^{O(1)}$ such that for each input $\bar{a} \in \{0,1\}^n$, if $C(\bar{a}) = i$, then $\varphi_i(\bar{p}/\bar{a})$ is an intuitionistic tautology (where variables $\bar{p}$ are substituted by $\bar{a}$, and $\bar{q}$ or $\bar{r}$ are still free).

A version of feasible interpolation for some special class of modal formulas was also shown for the modal logic $S4$ by Ferrari, Fiorentini, and Fiorino [FFF05]. Once we have feasible interpolation[2] for a proof system, this immediately implies conditional super-polynomial lower bounds to the proof size in the proof system as in the following theorem:

Theorem 3 (Buss, Pudlák [BP01], Ferrari, Fiorentini, Fiorino [FFF05]). *If* $\mathsf{NP} \cap \mathsf{coNP} \not\subseteq \mathsf{P/poly}$*, then neither intuitionistic Frege systems nor Frege systems for $S4$ are polynomially bounded.*

This method uses the following idea: suppose we *know* that a sequence of formulas $\varphi_0^n \vee \varphi_1^n$ cannot be interpolated by a family of polynomial-size circuits as in Theorem 2. Then the formulas $\varphi_0^n \vee \varphi_1^n$ do not have polynomial-size proofs in any proof system which has feasible interpolation. Such formulas $\varphi_0^n \vee \varphi_1^n$ are easy to construct under suitable assumptions. For instance, the formulas could express that factoring integers is not possible in polynomial time (which implies $\mathsf{NP} \cap \mathsf{coNP} \not\subseteq \mathsf{P/poly}$).

[2] A terminological note (which I owe to Emil Jeřábek): while it became customary to refer to "feasible interpolation" in the context of intuitionistic proof systems, it may be worth a clarification that this is actually a misnomer. Interpolation means that if $\varphi(\bar{p}, \bar{q}) \rightarrow \psi(\bar{p}, \bar{r})$ is provable, where $\bar{p}, \bar{q}, \bar{r}$ are disjoint sequences of variables, then there is a formula $\theta(\bar{p})$ such that $\varphi(\bar{p}, \bar{q}) \rightarrow \theta(\bar{p})$ and $\theta(\bar{p}) \rightarrow \psi(\bar{p}, \bar{r})$ are also provable. In intuitionistic logic, this is a quite different property from the reformulations using disjunction which come from classical logic. What is called "feasible interpolation" for intuitionistic logic (such as in Theorem 2) has nothing to do with interpolation, it is essentially a feasible version of Haldén completeness. Similarly, the modal "feasible interpolation" from [FFF05] is a restricted version of the feasible modal disjunction property.

To improve Theorem 3 to an unconditional lower bound, we need super-polynomial circuit lower bounds for suitable functions, and such lower bounds are only known for restricted classes of Boolean circuits (cf. [Vol99]). One such restricted class consists of all *monotone* Boolean circuits. Razborov [Raz85] and Alon and Boppana [AB87] were able to show exponential lower bounds to the size of monotone circuits which separate the Clique-Colouring pair. The components of this pair contain graphs which are k-colourable or have a clique of size $k+1$, respectively. Clearly, this yields a disjoint NP-pair. The disjointness of the Clique-Colouring pair can be expressed by a sequence of propositional formulas

$$\neg \mathit{Colour}_n^k(\bar{p}, \bar{s}) \vee \neg \mathit{Clique}_n^{k+1}(\bar{p}, \bar{r}) \tag{1}$$

where $\mathit{Colour}_n^k(\bar{p}, \bar{s})$ expresses that the graph encoded in the variables $\bar{p}$ is k-colourable. Similarly, $\mathit{Clique}_n^{k+1}(\bar{p}, \bar{r})$ expresses that the graph specified by $\bar{p}$ contains a clique of size $k+1$.

In order to prove lower bounds for the formulas (1) we need a *monotone feasible interpolation theorem*, i. e., a version of Theorem 2 where the circuits C are monotone. Such a result is known for a number of classical proof systems including Resolution and Cutting Planes, but does not hold for Frege systems under reasonable assumptions (factoring integers is not possible in polynomial time [KP98, BPR00]). Therefore, under the same assumptions we cannot expect a full version of monotone feasible interpolation for modal extensions of the classical Frege system. Note that the above mentioned feasible interpolation theorem of Ferrari et al. [FFF05] also only holds for a restricted class of modal formulas.

Hrubeš [Hru07b, Hru09] had the idea to modify the Clique-Colouring formulas (1) in a clever way by introducing the modal operator $\Box$ in appropriate places to obtain

$$\Box(\neg \mathit{Colour}_n^k(\bar{p}, \bar{s})) \vee \neg \mathit{Clique}_n^{k+1}(\Box\bar{p}, \bar{r}) \tag{2}$$

with $k = \sqrt{n}$. For these formulas he was able to show in [Hru09] that

1. the formulas (2) are modal tautologies;
2. if the formulas (2) are provable in K with $m(n)$ distributivity axioms, then the original formulas (1) can be interpolated by monotone circuits of size $O(m(n)^2)$.

Together these steps yield unconditional lower bounds for modal Frege systems:

Theorem 4 (Hrubeš [Hru07b, Hru09]). *The formulas (2) are K-tautologies. If L is a sublogic of GL or $S4$, then every Frege proof of the formulas (2) in the logic L uses $2^{n^{\Omega(1)}}$ steps.*

The first proof of Theorem 4 in [Hru07b] was obtained by a rather involved model-theoretic argument, but his later paper [Hru09] contains the simplified approach sketched above.

Along the same lines, Hrubeš proved lower bounds for intuitionistic Frege systems. For this he modified the Clique-Colouring formulas to the intuitionistic version

$$\bigwedge_{i=1}^{n}(p_i \vee q_i) \rightarrow (\neg Colour_n^k(\bar{p},\bar{s}) \vee \neg Clique_n^{k+1}(\neg\bar{q},\bar{r}) \tag{3}$$

where again $k = \sqrt{n}$.

Theorem 5 (Hrubeš [Hru07a, Hru09]). *The formulas* (3) *are intuitionistic tautologies and require intuitionistic Frege proofs with* $2^{n^{\Omega(1)}}$ *steps.*

The first proof of Theorem 5 in [Hru07a] was given via a translation of intuitionistic logic into modal logic, but again [Hru09] reproves the result via the simplified approach. Theorem 5 also implies an exponential speed-up of classical logic over intuitionistic logic, because the formulas (3) have polynomial-size classical Frege proofs [Hru07a]. The lower bounds of Theorems 4 and 5 were extended by Jeřábek [Jeř09] to further logics, namely all modal and superintuitionistic logics with infinite branching.

5 Simulations between Non-classical Proof Systems

Besides proving lower bounds a second important topic in proof complexity is the comparison of proof systems via simulations introduced in [CR79] and [KP89] (but see also [PS10] for a new notion). Frege systems and its extensions are one of the most interesting cases in this respect. We recall the definition of polynomial simulations from [CR79]: two proof systems P and Q are *polynomially equivalent* if every P-proof can be transformed in polynomial time into a Q-proof of the same formula, and vice versa. Frege systems also depend on the choice of the language, i. e., the choice of the propositional connectives. When speaking of the polynomial equivalence of two systems over different propositional languages, it is implicitly understood that the formulas are suitably translated into formulas over the new basis (see [PS10] for a discussion). In the classical setting, Cook and Reckhow were able to show the equivalence of all Frege systems using different axioms, rules, and propositional connectives [CR79, Rec76]. For this equivalence to hold, two things have to be verified:

– First, let F_1 and F_2 be two Frege systems using the same propositional language. Then the equivalence of F_1 and F_2 can be shown by deriving every F_1-rule in F_2 and vice versa.
– Second, if F_1 and F_2 are Frege systems over distinct propositional languages L_1 and L_2, respectively, then we have to translate L_1-formulas into L_2-formulas before we can apply the method from the previous item. To still obtain polynomial size formulas after the translation, Reckhow [Rec76] first rebalances the formulas to logarithmic logical depth. In classical propositional logic this is possible by Spira's theorem.

For non-classical logics the situation is more complicated. Rebalancing the formulas to logarithmic depth is not possible because in modal and intuitionistic logic there are examples of formulas which indeed require linear depth. For this reason, the equivalence of modal or intuitionistic Frege systems using different connectives is still open (cf. [Jeř06]).

But even for Frege systems in a fixed language the question is quite intricate because of the presence of *admissible rules*. In general, inference rules

$$R = \frac{\varphi_1 \quad \dots \quad \varphi_k}{\psi}$$

can be classified according to whether they are valid or admissible. The rule R is *valid* in a logic L if $\varphi_1, \dots, \varphi_k \models_L \psi$ where $\models_L$ is the consequence relation of the logic L. The rule R is *admissible* in L if for every substitution σ the following holds: if $\sigma(\varphi_1), \dots, \sigma(\varphi_k)$ are theorems of L, i. e., $\models_L \sigma(\varphi_i)$ holds for $i = 1, \dots, k$, then also $\sigma(\psi)$ is a theorem of L, i. e., $\models_L \sigma(\psi)$. In classical logic, every admissible rule is also valid. But this is not the case in non-classical logic. For instance, in the modal modal logic $K4$ the rule

$$\frac{\Box\varphi}{\varphi}$$

is admissible, but not valid. Admissibility has been thoroughly studied for many non-classical logics. In particular, starting with a question of Friedman [Fri75] it was investigated whether admissibility of a given rule is a decidable property, and this was answered affirmatively for many modal and intuitionistic logics [Ryb97]. In fact, for intuitionistic logic and many important modal logics such as $K4$, GL, $S4$, and $S4Grz$, deciding the admissibility of a given rule is coNEXP-complete as shown by Jeřábek [Jeř07]. Thus this task is presumably even harder than deciding derivability in these logics which is complete for PSPACE.

Let us come back to the above question of the equivalence of all Frege systems for a non-classical logic. If a Frege system uses non-valid admissible rules, then we might not be able to re-derive the rules in another Frege system. Hence, again Reckhow's proof method from the first item above fails. But of course, admissible rules may help to shorten proofs. Luckily, there is a way out. Building on a characterization of admissible rules for intuitionistic logic by Ghilardi [Ghi99], Iemhoff [Iem01] constructed an explicit set of rules which forms a basis for all admissible intuitionistic rules. Using this basis, Mints and Kojevnikov [MK06] were able to prove the equivalence of all intuitionistic Frege systems:

Theorem 6 (Mints, Kojevnikov [MK06]). *All intuitionistic Frege systems in the language $\to, \wedge, \vee, \bot$ are polynomially equivalent.*

Subsequently, Jeřábek [Jeř06] generalized these results to an infinite class of modal logics (so-called extensible logics [Jeř05]). We single out some of the most important instances in the next theorem:

Theorem 7 (Jeřábek [Jeř06]). *Let L be one of the modal logics $K4$, GL, $S4$, or $S4Grz$ and let B be a complete Boolean basis. Then any two Frege systems for L in the language $B \cup \{\Box\}$ are polynomially equivalent.*

We also mention that admissible rules have very recently been studied for many-valued logics by Jeřábek [Jeř10a,Jeř10b].

Another interesting topic is the comparison of Frege systems and their extensions such as extended and substitution Frege systems. *Extended Frege* allows the abbreviation of possibly complex formulas by propositional atoms. *Substitution Frege systems* allow to infer arbitrary substitution instances of a proven formula in one step by the so-called substitution rule. Both these mechanisms might decrease the size of proofs in comparison with Frege, but a separation between these systems is not known for classical propositional logic.

Already in the first paper [CR79] which introduces these systems, Cook and Reckhow observe that substitution Frege polynomially simulates extended Frege, but conjecture that the former might be strictly stronger than the latter. However, in classical propositional logic both systems are indeed polynomially equivalent as was shown independently by Dowd [Dow85] and Krajíček and Pudlák [KP89]. While this proof of equivalence fails in non-classical logics, it is still possible to extract some general information from it as in the next result:

Theorem 8 (Jeřábek [Jeř09]). *For any modal or superintuitionistic logic, extended Frege and tree-like substitution Frege are polynomially equivalent.*

This shows that Cook and Reckhow's intuition on extended vs. substitution Frege was indeed correct and is further confirmed by results of Jeřábek [Jeř09] who shows that going from extended to substitution Frege corresponds to a conservative strengthening of the underlying logic by a new modal operator. Building on these characterizations, Jeřábek exhibits examples for logics where the *EF* vs. *SF* question receives different answers:

Theorem 9 (Jeřábek [Jeř09])

1. *Extended Frege and substitution Frege are polynomially equivalent for all extensions of the modal logic KB.*
2. *Substitution Frege is exponentially better than extended Frege for the modal logic K and for intuitionistic logic.*

The precise meaning of the phrase "exponentially better" is that there are sequences of tautologies which have polynomial-size substitution Frege proofs, but require exponential-size proofs in extended Frege. These sequences are again the Clique-Colour tautologies used by Hrubeš [Hru09]. However, Hrubeš' lower bounds were extended by Jeřábek [Jeř09] to a large class of logics with infinite branching in the underlying Kripke frames, and item 2 of Theorem 9 also holds for these logics.

6 Further Logics and Open Problems

Besides modal and intuitionistic logics there are many other non-classical logics which are interesting to analyse from a proof complexity perspective. One example of such logics are non-monotonic logics of which Reiter's *default logic* [Rei80] is one of the most popular. The semantics and the complexity of default logic have been intensively studied during the last decades (cf. [CS93] for a survey). In particular, Gottlob [Got92] has identified and studied two reasoning tasks for propositional default logic: the *credulous* and the *skeptical* reasoning problem which can be understood as analogues of the classical problems SAT and TAUT. Due to the stronger expressibility of default logic, however, credulous and skeptical reasoning become harder than their classical counterparts—they are complete for the second level Σ_2^{p} and Π_2^{p} of the polynomial hierarchy [Got92].

Elegant sequent calculi were designed for the credulous and skeptical reasoning problems by Bonatti and Olivetti [BO02]. When analysing the proof complexity of these systems it turns out that lower and upper bounds to the proof size in credulous default reasoning and in classical Frege systems are the same up to a polynomial.

Theorem 10 ([BMM⁺10]). *The lengths of proofs in the credulous default calculus and in classical Frege systems are polynomially related. The same holds for the number of steps.*

This means that while the decision complexity of the logic increases, this increase does not manifest in the lengths of proofs. In contrast, for the skeptical default calculus of Bonatti and Olivetti an exponential lower bound to the number of steps applies [BMM⁺10].

A similar result as Theorem 10 was observed by Jeřábek [Jeř09] for tabular modal and superintuitionistic logics which are in coNP. Jeřábek constructs translations of extended Frege proofs in these logics to propositional proofs, thereby obtaining analogous versions of Theorem 10 for extended Frege in these modal and superintuitionistic logics. Thus, the current barrier in classical proof complexity admits natural restatements in terms of non-classical logics.

Let us conclude with some open problems. Besides extending research on proof lengths to further logics, we find the following questions interesting:

Problem 1. So far, research on proof complexity of non-classical logics has concentrated on Frege type systems or their equivalent sequent style formulations. Quite in contrast, many results in classical proof complexity concern systems which are motivated by algebra, geometry, or combinatorics. Can we construct algebraic or geometric proof systems for non-classical logics?

Problem 2. One important tool in the analysis of classically strong systems as Frege systems is their correspondence to weak arithmetic theories, known as bounded arithmetic (cf. [Kra95]). Is there a similar connection between non-classical logics, particularly modal logic, to first-order theories yielding further insight into lengths of proofs questions?

Acknowledgement

I thank Emil Jeřábek for helpful suggestions on an earlier draft of this paper.

References

[AB87] Alon, N., Boppana, R.B.: The monotone circuit complexity of boolean functions. Combinatorica 7(1), 1–22 (1987)

[ABSRW04] Alekhnovich, M., Ben-Sasson, E., Razborov, A.A., Wigderson, A.: Pseudorandom generators in propositional proof complexity. SIAM Journal on Computing 34(1), 67–88 (2004)

[Ajt94] Ajtai, M.: The complexity of the pigeonhole-principle. Combinatorica 14(4), 417–433 (1994)

[BBP95] Bonet, M.L., Buss, S.R., Pitassi, T.: Are there hard examples for Frege systems? In: Clote, P., Remmel, J. (eds.) Feasible Mathematics II, pp. 30–56. Birkhäuser, Basel (1995)

[BdRV01] Blackburn, P., de Rijke, M., Venema, Y.: Modal Logic. Cambridge Tracts in Theoretical Computer Science, vol. 53. Cambridge University Press, Cambridge (2001)

[BIK$^+$92] Beame, P.W., Impagliazzo, R., Krajíček, J., Pitassi, T., Pudlák, P., Woods, A.: Exponential lower bounds for the pigeonhole principle. In: Proc. 24th ACM Symposium on Theory of Computing, pp. 200–220 (1992)

[BIK$^+$96] Beame, P.W., Impagliazzo, R., Krajíček, J., Pitassi, T., Pudlák, P.: Lower bounds on Hilbert's Nullstellensatz and propositional proofs. Proc. London Mathematical Society 73(3), 1–26 (1996)

[BM99] Buss, S.R., Mints, G.: The complexity of the disjunction and existential properties in intuitionistic logic. Annals of Pure and Applied Logic 99(1-3), 93–104 (1999)

[BMM$^+$10] Beyersdorff, O., Meier, A., Müller, S., Thomas, M., Vollmer, H.: Proof complexity of propositional default logic. In: Proc. 13th International Conference on Theory and Applications of Satisfiability Testing. LNCS. Springer, Heidelberg (2010)

[BO02] Bonatti, P.A., Olivetti, N.: Sequent calculi for propositional nonmonotonic logics. ACM Transactions on Computational Logic 3(2), 226–278 (2002)

[BP01] Buss, S.R., Pudlák, P.: On the computational content of intuitionistic propositional proofs. Annals of Pure and Applied Logic 109(1-2), 49–63 (2001)

[BPI93] Beame, P.W., Pitassi, T., Impagliazzo, R.: Exponential lower bounds for the pigeonhole principle. Computational Complexity 3(2), 97–140 (1993)

[BPR97] Bonet, M.L., Pitassi, T., Raz, R.: Lower bounds for cutting planes proofs with small coefficients. The Journal of Symbolic Logic 62(3), 708–728 (1997)

[BPR00] Bonet, M.L., Pitassi, T., Raz, R.: On interpolation and automatization for Frege systems. SIAM Journal on Computing 29(6), 1939–1967 (2000)

[BSW01] Ben-Sasson, E., Wigderson, A.: Short proofs are narrow - resolution made simple. Journal of the ACM 48(2), 149–169 (2001)

[CEI96] Clegg, M., Edmonds, J., Impagliazzo, R.: Using the Groebner basis algorithm to find proofs of unsatisfiability. In: Proc. 28th ACM Symposium on Theory of Computing, pp. 174–183 (1996)

[CR79] Cook, S.A., Reckhow, R.A.: The relative efficiency of propositional proof systems. The Journal of Symbolic Logic 44(1), 36–50 (1979)

[CS93] Cadoli, M., Schaerf, M.: A survey of complexity results for nonmonotonic logics. Journal of Logic Programming 17(2/3&4), 127–160 (1993)

[Dow85] Dowd, M.: Model-theoretic aspects of P$\neq$NP (1985) (unpublished manuscript)

[FFF05] Ferrari, M., Fiorentini, C., Fiorino, G.: On the complexity of the disjunction property in intuitionistic and modal logics. ACM Transactions on Computational Logic 6(3), 519–538 (2005)

[Fri75] Friedman, H.: One hundred and two problems in mathematical logic. The Journal of Symbolic Logic 40(2), 113–129 (1975)

[Ghi99] Ghilardi, S.: Unification in intuitionistic logic. The Journal of Symbolic Logic 64(2), 859–880 (1999)

[Got92] Gottlob, G.: Complexity results for nonmonotonic logics. Journal of Logic and Computation 2(3), 397–425 (1992)

[Hak85] Haken, A.: The intractability of resolution. Theoretical Computer Science 39, 297–308 (1985)

[Hru07a] Hrubeš, P.: A lower bound for intuitionistic logic. Annals of Pure and Applied Logic 146(1), 72–90 (2007)

[Hru07b] Hrubeš, P.: Lower bounds for modal logics. The Journal of Symbolic Logic 72(3), 941–958 (2007)

[Hru09] Hrubeš, P.: On lengths of proofs in non-classical logics. Annals of Pure and Applied Logic 157(2-3), 194–205 (2009)

[Iem01] Iemhoff, R.: On the admissible rules of intuitionistic propositional logic. The Journal of Symbolic Logic 66(1), 281–294 (2001)

[Jeř05] Jeřábek, E.: Admissible rules of modal logics. Journal of Logic and Computation 15(4), 411–431 (2005)

[Jeř06] Jeřábek, E.: Frege systems for extensible modal logics. Annals of Pure and Applied Logic 142, 366–379 (2006)

[Jeř07] Jeřábek, E.: Complexity of admissible rules. Archive for Mathematical Logic 46(2), 73–92 (2007)

[Jeř09] Jeřábek, E.: Substitution Frege and extended Frege proof systems in non-classical logics. Annals of Pure and Applied Logic 159(1-2), 1–48 (2009)

[Jeř10a] Jeřábek, E.: Admissible rules of Łukasiewicz logic. Journal of Logic and Computation (to appear, 2010)

[Jeř10b] Jeřábek, E.: Bases of admissible rules of Łukasiewicz logic. Journal of Logic and Computation (to appear, 2010)

[KP89] Krajíček, J., Pudlák, P.: Propositional proof systems, the consistency of first order theories and the complexity of computations. The Journal of Symbolic Logic 54(3), 1063–1079 (1989)

[KP98] Krajíček, J., Pudlák, P.: Some consequences of cryptographical conjectures for S^1_2 and EF. Information and Computation 140(1), 82–94 (1998)

[KPW95] Krajíček, J., Pudlák, P., Woods, A.: Exponential lower bounds to the size of bounded depth Frege proofs of the pigeonhole principle. Random Structures and Algorithms 7(1), 15–39 (1995)

[Kra95] Krajíček, J.: Bounded Arithmetic, Propositional Logic, and Complexity Theory. Encyclopedia of Mathematics and Its Applications, vol. 60. Cambridge University Press, Cambridge (1995)

[Kra97] Krajíček, J.: Interpolation theorems, lower bounds for proof systems and independence results for bounded arithmetic. The Journal of Symbolic Logic 62(2), 457–486 (1997)

[Kra01] Krajíček, J.: Tautologies from pseudo-random generators. Bulletin of Symbolic Logic 7(2), 197–212 (2001)

[Kra04] Krajíček, J.: Dual weak pigeonhole principle, pseudo-surjective functions, and provability of circuit lower bounds. The Journal of Symbolic Logic 69(1), 265–286 (2004)

[Lad77] Ladner, R.E.: The computational complexity of provability in systems of modal propositional logic. SIAM Journal on Computing 6(3), 467–480 (1977)

[MK06] Mints, G., Kojevnikov, A.: Intuitionistic Frege systems are polynomially equivlalent. Journal of Mathematical Sciences 134(5), 2392–2402 (2006)

[PS10] Pitassi, T., Santhanam, R.: Effectively polynomial simulations. In: Proc. 1st Innovations in Computer Science (2010)

[Pud97] Pudlák, P.: Lower bounds for resolution and cutting planes proofs and monotone computations. The Journal of Symbolic Logic 62(3), 981–998 (1997)

[Raz85] Razborov, A.A.: Lower bounds on the monotone complexity of boolean functions. Doklady Akademii Nauk SSSR 282, 1033–1037 (1985); English translation in: Soviet Math. Doklady 31, 354–357

[Raz98] Razborov, A.A.: Lower bounds for the polynomial calculus. Computational Complexity 7(4), 291–324 (1998)

[Rec76] Reckhow, R.A.: On the lengths of proofs in the propositional calculus. PhD thesis, University of Toronto (1976)

[Rei80] Reiter, R.: A logic for default reasoning. Artificial Intelligence 13, 81–132 (1980)

[Ryb97] Rybakov, V.V.: Admissibility of logical inference rules. Studies in Logic and the Foundations of Mathematics, vol. 136. Elsevier, Amsterdam (1997)

[Seg07] Segerlind, N.: The complexity of propositional proofs. Bulletin of Symbolic Logic 13(4), 417–481 (2007)

[Tse68] Tseitin, G.C.: On the complexity of derivations in propositional calculus. In: Slisenko, A.O. (ed.) Studies in Mathematics and Mathematical Logic, Part II, pp. 115–125 (1968)

[Vol99] Vollmer, H.: Introduction to Circuit Complexity – A Uniform Approach. Texts in Theoretical Computer Science. Springer, Heidelberg (1999)

Optimal Acceptors and Optimal Proof Systems

Edward A. Hirsch[⋆]

Steklov Institute of Mathematics at St. Petersburg,
27 Fontanka, St. Petersburg 191023, Russia
http://logic.pdmi.ras.ru/~hirsch/

Abstract. Unless we resolve the **P** vs **NP** question, we are unable to say whether there is an algorithm (*acceptor*) that accepts Boolean tautologies in polynomial time and does not accept non-tautologies (with no time restriction). Unless we resolve the **co-NP** vs **NP** question, we are unable to say whether there is a proof system that has a polynomial-size proof for every tautology.

In such a situation, it is typical for complexity theorists to search for "universal" objects; here, it could be the "fastest" acceptor (called *optimal acceptor*) and a proof system that has the "shortest" proof (called *optimal proof system*) for every tautology. Neither of these objects is known to the date.

In this survey we review the connections between these questions and generalizations of acceptors and proof systems that lead or may lead to universal objects.

1 Introduction and Basic Definitions

Given a specific problem, does there exist the "fastest" algorithm for it? Does there exist a proof system possessing the "shortest" proofs of the positive instances of the problem? Although the first result in this direction was obtained by Levin [Lev73] in 1970s, these important questions are still open for most interesting languages, for example, the language of propositional tautologies.

Classical version of the problem. According to Cook and Reckhow [CR79], a proof system is a polynomial-time mapping of all strings ("proofs") onto "theorems" (elements of a certain language L; if $L = \texttt{TAUT}$ is the language of all propositional tautologies, the system is called a *propositional* proof system). The existence of a *polynomially bounded* propositional proof system (that is, a system that has a polynomial-size proof for every tautology) is equivalent to $\mathbf{NP} = \mathbf{co\text{-}NP}$. In the context of polynomial boundedness, a proof system can be equivalently viewed as a function that, given a formula and a "proof", verifies in polynomial time that the formula is a tautology: it must accept at least one "proof" for each tautology (*completeness*) and reject all proofs for non-tautologies (*soundness*).

⋆ Partially supported by RFBR grant 08-01-00640, the president of Russia grant "Leading Scientific Schools" NSh-5282.2010.1, and by Federal Target Programme "Scientific and scientific-pedagogical personnel of the innovative Russia" 2009–2013.

© Springer-Verlag Berlin Heidelberg 2010

One proof system $\mathit{\Pi}_w$ is *simulated* by another one $\mathit{\Pi}_s$ if the shortest proof for every tautology in $\mathit{\Pi}_s$ is at most polynomially longer than its shortest proof in $\mathit{\Pi}_w$. The notion of *p-simulation* is similar, but requires also a polynomial-time computable function for translating the proofs from $\mathit{\Pi}_w$ to $\mathit{\Pi}_s$. A *(p-)optimal* propositional proof system is one that (p-)simulates all other propositional proof systems.

The existence of an optimal (or p-optimal) proof system is a major open question for many languages including TAUT. Optimality would imply p-optimality for any system and any language if and only if the natural proof system for SAT (satisfying assignments) is p-optimal; the existence of optimal system would imply the existence of p-optimal system if there is some p-optimal system for SAT [BKM09a]. If an optimal system for TAUT existed, it would allow one to reduce the **NP** vs **co-NP** question to proving proof size bounds for just one proof system. It would also imply the existence of a complete disjoint **NP** pair [Raz94, Pud03]. The existence of a p-optimal system for quantified Boolean formulas would imply a complete language in $\mathbf{NP} \cap \mathbf{co\text{-}NP}$ [Sad97]. (See [BS09] regarding the situation for other languages and complexity classes.) Unfortunately, no concise widely believed structural assumptions (like $\mathbf{NP} \neq \mathbf{co\text{-}NP}$) are known to imply the (non-)existence of (p-)optimal proof systems. Krajíček and Pudlák [KP89] showed that the existence is implied by $\mathbf{NE} = \mathbf{co\text{-}NE}$ (resp., $\mathbf{E} = \mathbf{NE}$) for optimal (resp., p-optimal) propositional proof systems, and Köbler, Messner, and Torán [KMT03] weakened these conjectures to doubly exponential time, but these conjectures are not widely believed.

An *acceptor* for a language L is a semidecision procedure, i.e., an algorithm that answers 1 for $x \in L$ and does not stop for $x \notin L$. An acceptor O is *optimal* if for any other (correct) acceptor A, for every $x \in L$, the acceptor O stops on x in time bounded by a polynomial in $|x|$ and the time taken by $A(x)$. (In [KP89] optimal acceptors are called p-optimal algorithms; the term "acceptor" was later introduced by Messner in his PhD thesis). Krajíček and Pudlák [KP89] showed that for TAUT the existence of a p-optimal system is equivalent to the existence of an optimal acceptor. Then Sadowski [Sad99] proved a similar equivalence for SAT. Finally, Messner [Mes99] gave a different proof of these equivalences extending them to many other languages. We review these results in **Section 3**. Monroe [Mon09] recently formulated a conjecture implying that such an algorithm does not exist[1]. Note that Levin [Lev73] showed that an optimal algorithm does exist for finding witnesses for SAT (equivalently, for non-tautologies): just run all algorithms in parallel and stop as soon as one of them returns a satisfying assignment. However, this procedure gives an optimal acceptor neither for TAUT nor for SAT, because (1) on tautologies, it simply does not stop; (2) Levin's algorithm enumerates *search* algorithms, and for an acceptor we need *decision* algorithms, which may be faster for some inputs; the search-to-decision

[1] More precisely, if an optimal acceptor for TAUT exists, then there is an acceptor for $\overline{\mathtt{BH}}$ (where $\mathtt{BH} = \{(M, x, 1^t) \mid \text{nondeterministic Turing machine } M \text{ accepts } x \text{ in } t \text{ steps}\}$) that works in time polynomial in t for every particular (M, x) such that M does not stop on x.

reduction adds a lot to the running time by running the decision algorithm for *shorter formulas as well*, which may be surprisingly much larger than for the original input.

A proof system Π is *automatizable* if it has an automatization procedure that works in time bounded by a polynomial in the output length. This procedure A, given a "theorem", outputs its (correct) proof for Π of length polynomially bounded by the length of the shortest proof for Π. It is easy to see that such a proof system can be easily turned into an acceptor with running time polynomially related to the proof size and the input size. Vice versa, an acceptor can be converted into an automatizable proof system, where the proof is just the number of steps (written in unary) that the acceptor makes before accepting its input. Thus, in the classical case there is no difference between acceptors and automatizable proof systems.

Extensions that give optimality. An obvious obstacle to constructing an optimal proof system or an optimal acceptor by enumeration is that no efficient procedure is known for enumerating the set of all complete and sound proof systems (resp., acceptors). Recently, similar obstacles were overcome in other settings by considering either computations with *non-uniform advice* (see [FS06] for a survey) or *heuristic* algorithms [FS04, Per07, Its09]. In particular, (p-)optimal proof systems (and acceptors) with advice do exist [CK07], and we review this fact in **Section 4**.

As to heuristic computations, the situation is more complex. Recently, it was proved [HI10] that an optimal randomized heuristic acceptor does exist. However, in the heuristic case we lack the equivalence between optimal proof systems and optimal acceptors, and even the equivalence between acceptors and automatizable proof systems is not straightforward. We review the heuristic case (including more recent directions) in **Section 5**.

Another possibility to obtain optimal proof systems is to generalize the notion of simulation. Recently, Pitassi and Santhanam [PS10] suggested a notion of "effective simulation" and constructed a proof system for quantified Boolean formulas that effectively simulates all other proof systems for QBF. We do not review this result here.

We conclude the paper by listing open questions in **Section 6**.

We now continue to **Section 2** listing trivial facts that we will use in what follows.

2 Trivia

Enumeration. Almost all constructions used in this survey employ the enumeration of all Turing machines of certain kind. Some remarks regarding this follow.

First of all, recall that deterministic Turing machines (either decision machines that say yes/no, or transducers that compute arbitrary functions) can be efficiently enumerated by their Gödel numbers and simulated with only a polynomial overhead in time.

For a recursively enumerable language, one can enumerate acceptors only by running the semidecision procedure in parallel.

Also one can require, if necessary, that the construction of each machine (we will need it for proof systems) includes an "alarm clock" that interrupts the machine after a certain number of steps.

Each proof system Π is p-simulated by another proof system Π' where $\Pi'(x, w)$ runs in time, say, $100(|x| + |w|)^2$: just pad the proofs appropriately; the simulation omits the padding.

These facts allow us to limit ourselves to enumerating proof systems with quadratic alarm clock.

Frequently, we write that we execute "in parallel" a large (sometimes infinite) number of computations. In reality this is achieved, of course, sequentially by alternating consecutive steps of these computations (for example, simulating the step k of the i-th machine at step $(1 + 2k) \cdot 2^i$, as in Levin's optimal algorithm).

Acceptors and proof systems for subsets. For recursively enumerable L, for every acceptor A' for $L' \subseteq L$ there is an acceptor A for L that is almost as efficient on L' as A'. Similarly, for every proof system Π' for $L' \subseteq L$ there is a proof system Π for L that has the same proofs on L' as Π'.

Proofs vs candidate proofs. In what follows we call w a Π-proof of x if $\Pi(x, w) = 1$. Sometimes we write "u is a candidate Π-proof of x" to emphasize that u is intended for checking with Π (while it is not yet known whether $\Pi(x, u) = 1$).

3 Optimal Acceptors Exist iff p-Optimal Proof Systems Exist

3.1 Krajíček-Pudlák's Proof

In this section we give the proof by Krajíček and Pudlák [KP89]. The original exposition demonstrates the equivalence of many statements, while we need only two and show only the required implications.

Theorem 1 ([KP89]). *Optimal acceptors for* TAUT *exist iff p-optimal proof systems for* TAUT *exist. Moreover, the sufficiency ($\Leftarrow$) holds for any language, not just* TAUT.

Proof. $\boxed{\Rightarrow}$. A candidate proof of x for our p-optimal proof system Π_* contains a description of a proof system Π (i.e., a deterministic Turing machine given by its Gödel number and equipped with a quadratic alarm clock) and a candidate Π-proof π. To verify the proof, $\Pi_*(x, (\Pi, \pi))$ simply simulates $\Pi(x, \pi)$ ensuring that Π accepts π and then verifies the correctness of Π by querying the optimal acceptor A_* for the statement

$$\forall y \in \{0,1\}^{|x|}\ \forall \psi \in \{0,1\}^{|\pi|}\ \forall z \in \{0,1\}^{|x|}\ (\Pi(y, \psi) = 0 \vee y[z] = 0)$$

written as a Boolean formula (here $y[z]$ denotes the result of substituting the consecutive bits of z for the consecutive variables of the Boolean formula y).

Since for every Π there is an algorithm that, given $|x|$ and $|\pi|$, writes such statement in time polynomial in $|x| + |\pi|$, A_* must stop in (specific) polynomial time for specific (correct) Π, which proves that Π_* p-simulates Π.

$\boxed{\Leftarrow}$. The optimal acceptor A_* just applies in parallel all deterministic transducers hoping one of them outputs a Π_*-proof and lets Π_* verify the result. Once Π_* returns 1, the acceptor A_* stops.

Since every acceptor A is a particular case of a proof system, there is a polynomial-time algorithm that, given tautology x, outputs Π_*-proofs in time polynomial in $|x|$ and time spent by A on x. □

Remark 1. Sadowski [Sad07] explains that there exist an optimal automatizable proof system for `TAUT` iff there is a (deterministic) acceptor that is optimal for non-deterministic acceptors as well.

3.2 Messner's Proof

Messner [Mes99] generalized the result of Krajíček and Pudlák to a wider class of languages. His proof is also interesting even in the `TAUT` case, because it replaces the statement about the correctness of a proof system on all inputs of certain size by the statement about the correctness of a single proof of a single input.

Definition 1 ([BH77]). *A language L is* paddable *if there is an injective non-length-decreasing polynomial-time padding function* $\mathrm{pad}_L\colon \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *that is polynomial-time invertible on its image and such that for every x and w,*

$$x \in L \iff \mathrm{pad}_L(x, w) \in L.$$

Theorem 2 ([Mes99]). *For every paddable r.e. language L, optimal acceptors for L exist iff p-optimal proof systems for L exist.*

Proof. $\boxed{\Rightarrow}$. Similarly to Krajíček-Pudlák's proof, a candidate proof for our p-optimal proof system Π_* contains a description of a proof system Π (with a quadratic alarm clock) and a candidate Π-proof π, and $\Pi_*(x, (\Pi, \pi))$ starts the verification by simulating $\Pi(x, \pi)$. What makes a difference is how Π_* verifies the correctness of Π.

One could simulate the optimal acceptor A_* on x restricting its running time to a certain polynomial of $|x| + |\pi|$. However, there is no warranty that this amount of time is enough for A_*. Therefore, we run it on a different input where A_* is guaranteed to run in polynomial time and certifies the correctness of the proof π. Namely, we run it on $\mathrm{pad}_L(x, \pi)$. By the definition of pad_L, the result is 1 iff $x \in L$. For a correct proof π, this result is output in a polynomial time because for a correct system Π, the set $\{\mathrm{pad}_L(x, \pi) \mid x \in L,\ \Pi(x, \pi) = 1\} \subseteq L$ can be accepted in a polynomial time (the polynomial is the sum of the time spent by pad_L^{-1} and by Π), and A_* is an optimal acceptor.

$\boxed{\Leftarrow}$. See Theorem 1. □

4 Non-uniform Advice Gives Optimal Proof Systems

Cook and Krajíček [CK07] show that allowing one bit of non-uniform advice yields a p-optimal proof system (against simulations with advice). A similar result for acceptors is not known.

Definition 2. *A proof system with $t(n)$ bits of advice is a polynomial-time algorithm $\Pi(x, w, \alpha)$ and a sequence $(\alpha_n)_{n\in\mathbb{N}}$, where $\alpha_n \in \{0,1\}^{t(n)}$, such that for all x,*

$$x \in L \iff \exists w\ \Pi(x, w, \alpha_{|x|+|w|}) = 1.$$

Theorem 3 ([CK07], see also [BKM09b]). *For every language L, there is a proof system with 1 bit of advice that simulates every proof system with $k(n) = O(\log n)$ bits of advice. Moreover, the simulation can be computed in polynomial time with $k(n)$ bits of advice.*

Proof. A candidate proof for the constructed proof system Π_* contains a description of a proof system Π (a deterministic Turing machine given by its Gödel number and equipped with a quadratic alarm clock) *written in unary* as 1^{Π}, a candidate Π-proof π, and an advice string $\alpha \in \{0,1\}^{k(n)}$ written in unary as a string 1^{α} of length $\leq 2^{k(n)}$. Then $\Pi_*(x, (1^{\Pi}, \pi, 1^{\alpha}))$ starts the verification by simulating $\Pi(x, \pi, \alpha)$. To verify the correctness of Π, the system Π_* simply queries its advice bit, which is supposed to say whether Π with advice string α is correct on all candidate theorems of size $|x|$ and proofs of size $|\pi|$. To ensure that this is the same bit for all couples $(x, (1^{\Pi}, \pi, 1^{\alpha}))$ of the same size, we must choose pairing function such that for all strings $a_1, a_2, a_3, a_4, b_1, b_2, b_3, b_4$, for each i, $|a_i| \neq |b_i| \Rightarrow |(a_1, (a_2, a_3, a_4))| \neq |(b_1, (b_2, b_3, b_4))|$.

The simulation can be computed trivially. □

Remark 2. One may suppose that, similarly to the classical case, the existence of optimal proof systems with advice implies the existence of disjoint **NP** pairs with advice. This is indeed the case; however, in order to keep the closedness under reductions (with advice) the advice given must have length $O(1)$ and not a specific constant number of bits. For exactly one bit of advice one gets only an **NP** pair that is *hard* for disjoint **NP** without advice under many-one reductions without advice [BS09].

5 Heuristic Case: Optimal Acceptors Exist, Hope for Proof Systems

Hirsch and Itsykson [HI10] introduced heuristic[2] acceptors (called heuristic automatizers in [HI10]) and heuristic proof systems. Heuristic algorithms (see, e.g.,

[2] All heuristic computations that we consider are randomized. Therefore, we omit the word "randomized" from the terms that we introduce and mention it explicitly only in the definitions. However, it is possible to consider deterministic heuristic acceptors and proof systems as well.

[BT06]) are algorithms that make errors for a small amount of inputs. Similarly, heuristic proof systems claim a small amount of wrong "theorems".

Since we are interested in the behaviour only on the positive instances ("theorems"), our definition of a distributional problem is different from the usual definition used for heuristic algorithms. Formally, we have a probability distribution concentrated on non-theorems and require that the probability of sampling a non-theorem accepted by an algorithm or validated by a proof system is small.

Definition 3. *We call a pair* (D, L) *a* distributional proving problem *if* D *is a collection of probability distributions* D_n *concentrated on* $\overline{L} \cap \{0,1\}^n$.

In what follows we write $\Pr_{x \leftarrow D_n}$ to denote the probability taken over x from such distribution, while $\Pr_A$ denotes the probability taken over internal random coins used by algorithm A.

5.1 Heuristic Acceptors

Definition 4. *A* heuristic acceptor *for distributional proving problem* (D, L) *is a randomized algorithm* A *with two inputs* $x \in \{0,1\}^*$ *and* $d \in \mathbb{N}$ *that satisfies the following conditions:*

1. *A either outputs* 1 *(denoted* $A(\ldots) = 1$*) or does not halt at all;*
2. *For every* $x \in L$ *and* $d \in \mathbb{N}$, $A(x, d) = 1$.
3. *For every* $n, d \in \mathbb{N}$,

$$\Pr_{r \leftarrow D_n} \left\{ \Pr_A \{A(r, d) = 1\} > \frac{1}{8} \right\} < \frac{1}{d}.$$

(In fact, the specific constant is not important.)

Remark 3. Similarly to the classical case, for recursively enumerable L, conditions 1 and 2 can be easily enforced at the cost of a slight overhead in time by running L's semidecision procedure in parallel.

Given the definition of heuristic acceptor, we now adapt the classical notions of simulation and optimality to the heuristic case and give related basic facts. In what follows, all acceptors are for the same problem (D, L).

Definition 5. *The* time *spent by heuristic acceptor* A *on input* (x, d) *is defined as the median time*

$$t_A(x, d) = \min \left\{ t \in \mathbb{N} \;\middle|\; \Pr_A \{A(x, d) \textit{ stops in time at most } t\} \geq \frac{1}{2} \right\}.$$

Definition 6. *Heuristic acceptor* S *simulates heuristic acceptor* W *if there are polynomials* p *and* q *such that for every* $x \in L$ *and* $d \in \mathbb{N}$,

$$t_S(x, d) \leq \max_{d' \leq q(d \cdot |x|)} p(t_W(x, d') \cdot |x| \cdot d).$$

An optimal *heuristic acceptor is one that simulates every heuristic acceptor.*

Definition 7. *Heuristic acceptor A is* polynomially bounded *if there is a polynomial p such that for every $x \in L$ and every $d \in \mathbb{N}$,*

$$t_A(x, d) \leq p(d \cdot |x|).$$

The following proposition follows directly from the definitions.

Proposition 1

1. *If W is polynomially bounded and is simulated by S, then S is polynomially bounded too.*
2. *An optimal heuristic acceptor is not polynomially bounded if and only if no heuristic acceptor is polynomially bounded.*

Remark 4 (I.Monakhov). If one-way functions exist, then there is a polynomial-time samplable distribution D such that $(D, \texttt{TAUT})$ has no polynomially bounded heuristic acceptor.

Theorem 4 ([HI10]). *Let (D, L) be a distributional proving problem, where L is recursively enumerable and D is polynomial-time samplable, i.e., there is a polynomial-time randomized Turing machine that given 1^n on input outputs x with probability $D_n(x)$ for every $x \in \{0,1\}^n$. Then there exists an optimal heuristic acceptor for (D, L).*

Proof (sketch). The construction consists of three procedures.

The first one, Test, estimates the probability of error of a candidate acceptor A on a given input by repeating A and counting its errors, it accepts if the number of errors is above certain threshold.

The second one, Certify, makes sure that the outermost probability in the correctness condition of a candidate acceptor A is small enough by repeating A at randomly sampled inputs of prescribed size, and counting errors reported by Test.

Finally, the optimal acceptor U, given x and d, runs the following processes for $i \in \{1, \ldots, l(n)\}$, where $l(n)$ is any slowly growing function, in parallel:

1. For certain d', run $A_i(x, d')$, the algorithm with Gödel number i satisfying conditions 1 and 2 of Def. 4, and compute the number of steps T_i made by it before it stops.
2. If Certify accepts A_i executed on inputs of size $|x|$ for at most T_i steps, then output 1 and stop U (all processes).

If none of the processes has stopped, U goes into an infinite loop. □

5.2 Heuristic Proof Systems

Definition 8. *Randomized Turing machine Π is a* heuristic proof system *for distributional proving problem (D, L) if it satisfies the following conditions.*

1. *The running time of $\Pi(x, w, d)$ is bounded by a polynomial in d, $|x|$, and $|w|$.*

2. *(Completeness) For every $x \in L$ and every $d \in \mathbb{N}$, there exists a string w such that* $\Pr\{\Pi(x,w,d)=1\} \geq \frac{1}{2}$*. Every such string w is called a* correct $\Pi^{(d)}$-proof of x.
3. *(Soundness)* $\Pr_{x \leftarrow D_n}\{\exists w : \Pr\{\Pi(x,w,d)=1\} > \frac{1}{8}\} < \frac{1}{d}$.

Unfortunately, we cannot prove the equivalence between acceptors and proof systems in the heuristic case. Therefore, we focus our attention on automatizable heuristic proof systems. Surprisingly, even the equivalence between acceptors and automatizable proof systems is not straightforward in this case.

In [HI10], a heuristic automatizable proof system is defined straightforwardly, and it is shown that it necessarily defines an acceptor taking time at most polynomially larger than the length of the shortest proof in the initial system. This shows that heuristic acceptors form a more general notion than automatizable heuristic proof systems. Neither the converse nor the existence of an optimal automatizable heuristic proof system is shown in [HI10].

However, for a relaxed definition presented below, the equivalence does take place. (The proof of this statement is not published yet [HIS10], so you should take it with a bunch of salt for now.) In particular, there exists a p-optimal system in the class of heuristic automatizable proof systems.

Definition 9. *Heuristic proof system is* automatizable *if there is a randomized Turing machine A satisfying the following conditions.*

1. *For every $x \in L$ and every $d \in \mathbb{N}$, there is a polynomial p such that*

$$\Pr_{w \leftarrow A(x,d)}\left\{|w| \leq p(d \cdot |x| \cdot |w_*|) \wedge \Pr\{\Pi(x,w,d)=1\} \geq \tfrac{1}{4}\right\} \geq \tfrac{1}{4}, \qquad (1)$$

where w_* is the shortest correct $\Pi^{(d)}$-proof of x.

2. *The running time of $A(x,d)$ is bounded by a polynomial in $|x|$, d, and the size of its own output.*

Remark 5. 1. This definition is different from one in [HI10].

2. We do not require the algorithm A to generate correct proofs. It suffices to generate "quasi-correct" (such that $\Pr\{\Pi(x,w,d)=1\} \geq \frac{1}{4}$) with probability $\frac{1}{4}$. At present, we are unable to construct an optimal system or to show the equivalence to heuristic acceptors if $\frac{1}{4}$ is strengthened to $\frac{1}{2}$ as in [HI10].

Definition 10. *We say that heuristic proof system Π_1* simulates *heuristic proof system Π_2 if there exist polynomials p and q such that for every $x \in L$, the shortest correct $\Pi_1^{(d)}$-proof of x has size at most*

$$p(d \cdot |x| \cdot \max_{d' \leq q(d \cdot |x|)}\{\textit{the size of the shortest correct } \Pi_2^{(d')}\textit{-proof of } x\}). \qquad (2)$$

We say that heuristic proof system Π_1 p-simulates[3] *heuristic proof system Π_2 if Π_1 simulates Π_2 and there is a polynomial-time (deterministic) algorithm that*

[3] The definition we give here may be too restrictive: it requires a *deterministic* algorithm that is given a proof for *specific* $d' = q(d \cdot |x|)$. It works for our purposes, but can be relaxed.

converts a $\Pi_2^{q(d\cdot|x|)}$-proof into a $\Pi_1^{(d)}$-proof of size at most (2) such that the probability to accept a new proof is no smaller than the probability to accept the original one.

Definition 11. *Heuristic proof system Π is* polynomially bounded *if there exists a polynomial p such that for every $x \in L$ and every $d \in \mathbb{N}$, the size of the shortest correct $\Pi^{(d)}$-proof of x is bounded by $p(d \cdot |x|)$.*

Proposition 2. *If heuristic proof system Π_1 simulates system Π_2 and Π_2 is polynomially bounded, then Π_1 is also polynomially bounded.*

We now show how heuristic acceptors and automatizable heuristic proof systems are related.

Consider automatizable proof system (Π, A) for distributional proving problem (D, L) with recursively enumerable language L. Let us consider the following algorithm $A_\Pi(x, d)$:

```
1. Execute 1000 copies of $A(x, d)$ in parallel.
   For each copy,
   (a) if it stops with result $w$, then
       – execute $\Pi(x, w, d)$ 60000 times;
       – if there were at least 10000 accepts of $\Pi$ (out of 60000), stop all
         parallel processes and output 1.
2. Execute the enumeration algorithm for $L$; output 1 if this algorithm says
   that $x \in L$; go into an infinite loop otherwise.
```

Proposition 3. *If (Π, A) is a (correct) heuristic automatizable proof system for recursively enumerable language L, then A_Π is a (correct) heuristic acceptor for $x \in L$ and $t_{A_\Pi}(x, d)$ is bounded by polynomial in the size of the shortest correct $\Pi^{(d)}$-proof of x.*

The proof is a routine application of Chernoff bounds, and we skip it. The converse statement is also true. The construction of automatizable heuristic proof system (Π, A) made from heuristic acceptor B is as follows.

$A(x, d)$:

```
1. Run $B(x, d)$.
2. If $B(x, d) = 1$, output $1^T$, where $T$ is the total number of steps made by $B$.
```

$\Pi(x, w, d)$:

```
1. Run $B(x, d)$.
2. If $B$ makes less than $|w|$ steps and outputs a 1, accept. Otherwise, reject.
```

Theorem 5. *(Π, A) is an automatizable heuristic proof system. For every $x \in L$, the length of the shortest Π-proof is upper bounded by the (median) running time of B.*

Corollary 1 (optimal automatizable heuristic proof system). *There is an automatizable heuristic proof system that p-simulates every automatizable heuristic proof system.*

6 Open Questions

Classical systems. It is still unknown whether (p-)optimal propositional proof systems exist. No concise widely believed structural assumption (such as **NP** $\neq$ **co-NP**) is known to imply their nonexistence.

Acceptors with advice. Construct an optimal acceptor with short non-uniform advice either by extending the equivalence between optimal acceptors and p-optimal proof systems or directly. The main obstacle here is the difference between proof systems with "input advice" (as defined above) and proof systems with "output advice" (where an advice string corresponds to the length of input x, not proof w). However, [BKM09b] shows that for propositional proof systems with logarithmic advice, these cases are equivalent w.r.t. the polynomial boundedness.

Heuristic systems. It is challenging to extend the equivalence between optimal acceptors and p-optimal proof systems to the heuristic case, as it would give an optimal heuristic system.

It would be interesting to strengthen the automatizability condition for heuristic systems by requiring to output real proofs (i.e., replace the innermost probability $\frac{1}{4}$ by $\frac{1}{2}$) without losing the equivalence to heuristic acceptors.

It would also be interesting to suggest a notion of a heuristic disjoint **NP** pair and show the existence of a complete pair.

Acknowledgements

The author is very grateful to Olaf Beyersdorff, Dmitry Itsykson, Hunter Monroe, Sergey Nikolenko, Natalia Tsilevich, Zenon Sadowski, and Alexander Smal for valuable comments, and to all participants of the proof complexity seminar in PDMI in September–December 2009 for numerous discussions on the topic.

References

[BH77] Berman, L., Hartmanis, J.: On isomorphisms and density of NP and other complete sets. SIAM Journal on Computing 6(2), 305–322 (1977)

[BKM09a] Beyersdorff, O., Köbler, J., Messner, J.: Nondeterministic functions and the existence of optimal proof systems. Theor. Comput. Sci. 410(38-40), 3839–3855 (2009)

[BKM09b] Beyersdorff, O., Köbler, J., Müller, S.: Nondeterministic instance complexity and proof systems with advice. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 164–175. Springer, Heidelberg (2009)

[BS09] Beyersdorff, O., Sadowski, Z.: Characterizing the existence of optimal proof systems and complete sets for promise classes. In: Frid, A., Morozov, A., Rybalchenko, A., Wagner, K.W. (eds.) CSR 2009. LNCS, vol. 5675, pp. 47–58. Springer, Heidelberg (2009)

[BT06] Bogdanov, A., Trevisan, L.: Average-case complexity. Foundation and Trends in Theoretical Computer Science 2(1), 1–106 (2006)

[CK07] Cook, S.A., Krajíček, J.: Consequences of the provability of $NP \subseteq P/poly$. The Journal of Symbolic Logic 72(4), 1353–1371 (2007)

[CR79] Cook, S.A., Reckhow, R.A.: The relative efficiency of propositional proof systems. The Journal of Symbolic Logic 44(1), 36–50 (1979)

[FS04] Fortnow, L., Santhanam, R.: Hierarchy theorems for probabilistic polynomial time. In: Proceedings of the 45th IEEE Symposium on Foundations of Computer Science, pp. 316–324 (2004)

[FS06] Fortnow, L., Santhanam, R.: Recent work on hierarchies for semantic classes. SIGACT News 37(3), 36–54 (2006)

[HI10] Hirsch, E.A., Itsykson, D.: On optimal heuristic randomized semidecision procedures, with application to proof complexity. In: Proceedings of STACS 2010, pp. 453–464 (2010)

[HIS10] Hirsch, E.A., Itsykson, D., Smal, A.: Optimal heuristic randomized acceptors and automatizable proof systems (March 2010) (unpublished manuscript)

[Its09] Itsykson, D.M.: Structural complexity of AvgBPP. In: Frid, A., Morozov, A., Rybalchenko, A., Wagner, K.W. (eds.) CSR 2009. LNCS, vol. 5675, pp. 155–166. Springer, Heidelberg (2009)

[KMT03] Köbler, J., Messner, J., Torán, J.: Optimal proof systems imply complete sets for promise classes. Inf. Comput. 184(1), 71–92 (2003)

[KP89] Krajíček, J., Pudlák, P.: Propositional proof systems, the consistency of first order theories and the complexity of computations. The Journal of Symbolic Logic 54(3), 1063–1079 (1989)

[Lev73] Levin, L.A.: Universal sequential search problems. Problems of Information Transmission 9, 265–266 (1973) (in Russian); English translation in: Trakhtenbrot, B.A.: A Survey of Russian Approaches to Perebor (Brute-force Search) Algorithms. Annals of the History of Computing 6(4), 384–400 (1984)

[Mes99] Messner, J.: On optimal algorithms and optimal proof systems. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 361–372. Springer, Heidelberg (1999)

[Mon09] Monroe, H.: Speedup for natural problems and coNP?=NP. Technical Report 09-056, Electronic Colloquium on Computational Complexity (2009)

[Per07] Pervyshev, K.: On heuristic time hierarchies. In: Proceedings of the 22nd IEEE Conference on Computational Complexity, pp. 347–358 (2007)

[PS10] Pitassi, T., Santhanam, R.: Effectively polynomial simulations. In: Proceedings of ICS 2010 (2010)

[Pud03] Pudlák, P.: On reducibility and symmetry of disjoint NP pairs. Theoretical Computer Science 295(1-3), 323–339 (2003)

[Raz94] Razborov, A.A.: On provably disjoint NP-pairs. Technical Report 94-006, Electronic Colloquium on Computational Complexity (1994)

[Sad97] Sadowski, Z.: On an optimal quantified propositional proof system and a complete language for NP∩co-NP. In: Chlebus, B.S., Czaja, L. (eds.) FCT 1997. LNCS, vol. 1279, pp. 423–428. Springer, Heidelberg (1997)

[Sad99] Sadowski, Z.: On an optimal deterministic algorithm for SAT. In: Gottlob, G., Grandjean, E., Seyr, K. (eds.) CSL 1998. LNCS, vol. 1584, pp. 179–187. Springer, Heidelberg (1999)

[Sad07] Sadowski, Z.: Optimal proof systems, optimal acceptors and recursive representability. Fundamenta Informaticae, 169–185 (2007)

The Complexity of Geometric Problems in High Dimension[⋆]

Christian Knauer

Institut für Informatik, Universität Bayreuth, Germany

Abstract. Many important NP-hard geometric problems in $\mathbb{R}^d$ are trivially solvable in time $n^{O(d)}$ (where n is the size of the input), but such a time dependency quickly becomes intractable for higher-dimensional data, and thus it is interesting to ask whether the dependency on d can be mildened. We try to adress this question by applying techniques from parameterized complexity theory.

More precisely, we describe two different approaches to show parameterized intractability of such problems: An "established" framework that gives fpt-reductions from the k-clique problem to a large class of geometric problems in $\mathbb{R}^d$, and a different new approach that gives fpt-reductions from the k-Sum problem.

While the second approach seems conceptually simpler, the first approach often yields stronger results, in that it further implies that the d-dimensional problems reduced to cannot be solved in time $n^{o(d)}$, unless the Exponential-Time Hypothesis (ETH) is false.

Keywords: parameterized complexity, geometric dimension, lower bounds, exponential-time hypothesis.

1 Introduction

The algorithm of Megiddo [27] for linear programming in $\mathbb{R}^d$ with n constraints runs in $T_{LP}(n, d) = O(2^{2^d} n)$ time. When d is bounded, linear programming can be solved in strongly polynomial time, i.e., time that depends only on n and not on the binary representation of the input; when d is unbounded, only weakly polynomial time algorithms are known. What about geometric problems in $\mathbb{R}^d$ that are NP-hard when the dimension d is unbounded? For many such problems like, e.g., the problem of computing the minimum enclosing cylinder of a set of n points in $\mathbb{R}^d$ [29], the problem of separating two n point sets in $\mathbb{R}^d$ by two hyperplanes [28], or the problem of deciding whether the convex hull of n points in $\mathbb{R}^d$ is simplicial [17], all known algorithms run in $n^{\Theta(d)}$ time. It is widely conjectured that the linear dependence on d cannot be removed from the exponent of n. However little evidence of this has been given so far.

[⋆] This research was supported by the German Science Foundation (DFG) under grant Kn 591/3-1.

© Springer-Verlag Berlin Heidelberg 2010

Parameterized complexity theory provides a framework for the study of algorithmic problems by measuring their complexity in terms of one or more parameters, explicitly or implicitly given by their underlying structure, in addition to the problem input size. The main idea is to devise exponential-time algorithms for NP-hard problems, confining exponentiality to the parameters (or to show that such algorithms do not exist). For an introduction to the field of parameterized complexity theory, we refer to the recent textbooks by [19] and [30], in addition to the first, classic work of [14]. For completeness we review some basic definitions: A problem with input instance of size n and with a non-negative integer parameter k is *fixed-parameter tractable* (fpt) if it can be solved by an algorithm that runs in $O(f(k) \cdot n^c)$ time, where f is a computable function depending only on k and c is a constant independent of k; such an algorithm is called an fpt-algorithm and it is (informally) said to run in fpt-time. The class of all fixed-parameter tractable problems, denoted by FPT, is the parameterized complexity analog of P. Megiddo's algorithm for linear programming in $\mathbb{R}^d$ mentioned above is an example of an fpt-algorithm (with the dimension d being the parameter)[1]. Parameterized complexity theory provides a set of general algorithmic techniques for proving fixed-parameter tractability that have been successfuly used for a variety of parameterized algorithmic problems in graph theory, logic, and computational biology. More importantly for our purposes, the theory also provides a framework for establishing fixed-parameter *intractability*. To this end, an (infinite) hierarchy of complexity classes has been introduced, the W-hierarchy, with FPT being its lowest class. Its first level, W[1], can be thought of as the parameterized analog of NP. Hardness is sought via *fpt-reductions*, i.e., a fpt-time many-one mapping from a problem Π, parameterized with k, to a problem Π', parameterized with k', such that $k' \leq g(k)$ for some computable function g. These reductions preserve fixed-parameter tractability between parameterized problems. Hardness for some level of the hierarchy can be thought as the parameterized complexity analog of NP-hardness, and, as in classical complexity theory, intractability results are conditional and, thus, serve as relative lower bounds. The working assumption for parameterized complexity is that all levels of the W-hierarchy are pairwise distinct. For example, a problem that is W[1]-hard for some parameterization is not fixed-parameter tractable for this parameterization unless FPT=W[1] (which is considered highly unlikely under current standard complexity theoretic assumptions).

The dimension of geometric problems in $\mathbb{R}^d$ is a natural parameter for studying their parameterized complexity. In terms of parameterized complexity theory the question alluded to above is whether any of these problems is fixed-parameter tractable with respect to d; note that NP-hardness for a problem does not exclude such a possibility. Proving any of these problems to be W[1]-hard with respect to d, gives a strong evidence that an fpt-algorithm does not exist, under standard complexity theoretic assumptions.

[1] This is of course not surprising, since the linear programming problem is in fact in P. The fact that the NP-hard *integer* linear programming problem is FPT when parameterized by the dimension may be somewhat more interesting [19,30].

W[1]-hardness is often established by fpt-reductions from the k-clique problem in general graphs (where the parameter k measures the size of the clique), which is known to be W[1]-complete [14]. In section 2 we describe a framework that gives such fpt-reductions for a surprisingly large class of geometric problems in $\mathbb{R}^d$ (parameterized by the dimension d). These reductions often have an additional property which implies that the problems considered cannot be solved in time $n^{o(d)}$, unless the Exponential-Time Hypothesis (ETH) [24] is false[2]: As was show in [13,12], the k-clique problem takes time $n^{\Omega(k)}$ unless ETH fails to hold. These papers also establish that, if the k-clique problem is reducible to a problem Q by an fpt-reduction in which the parameter d is linearly bounded by k (a feature that many reductions that follow the framework have), then the problem Q takes time $n^{\Omega(d)}$ unless ETH fails.

In section 3 we describe a different approach to obtain W[1]-hardness results for geometric problems parameterized by the dimension by giving fpt-reductions from the k-Sum problem [20,17] (a variant of the subset-sum problem) which was shown to be W[1]-hard by Downey and Koblitz [18]. These reductions are conceptually much simpler but "weaker" in the sense that the reduction of [18] transforms an instance of the k-clique problem into a $O(k^2)$-Sum instance. This only implies that geometric problems in $\mathbb{R}^d$ that are reduced from k-Sum such that $d = O(k)$ cannot be solved in time $n^{o(\sqrt{d})}$ (assuming the ETH).

At this point the reader should be *cautious*: many algorithmic results in computational geometry assume the real RAM, a much stronger model of computation than the standard Turing machine adopted in parameterized complexity theory. Extending the theory to accomodate other models of computation is a current research issue and one has to be careful before claiming (in)tractability of geometric problems. To make geometric reductions suitable for the Turing machine model, the data must often be perturbed using fixed-precision roundings (using only polynomially many bits, of course). We will ingore this issue in the following and refer to [8,7] where such a rounding procedure is described and analyzed in detail.

Related independent work. The dimension of geometric problems is a natural parameter for studying their parameterized complexity. However, apart from the classic results on linear (integer) programming mentioned above and the body of our own work [6,8,7,21,22] which is described later in some more detail, there are only few other results of this type: Langerman and Morin [25] gave fixed-parameter tractability results for the problem of covering points with hyperplanes, while the problems of computing the volume of the union of axis parallel boxes has been shown to be W[1]-hard by Chan [9]. The problem of linear programming with violations (given a system of n linear inequalities with d variables, decide whether l inequalities can be removed so that the system becomes feasible) was shown to be fixed-parameter tractable when parameterized with both d *and* l. In fact it can be solved in $O(l^d T_{LP}(n,d))$ time [26], and in $O(d^l T_{LP}(n,d))$ time [5].

[2] The Exponential Time Hypothesis conjectures that n-variable 3-CNFSAT cannot be solved in $2^{o(n)}$-time.

2 The Scaffold-Constraint Technique

W[1]-hardness is often established by fpt-reductions from the k-clique problem in general graphs (where the parameter k measures the size of the clique), which is known to be W[1]-complete [14]. In this section we describe a framework pioneered in Cabello et al. [8,7] that gives such fpt-reductions for a surprisingly large class of geometric problems in $\mathbb{R}^d$ when they are parameterized by the dimension d.

2.1 Methodology

In order to show W[1]-hardness of a geometric decision problem Π in $\mathbb{R}^d$ (parameterized with d), we will use a reduction from the W[1]-hard k-clique problem: Given a graph G, decide if it has a clique of size k. We first construct a *scaffolding* structure that restricts the solutions of Π to n^k combinatorially different solutions, which can be interpreted as potential k-cliques in a graph with n vertices. Additional *constraint* objects will then encode the edges of the input graph G and "cancel out" some of the potential k-cliques. Of course we have to be careful to construct geometric instances which lie in Euclidean space whose dimension depends only on k, i.e., we have to make sure that $d = f(k)$ for some arbitrary (but computable) function f. The "milder" the dependence of d on k, the better the lower bound we get from the hardness result. In many cases the dependence is linear. The scaffolding structure is usually highly symmetric. It is composed of k symmetric subsets of a polynomial (in n) number of constraint objects that lie in *orthogonal* subspaces. Orthogonality together with the specific geometric properties of the problem Π then often allows us to restrict the number of potential solutions of Π to n^k *combinatorially different* solutions; these in turn correspond to k-subsets of the vertex set of G, i.e., potential k-cliques of G. The way of placing the constraint objects is crucial: each such object lies in a 4-dimensional (or 2-dimensional) subspace and "cancels out" an exponential number of potential solutions; these correspond to k-subsets of the vertex set of G that do not form a k-clique in G.

2.2 An Example: Maximum-Size Feasible Subsystem

We illustrate the technique with the problem of computing the maximum-size feasible subsystem of a system of linear inequalities with d variables (this reduction has been described in [21]):

– Given a system of n linear inequalities with d variables and an integer l, decide whether there is a solution satisfying l of the inequalities.

This problem is the same as the problem of computing the depth of an arrangement of a set $\mathcal{H}$ of n halfspaces in $\mathbb{R}^d$ (which is the maximum number of halfspaces in $\mathcal{H}$ a point of $\mathbb{R}^d$ can be contained in). It also equivalent (in terms of standard complexity theory) to the problem of linear programming with violatons [26]:

– Given a system of n linear inequalities with d variables and an integer l, decide whether l inequalities can be removed so that the remaining system becomes feasible.

The classical complexity of the maximum-size feasible subsystem problem was studied in Amaldi and Kann [1]. Several results on the hardness of approximability (it is basically as hard to approximate as the clique problem) can also be found in this paper, as well as in Arora et al. [3]. Both papers reduce from k-vertex cover (which is fpt wrt. k) and create instances in $O(n)$-dimensional space. For exact and approximation algorithms for this and several related problems see Aronov and Har-Peled [2]. The paper of Arora et al. [3] also establishes the hardness of approximability for the problem of linear programming with violations. As mentioned above, from a parameterized complexity point of view, the problem was shown to be fixed-parameter tractable when parameterized with both d *and* l.

We first consider the following problem: Given a system of linear *equations* find a solution that satisfies as many equations as possible. The decision version of this problem is as follows: Given a set of n hyperplanes in $\mathbb{R}^d$ and an integer l, decide whether there exists a point in $\mathbb{R}^d$ that is covered by at least l of the hyperplanes. For $l = d+1$ this problem is the dual of the affine degeneracy-detection problem we will consider in section 3; there we will give a conceptually much simpler W[1]-hardness proof (which unfortunately gives a somewhat "weaker" result when assuming the ETH).

In the following, $\boldsymbol{x} = (x_1, \ldots, x_k) \in \mathbb{R}^k$ denotes a k-dimensional vector. Let $[n] = \{1, \ldots, n\}$. We identify the grid $[n]^k$ with the set of vectors in $\mathbb{R}^k$ with integer coordinates in $[n]$. For a set $\mathcal{H}$ of hyperplanes in $\mathbb{R}^k$ and a point $\boldsymbol{x} \in \mathbb{R}^k$ we define

$$\operatorname{depth}(\boldsymbol{x}, \mathcal{H}) = |\{h \in \mathcal{H} \mid \boldsymbol{x} \in h\}|.$$

Given an undirected graph $G([n], E)$ and $k \in \mathbb{N}$, we will now construct a set $\mathcal{H}_{G,k}$ of $nk + 2|E|\binom{k}{2}$ hyperplanes in $\mathbb{R}^k$ such that G has a clique of size k if and only if there is a point $\boldsymbol{x} \in \mathbb{R}^k$ with $\operatorname{depth}(\boldsymbol{x}, \mathcal{H}_{G,k}) = k + \binom{k}{2}$.

For $1 \le i \le k$ and $1 \le v \le n$ we define the hyperplane $h_i^v = \{\boldsymbol{x} \mid x_i = v\}$. The *scaffolding set* $\mathcal{H}^0 = \{\, h_i^v \mid 1 \le i \le k,\ 1 \le v \le n \,\}$ consists of nk hyperplanes. Any point $\boldsymbol{x}$ is contained in at most k hyperplanes in $\mathcal{H}^0$; equality is realized for the points in $[n]^k$:

Lemma 1. $\operatorname{depth}(\boldsymbol{x}, \mathcal{H}^0) \le k$ *for any* $\boldsymbol{x} \in \mathbb{R}^k$*, and* $\operatorname{depth}(\boldsymbol{x}, \mathcal{H}^0) = k$ *if and only if* $\boldsymbol{x} \in [n]^k$*.*

For $1 \le i < j \le k$ and $1 \le u, v \le n$ we define the hyperplane $h_{ij}^{uv} = \{\, \boldsymbol{x} \mid (x_i - u) + n(x_j - v) = 0 \,\}$. This hyperplane contains only those points $\boldsymbol{x}$ of the grid for which $x_i = u$ and $x_j = v$ (see Figure 1):

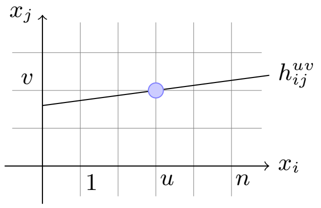


Fig. 1. The hyperplane h_{ij}^{uv} contains only points $\boldsymbol{x} \in [n]^k$ for which $x_i = u$ and $x_j = v$

Lemma 2. *$\boldsymbol{x} \in h_{ij}^{uv} \cap [n]^k$ if and only if $x_i = u$ and $x_j = v$.*

Proof. Assume $\boldsymbol{x} \in h_{ij}^{uv} \cap [n]^k$, i. e., $(x_i - u) + n(x_j - v) = 0$ and $x_i, x_j \in [n]$. If $x_i \neq u$, the left-hand side of the equation is not divisible by n and thus cannot be 0. Therefore, $x_i = u$ and thus, $x_j = v$. The other direction is obvious.

For $1 \leq i < j \leq k$ we define the set $\mathcal{H}_{ij}^E = \{ h_{ij}^{uv} \mid uv \in E \text{ or } vu \in E \}$ of $2|E|$ hyperplanes. All these hyperplanes are parallel; thus a point is contained in at most one hyperplane of $\mathcal{H}_{ij}^E$. By Lemma 2, a point $\boldsymbol{x} \in [n]^k$ is contained in a hyperplane of $\mathcal{H}_{ij}^E$ if and only if $x_i x_j$ is an edge of E.

We define the *constraint set* $\mathcal{H}^E = \bigcup_{1 \leq i < j \leq k} \mathcal{H}_{ij}^E$ consisting of $2|E|\binom{k}{2}$ hyperplanes. From the above, we have the following facts:

Lemma 3. (a) $\operatorname{depth}(\boldsymbol{x}, \mathcal{H}^E) \leq \binom{k}{2}$ *for any* $\boldsymbol{x} \in \mathbb{R}^k$.
(b) *Let* $\boldsymbol{x} \in [n]^k$. *Then* $\operatorname{depth}(\boldsymbol{x}, \mathcal{H}^E) = |\{ (i,j) \mid 1 \leq i < j \leq k,\ x_i x_j \in E \}|$
(c) *Let* $\boldsymbol{x} \in [n]^k$. *Then* $\operatorname{depth}(\boldsymbol{x}, \mathcal{H}^E) = \binom{k}{2}$ *iff* $\{x_1, \ldots, x_k\}$ *is a k-clique in G.*

For the set $\mathcal{H}_{G,k} = \mathcal{H}^0 \cup \mathcal{H}^E$, Lemmas 1 and 3 immediately imply:

Lemma 4. $\operatorname{depth}(\boldsymbol{x}, \mathcal{H}_{G,k}) = k + \binom{k}{2}$ *if and only if* $\boldsymbol{x} \in [n]^k$ *and* $\{x_1, \ldots, x_k\}$ *is a k-clique in G.*

The above construction of the set $\mathcal{H}_{G,k}$ is an fpt-reduction with respect to both the depth of the set of hyperplanes, i. e., the maximum number of hyperplanes covering any point, and the dimension. Hence, we have the following.

Theorem 1. *Given a set of n of linear equations on d variables and an integer l, deciding whether there exists a solution that satisfies l of the equations is W[1]-hard with respect to both l and d.*

Replacing each equation by 2 inequalities, an instance of the above problem is transformed into an instance with linear inequalities such that there exists a solution satisfying l out of the n equations of the original instance if and only if there exists a solution satisfying $n + l$ out of the $2n$ inequalities of the final instance; the number of variables stays the same. Hence, we have the following:

Theorem 2. *Given a set of n linear inequalities on d variables and an integer l, deciding whether there exists a solution that satisfies l of the inequalities is W[1]-hard with respect to d.*

In this reduction the dimension is linear in the size of the clique, hence an $n^{o(d)}$-time algorithm for any of these problems implies an $n^{o(k)}$-time algorithm for the k-clique problem, which in turn implies that n-variable 3-CNFSAT can be solved in $2^{o(n)}$-time. The Exponential Time Hypothesis conjectures that no such algorithm exists.

2.3 Further Applications

The scaffold-constraint technique has been applied succesfully to a number of geometric problems in $\mathbb{R}^d$ from various application areas. Most of them were known to be NP-hard and all of them are trivially solvable in time $n^{O(d)}$, see for example [28,29,23,16,4]:

- Shape matching problems [6]:
  - Given two point sets $A, B \in \mathbb{R}^d$, decide whether A is congruent to a subset of B. This result generalizes to any point-set distance D for which $D(A,B) = 0 \Leftrightarrow A \subset B$; this is a desired property for any distance that is used to find small patterns in larger ones, e.g., the directed Hausdorff distance or the Earth Mover's distance.
- Clustering and related problems [7,21]:
  - Given points in $\mathbb{R}^d$, decide whether they can be covered by the union of 2 unit balls/4 unit cubes.
  - Given points in $\mathbb{R}^d$, compute their minimum enclosing cylinder.
  - Given two point sets in $\mathbb{R}^d$, decide whether they can be separated by two hyperplanes.
- Discrepancy-computation and related problems [22]:
  - Given two point sets R, B in $\mathbb{R}^d$, compute an axis-aligned box that does not contain any point of R and that contains as many points of B as possible.
  - Given a point set P in $[0,1]^d$, compute
    - the largest empty axis-aligned box inside $[0,1]^d$ that contains the origin.
    - the largest empty axis-aligned box inside $[0,1]^d$.
    - the star/box discrepancy of P.

Using reductions from k-clique following the framework described above we can show that (the decision versions of) all these problems are W[1]-hard when parameterized by the dimension d.

3 The k-Sum Technique

In this section we describe a different approach to obtain W[1]-hardness results for geometric problems parameterized by the dimension by giving fpt-reductions from the k-Sum problem [17]:

- Given n pairwise distinct integers, decide whether k of them sum to zero.

This problem is NP-hard [17] and can be solved in (roughly) $O(n^{k/2})$ time. It can be shown to be W[1]-hard with respect to k from a simple reduction from the subset-sum problem which was shown to be W[1]-hard by Downey and Koblitz [18]. The reduction of [18] transforms an instance of the k-clique problem on a graph with n vertices into a $O(k^2)$-Sum instance on integers with $O(n)$ many bits.

Reductions from k-Sum seem somewhat more "natural" for computational geometers: Gajentaan and Overmars [20] introduced the 3-Sum problem for the purpose of arguing that certain problems in planar geometry "should" take $\Omega(n^2)$ time; showing 3-Sum-hardness for such problems is considered a routine task today. Surprisingly, to the best of our knowledge, this approach has not been used to show W[1]-hardness of geometric problems in $\mathbb{R}^d$.

Based on the work of Erickson [17] it is rather straightforward to show W[1]-hardness for two problems by a reduction from k-Sum. We first describe a reduction for the *affine degeneracy-detection problem*:

– Given an n-point set P in $\mathbb{R}^d$, decide it is affinely degenerate, i.e., if any $d+1$ points of P lie on a single hyperplane.

As mentioned above, this problem is the dual of the maximum-size feasible subsystem problem for linear *equations* (for $l = d+1$), so we already know that it is W[1]-hard. The following argument seems conceptually much simpler than the one from the previous section: The *weird moment curve* in $\mathbb{R}^d$, denoted $\omega_d(t)$, is the set of points $\omega_d(t) = (t, t^2, \ldots, t^{d-1}, t^{d+1})$, where the parameter t ranges over the reals. Erickson [17] has shown that for real numbers $x_0 < x_1 < \cdots < x_d$ the orientation of the simplex $(\omega_d(x_0), \omega_d(x_1), \ldots, \omega_d(x_d))$ is given by the sign of $\sum_{i=0}^{d} x_i$; in particular the simplex is affinely degenerate iff $\sum_{i=0}^{d} x_i = 0$. So by lifting a k-Sum instance to the weird moment curve in $\mathbb{R}^k$ we immediately get an fpt-reduction and thus the following:

Theorem 3. *Given an n-point set P in $\mathbb{R}^d$, deciding if it is affinely degenerate is W[1]-hard with respect to d.*

As was already noted this approach has a drawback: The reduction of [18] transforms an instance of the k-clique problem into a $O(k^2)$-Sum instance. If we "append" the lifting reduction just described we can merely conclude that the affine degeneracy-detection problem in $\mathbb{R}^d$ cannot be solved in time $n^{o(\sqrt{d})}$ (assuming the ETH).

A very similar reduction works for the *convex hull simlicity-detection problem*:

– Given an n-point set in $\mathbb{R}^d$, decide whether its convex hull is *simplicial*, i.e., if all its facets (and thus all its faces) are simplices.

In unbouded dimension this problem is known to be coNP-complete [10,15] and counting the number of convex hull facets is #P-hard [15]. An n-vertex polytope in $\mathbb{R}^d$ can have $\Omega(n^{\lfloor d/2 \rfloor})$ facets and the complete hull can be computed in $O(n^{\lfloor d/2 \rfloor})$ time [11]; output sensitive algorithm are known that run in time $O(n^{1-1/(\lfloor d/2 \rfloor+1)} \operatorname{polylog} n + f \log n)$ (where f denotes the size of the output), c.f. [17] and the references therein.

Using the weird moment curve, Erickson [17] describes a reduction from the k-Sum problem to the problem of detecting simplicial convex hulls in $\mathbb{R}^{2k-1}$: A set X of n integers is transformed into a set X' of $2n$ points in $\mathbb{R}^{2k-1}$ such that k elements of X sum to zero iff the convex hull of X' is simplicial. In the reduction the coordinates of X' increase by a multiplicative factor of 2^{dn} (so their bit-length increases only *polynomially*). This gives the following:

Theorem 4. *Given an n-point set P in $\mathbb{R}^d$, deciding if its convex hull is simplicial is W[1]-hard with respect to d.*

Of course this approach has the same drawback as the reduction for the affine degeneracy-detection problem.

References

1. Armaldi, E., Kann, V.: The complexity and approximability of finding maximum feasible subsystems of linear relations. Theoretical Computer Science 147, 181–210 (1995)
2. Aronov, B., Har-Peled, S.: On approximating the depth and related problems. SIAM J. Comput. 38(3), 899–921 (2008)
3. Arora, S., Babai, L., Stern, J., Sweedyk, Z.: The hardness of approximate optima in lattices, codes, and systems of linear equations. J. Comput. Syst. Sci. 54(2), 317–331 (1997)
4. Backer, J., Keil, J.: The mono- and bichromatic empty rectangle and square problems in all dimensions. In: Proc. of the 9th Latin American Theoretical Informatics Symposium, Oaxaca, Mexico. LNCS. Springer, Heidelberg (to appear, 2010)
5. Bremner, D., Chen, D., Iacono, J., Langerman, S., Morin, P.: Output-sensitive algorithms for Tukey depth and related problems. Statistics and Computing 18(3), 259–266 (2008)
6. Cabello, S., Giannopoulos, P., Knauer, C.: On the parameterized complexity of d-dimensional point set pattern matching. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 175–183. Springer, Heidelberg (2006)
7. Cabello, S., Giannopoulos, P., Knauer, C., Marx, D., Rote, G.: Geometric clustering: fixed-parameter tractability and lower bounds with respect to the dimension. ACM Transactions on Algorithms (2009) (to appear)
8. Cabello, S., Giannopoulos, P., Knauer, C., Rote, G.: Geometric clustering: fixed-parameter tractability and lower bounds with respect to the dimension. In: Proc. 19th Ann. ACM-SIAM Sympos. Discrete Algorithms, pp. 836–843 (2008)
9. Chan, T.M.: A (slightly) faster algorithm for Klee's measure problem. In: Proc. 24th Annual Symposium on Computational Geometry, pp. 94–100 (2008)
10. Chandrasekaran, R., Kabadi, S.N., Murthy, K.G.: Some NP-complete problems in linear programming. Operations Research Letters 1(3), 101–104 (1982)
11. Chazelle, B.: An optimal convex hull algorithm in any fixed dimension. Discrete & Computational Geometry 10, 377–409 (1993)
12. Chen, J., Chor, B., Fellows, M., Huang, X., Juedes, D., Kanj, I.A., Xia, G.: Tight lower bounds for certain parameterized NP-hard problems. Information and Computation 201(2), 216–231 (2005)
13. Chen, J., Huang, X., Kanj, I.A., Xia, G.: Strong computational lower bounds via parameterized complexity. J. Comput. Syst. Sci. 72(8), 1346–1367 (2006)

14. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science. Springer, Heidelberg (November 1999)
15. Dyer, M.E.: The Complexity of Vertex Enumeration Methods. Mathematics of Operations Research 8(3), 381–402 (1983)
16. Eckstein, J., Hammer, P.L., Liu, Y., Nediak, M., Simeone, B.: The maximum box problem and its application to data analysis. Comput. Optim. Appl. 23(3), 285–298 (2002)
17. Erickson, J.: New lower bounds for convex hull problems in odd dimensions. SIAM J. Comput. 28(4), 1198–1214 (1999)
18. Fellows, M.R., Koblitz, N.: Fixed-parameter complexity and cryptography. In: Moreno, O., Cohen, G., Mora, T. (eds.) AAECC 1993. LNCS, vol. 673, pp. 121–131. Springer, Heidelberg (1993)
19. Flum, J., Grohe, M.: Parameterized Complexity Theory. Texts in Theoretical Computer Science. Springer, Heidelberg (2006)
20. Gajentaan, A., Overmars, M.H.: On a class of $O(n^2)$ problems in computational geometry. Comput. Geom. Theory Appl. 5(3), 165–185 (1995)
21. Giannopoulos, P., Knauer, C., Rote, G.: The parameterized complexity of some geometric problems in unbounded dimension. In: Chen, J., Fomin, F.V. (eds.) IWPEC 2009. LNCS, vol. 5917, pp. 198–209. Springer, Heidelberg (2009)
22. Giannopoulos, P., Knauer, C., Wahlström, M., Werner, D.: Hardness of discrepancy and related problems parameterized by the dimension. In: 26th European Workshop on Computational Geometry, EuroCG (to appear, 2010)
23. Gnewuch, M., Srivastav, A., Winzen, C.: Finding optimal volume subintervals with k points and calculating the star discrepancy are NP-hard problems. J. Complexity 25, 115–127 (2009)
24. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. J. Comput. Syst. Sci. 62(2), 367–375 (2001)
25. Langerman, S., Morin, P.: Covering things with things. Discrete & Computational Geometry 33(4), 717–729 (2005)
26. Matousek, J.: On geometric optimization with few violated constraints. Discrete & Computational Geometry 14(4), 365–384 (1995)
27. Megiddo, N.: Linear programming in linear time when the dimension is fixed. J. ACM 31, 114–127 (1984)
28. Megiddo, N.: On the complexity of polyhedral separability. Discrete & Computational Geometry 3, 325–337 (1988)
29. Megiddo, N.: On the complexity of some geometric problems in unbounded dimension. J. Symb. Comput. 10, 327–334 (1990)
30. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, Oxford (2006)

Different Approaches to Proof Systems

Olaf Beyersdorff[1,⋆] and Sebastian Müller[2,⋆⋆]

[1] Institute of Computer Science, Humboldt University Berlin, Germany
[2] Faculty of Mathematics and Physics, Charles University Prague, Czech Republic
{beyersdo,smueller}@informatik.hu-berlin.de

Abstract. The classical approach to proof complexity perceives proof systems as deterministic, uniform, surjective, polynomial-time computable functions that map strings to (propositional) tautologies. This approach has been intensively studied since the late 70's and a lot of progress has been made. During the last years research was started investigating alternative notions of proof systems. There are interesting results stemming from dropping the uniformity requirement, allowing oracle access, using quantum computations, or employing probabilism. These lead to different notions of proof systems for which we survey recent results in this paper.

1 Introduction

In their seminal paper [CR79], Cook and Reckhow defined the notion of a *proof system* for an arbitrary language L as a polynomial-time computable function f with range L. A string w with $f(w) = x$ is called an f*-proof* for $x \in L$. All classical proof systems like Resolution, Cutting Planes, or Frege systems fall under this general concept, and in the last thirty years there has been great progress in understanding the complexity of proofs in this model (cf. [Seg07] for a recent survey).

While the Cook-Reckhow approach is certainly the most useful setting for practical applications, it is nevertheless interesting to ask what happens if we allow alternative computational resources for the verification of proofs. This approach is very common in complexity theory where besides (non-)deterministic polynomial time a number of other models like randomisation, non-uniformity, oracle access, or new paradigms as quantum computing are studied.

In proof complexity these considerations were started recently by several researchers. In this paper we mainly survey results on proof systems with advice which were introduced by Cook and Krajíček [CK07], but also mention randomised systems investigated by Hirsch and Itsykson [HI10,Hir10] and quantum proof systems introduced by Pudlák [Pud09]. The common idea in these approaches is that verification of proofs can be performed with additional resources,

⋆ This paper was written while the first author was visiting Sapienza University of Rome under support of DFG grant KO 1053/5-2.

⋆⋆ Supported by the Marie Curie FP7 Initial Training Network MALOA (no. 238381).

© Springer-Verlag Berlin Heidelberg 2010

not just polynomial time. The results show a number of new phenomena such as the existence of optimal proof systems with advice or under weak oracles. Such results are not known in the classical setting. We also address other interesting questions such as the existence of polynomially bounded proof systems—which receives a different characterization in the advice model—and whether proofs can be shortened by using quantum rules.

2 Proof Systems Using Advice

Our first non-classical model will be proof systems that use advice. Like in the classical setting of Karp and Lipton [KL80] this will allow the proof systems to use a specified amount of non-uniform information. Proof systems with advice were recently introduced by Cook and Krajíček [CK07] and further developed by Beyersdorff, Köbler, and Müller [BKM09,BM,BM09].

2.1 Setting the Stage

Our general model of computation for proof systems f with advice is a polynomial-time Turing transducer with several tapes: an input tape containing the proof π, possibly several work tapes for the computation of the machine, an output tape where we output the proven element $f(\pi)$, and an advice tape containing the advice. We start with a quite flexible definition of proof systems with advice for arbitrary languages, generalizing the notion of propositional proof systems with advice from [CK07].

Definition 1 ([BKM09]). *For a function $k : \mathbb{N} \to \mathbb{N}$, a proof system f for L is a* proof system with k bits of advice*, if there exist a polynomial-time Turing transducer M, an advice function $h : \mathbb{N} \to \Sigma^*$, and an advice selector function $\ell : \Sigma^* \to 1^*$ such that*

1. *ℓ is computable in polynomial time,*
2. *M computes the proof system f with the help of the advice h, i.e., for all $\pi \in \Sigma^*$, $f(\pi) = M(\pi, h(|\ell(\pi)|))$, and*
3. *for all $n \in \mathbb{N}$, the length of the advice $h(n)$ is bounded by $k(n)$.*

For a class F of functions, we denote by ps/F the class of all ps/k with $k \in F$.

We say that f *uses k bits of input advice* if ℓ has the special form $\ell(\pi) = 1^{|\pi|}$. On the other hand, in case $\ell(\pi) = 1^{|f(\pi)|}$ for all π in the domain of f, then f is said to *use k bits of output advice*. By this definition, proof systems with input advice use non-uniform information depending on the length of the proof, while proof systems with output advice use non-uniform information depending on the length of the proven formula.

We note that proof systems with advice are a quite powerful concept, as for every language $L \subseteq \Sigma^*$ there exists a proof system for L with only one bit of advice. In contrast, the class of all languages for which proof systems without advice exist coincides with the class of all recursively enumerable languages.

2.2 Polynomially Bounded Proof Systems with Advice

The classical Cook-Reckhow Theorem states that $\mathsf{NP} = \mathsf{coNP}$ if and only if the set of all tautologies TAUT has a polynomially bounded proof system, i.e., there exists a polynomial p such that every tautology φ has a proof of size $\leq p(|\varphi|)$ in the system. Consequently, showing super-polynomial lower bounds to the proof size in propositional proof systems of increasing strength provides one way to attack the P/NP problem. This approach, also known as the Cook-Reckhow program, has lead to a very fruitful research on the length of propositional proofs.

What happens if the proof systems may use advice? Which languages admit polynomially bounded proof systems in this new model? In [BKM09] a complete characterization of this question was given. In particular, there is a tight connection of this problem to the notion of nondeterministic instance complexity. Similarly as Kolmogorov complexity, instance complexity measures the complexity of individual instances of a language [OKSW94]. We now give the definition of nondeterministic instance complexity from [AKMT00].

Definition 2 (Arvind et al. [AKMT00]). *For a set L and a time bound t, the t-time-bounded* nondeterministic instance complexity *of x with respect to L is defined as $nic^t(x : L) = \min\{\, |M| \,:\, M$ is a t-time-bounded nondeterministic machine, $L(M) \subseteq L$, and M decides correctly on $x\,\}$.*

We collect all languages with prescribed upper bounds on the running time and nondeterministic instance complexity in a complexity class.

Definition 3 ([BKM09]). *The complexity class $\mathsf{NIC}[\mathsf{log}, \mathsf{poly}]$ contains all languages L for which there exists a polynomial p such that $nic^p(x : L) \leq O(\log |x|)$ holds for all $x \in \Sigma^*$.*

This class can be strictly placed between familiar non-uniform complexity classes:

Theorem 4 ([BKM09]). $\mathsf{NP} \subsetneq \mathsf{NP}/1 \subsetneq \mathsf{NP}/\mathsf{log} \subsetneq \mathsf{NIC}[\mathsf{log}, \mathsf{poly}] \subsetneq \mathsf{NP}/\mathsf{poly}$.

The classes in Theorem 4 are exactly the classes which in appear in the characterization of polynomially bounded proof systems with advice, as given in Table 1. Quite unusually in complexity theory, all complexity classes appearing in this table are distinct by Theorem 4.

Table 1. Languages with polynomially bounded proof systems

	input advice	output advice	reference
ps/poly	$\mathsf{NP}/\mathsf{poly}$	$\mathsf{NP}/\mathsf{poly}$	[BKM09]
ps/log	$\mathsf{NIC}[\mathsf{log}, \mathsf{poly}]$	NP/log	[BKM09]
$ps/1$	$\mathsf{NIC}[\mathsf{log}, \mathsf{poly}]$	$\mathsf{NP}/1$	[BKM09]
$ps/0$	NP		[CR79]

Concentrating on propositional proof systems (or more generally, on languages from coNP), the picture simplifies a bit because it was shown in [BKM09] that for a language $L \in \mathsf{coNP}$, $L \in \mathsf{NP}/\mathsf{log}$ if and only if $L \in \mathsf{NIC}[\mathsf{log}, \mathsf{poly}]$.

It is also natural to ask, how likely these assumptions actually are, i.e., what consequences follow from the assumption that such proof systems exist. For TAUT we obtain a series of collapse consequences of presumably different strength as shown in Table 2.

Table 2. Consequences of the existence of polynomially bounded proof systems (results are from [BKM09])

Assumption *if* TAUT *has a polynomially bounded ...*		**Consequence** *then* PH *collapses to ...*
ps/poly	(input or output advice)	$\mathsf{S}_2^{\mathsf{NP}} \subseteq \Sigma_3^{\mathsf{p}}$
ps/log	(input or output advice)	$\mathsf{P}^{\mathsf{NP}[\mathsf{log}]}$
$ps/O(1)$	(input advice)	$\mathsf{P}^{\mathsf{NP}[\mathsf{log}]}$
$ps/O(1)$	(output advice)	$\mathsf{P}^{\mathsf{NP}[O(1)]} = \mathsf{BH}$
$ps/0$	(no advice)	NP

2.3 Optimal Proof Systems with Advice

Proof systems are compared according to their strength by simulations as introduced in [CR79] and [KP89]. If f and g are proof systems for L, we say that g *simulates* f if there exists a polynomial p such that for all $x \in L$ and f-proofs w of x there is a g-proof w' of x with $|w'| \leq p(|w|)$. If such a proof w' can even be computed from w in polynomial time, we say that g *p-simulates* f. Proof systems f, g which mutually (p-)simulate each other are called *(p-)equivalent.*

A prominent open question posed in [KP89] is whether there exists a strongest proof system, called a *(p-)optimal* proof system, which (p-)simulates all proof systems for L. This question has interesting consequences such as existence of complete languages for promise classes [KMT03, BS09]. Despite a considerable research effort the existence of optimal proof systems is still open (cf. [Hir10] in this volume). Surprisingly, Cook and Krajíček [CK07] have shown that there exists a propositional proof system with one bit of input advice which simulates all classical Cook-Reckhow proof systems. The proof of this result easily generalizes to arbitrary languages L, thus yielding:

Theorem 5 (Cook, Krajíček [CK07], [BKM09]). *For every language L there exists a proof system P with one bit of input advice such that P simulates all ps/log for L. Moreover, P p-simulates all advice-free proof systems for L.*

In contrast, it seems unlikely that we can obtain a similar result for output advice by current techniques (cf. [BM08] were we investigated this problem for propositional proof systems). The question whether this optimality result can be strengthened to p-optimality (where the simulations are replaced by p-simulations) was also studied in detail in [BM08], with both negative and positive results providing partial answers to the question.

We remark that optimal proof systems are known to imply complete sets for various promise classes [KMT03], and this relation also holds in the presence of advice [BS09]. A related line of research has shown strong time and space hierarchy theorems for randomised and other semantic classes which use advice [FS04, FST05, vMP07, KvM08]. All these results are not known to hold in the classical advice-free setting.

2.4 Proof Systems with Advice and Bounded Arithmetic

Propositional proof systems enjoy a very close relationship to weak arithmetic theories, so-called bounded arithmetic, which in particular yields insight into strong proof systems as Frege systems and their extensions [Kra95]. This connection also holds in the presence of advice, and this, in fact, was the motivation for their introduction in [CK07]. There, Cook and Krajíček investigate Karp-Lipton collapse consequences of the assumption $\mathsf{NP} \subseteq \mathsf{P/poly}$. The classical Karp-Lipton Theorem states that $\mathsf{NP} \subseteq \mathsf{P/poly}$ implies a collapse of the polynomial hierarchy PH to its second level [KL80]. Subsequently, these collapse consequences have been improved by Köbler and Watanabe [KW98] to $\mathsf{ZPP}^{\mathsf{NP}}$ and by Sengupta and Cai to $\mathsf{S}_2^{\mathsf{p}}$ (cf. [Cai07]). Making the stronger assumption that $\mathsf{NP} \subseteq \mathsf{P/poly}$ is provable in some weak arithmetic theory, Cook and Krajíček obtained stronger collapse consequences, namely to the Boolean hierarchy if the theory is PV (cf. also [BM, Jeř09]).

One important intermediate step towards this result is a surprising trade-off between advice and nondeterminism (which is unlikely to hold without reference to bounded arithmetic):

Theorem 6 (Cook, Krajíček [CK07]). *PV proves $\mathsf{NP} \subseteq \mathsf{P/poly}$ if and only if PV proves $\mathsf{coNP} \subseteq \mathsf{NP}/O(1)$.*

The latter condition can be interpreted as saying that there exists a polynomially bounded proof system using constant advice (and, moreover, the polynomial boundedness in provable in PV). In fact, Cook and Krajíček even exhibit a natural proof system P with advice that is polynomially bounded if PV proves $\mathsf{NP} \subseteq \mathsf{P/poly}$: the system P is an extended Frege system with constant advice.

2.5 Simplifying the Advice

From a practical point of view, proof systems with advice are susceptive to criticism: advice can be arbitrarily complex (even non-recursive) and thus verifying proofs with the help of advice does not form a feasible model to use in practice. The next result shows that for propositional proof systems, logarithmic advice can be replaced by a sparse NP-oracle without increasing the proof length.

Theorem 7 ([BM09])

1. *Every propositional proof system with logarithmic advice is simulated by a propositional proof system computable in polynomial time with access to a sparse NP-oracle.*

2. *Conversely, every propositional proof system computable in polynomial time with access to a sparse* NP*-oracle is simulated by a propositional proof system with logarithmic advice.*

We remark that sparse NP-sets indeed seem to be very weak if used as oracles. For instance, TAUT $\notin \mathsf{NP}^S$ with a sparse NP-oracle S, unless the polynomial hierarchy collapses to its second level [Kad89].

Another simplification of advice was investigated in [BM09]. As we have seen, there are two natural ways to enhance proof systems with advice by either supplying non-uniform information to the proof (input advice) or to the proven formula (output advice). Intuitively, input advice is the stronger model: proofs can be quite long and formulas of the same size typically require proofs of different size. Hence, supplying advice depending on the proof size is not only more flexible, but also results in more advice per formula.

Therefore, shifting the advice from the proof to the formula will result in a simplification of advice. In this direction in was shown in [BM09] that if there exists a proof system with advice with nontrivial upper bounds on the proof lengths, then there is such a proof system with output advice.

3 Probabilistic Proof Systems

We will now turn to the use of probabilism to compute proof systems. Usually, the term "probabilistic proofs" is associated with interactive proof systems like IP or Babai's Arthur-Merlin classes MA and AM. Besides from randomisation, the power of these proof systems stems from using interaction between a powerful prover and a polynomial-time verifier.

A non-interactive model of randomized proofs was very recently introduced by Hirsch and Itsykson [HI10]. They define two concepts: heuristic acceptors and heuristic proof systems. Acceptors are not really proof systems, but algorithms which accept all elements from the language and do not stop on other inputs. There is, however, a close relationship between acceptors and proof systems (cf. [KP89]). As there is a nice survey on optimal acceptors and optimal proof systems in this volume [Hir10], we will be very brief on this randomized model.

For the randomized approach, we have to consider a probability distribution. A distribution D is concentrated on some set A, if $\mu_D(A) = 1$.

Definition 8 (Hirsch, Itsykson [HI10]). *A pair (D, L) is a* distributional proving problem *if D is a family of probability distributions D_n concentrated on $\overline{L} \cap \{0,1\}^n$.*

Hirsch and Itsykson define a *heuristic acceptor* for a distributional proving problem (D, L) as a randomized algorithm which always accepts inputs from L and accepts inputs from $\bar{L}$ only with small probability (see [Hir10] for the exact definition). For this model they show an optimality result:

Theorem 9 (Hirsch, Itsykson [HI10]). *Let L be recursively enumerable and D be a polynomial-time samplable distribution. Then there is an optimal automatizer for (D, L).*

The authors also consider heuristic proof systems and show interesting results on these systems with respect to automatizability, i. e., the problem to construct proofs for given formulas (see [Hir10]).

4 Quantum Proof Systems

As our last model we briefly mention quantum proof systems as introduced by Pudlák [Pud09]. Since Shor's polynomial-time quantum algorithm for factoring [Sho97], quantum computations are a computational model which has attracted an enormous amount of research. Recently, Pudlák investigated the usage of quantum rules in propositional proof systems [Pud09].

Pudlák first introduces a general model of quantum proof systems and then focuses on quantum Frege systems. Let us start with the general concept.

Definition 10 (Pudlák [Pud09]). *A* quantum proof system *consists of a set $A \subseteq \Sigma^*$ (the set of valid proofs) and a family of circuits C_n (the proof system) such that*

1. *A is decidable in polynomial time and C_n is* P*-uniform* (Efficiency)*;*
2. *for any proof $\pi \in A$, $C_{|\pi|}(\pi)$ produces a superposition of strings of tautologies* (Correctness)*;*
3. *for every tautology φ there exists $\pi \in A$ such that φ occurs in the superposition of $C_{|\pi|}(\pi)$* (Completeness)*.*

Regarding the completeness condition, it is also important that by measuring $C_{|\pi|}(\pi)$ we can obtain φ with a probability which is not too small. Hence quantum proof systems also have probabilistic aspects.

The next concept which Pudlák introduces are *quantum rules* which are based on unitary transformations. Using a finite set of quantum rules, Pudlák arrives at the notion of *quantum Frege systems.* Comparing quantum Frege with classical Frege systems, Pudlák obtains the surprising result that quantum Frege systems do not have shorter proofs, i. e., every quantum Frege system is simulated by a classical Frege system. On the other hand, it does not seem possible to extract classical proofs from quantum Frege proofs, i. e., under cryptographic assumptions quantum Frege systems are not p-simulated by classical Frege systems.

5 Conclusion

We conclude by mentioning that there are more interesting approaches which we did not cover in this survey. For instance, *space complexity* for proof systems was intensively investigated in the context of Resolution [ET01, ABSRW02, BSN08]. Here the minimal space to refute a set of clauses is of particular interest as it corresponds to the memory consumption of modern SAT solvers which often combine DPLL algorithms with clause learning. Therefore, both lower bounds for Resolution space [ABSRW02, BSG03, EGM04, ET03] as well as optimal trade-offs between space and length, i. e., between memory and run-time consumption, have been intensively studied [Nor06, NH08, BSN08, BSN09].

Another approach is to provide a finer analysis of proof lengths in the model of *parameterized proof complexity.* Parameterized resolution and, moreover, a general framework for parameterized proof complexity was recently introduced by Dantchev, Martin, and Szeider [DMS07]. In that paper, Dantchev et al. show a complexity gap in parameterized tree-like resolution for propositional formulas arising from translations of first-order principles. A purely combinatorial approach to obtain lower bounds to the proof size in parameterized tree-like resolution was developed in [BGL10].

Of course, non-classical proof complexity is still a relatively young area of research and many problems are still open. In particular, it is interesting to determine the relationship between the different approaches (e. g. with respect to simulations as in Theorem 7). We believe that further research into non-classical measures of proofs will both strengthen the connections between computational and proof complexity and lead to new insights for classical proof systems.

References

[ABSRW02] Alekhnovich, M., Ben-Sasson, E., Razborov, A.A., Wigderson, A.: Space complexity in propositional calculus. SIAM Journal on Computing 31(4), 1184–1211 (2002)

[AKMT00] Arvind, V., Köbler, J., Mundhenk, M., Torán, J.: Nondeterministic instance complexity and hard-to-prove tautologies. In: Reichel, H., Tison, S. (eds.) STACS 2000. LNCS, vol. 1770, pp. 314–323. Springer, Heidelberg (2000)

[BGL10] Beyersdorff, O., Galesi, N., Lauria, M.: The strength of parameterized tree-like resolution (preprint, 2010)

[BKM09] Beyersdorff, O., Köbler, J., Müller, S.: Nondeterministic instance complexity and proof systems with advice. In: Dediu, A.H., Ionescu, A.M., Martín-Vide, C. (eds.) LATA 2009. LNCS, vol. 5457, pp. 164–175. Springer, Heidelberg (2009)

[BM] Beyersdorff, O., Müller, S.: A tight Karp-Lipton collapse result in bounded arithmetic. ACM Transactions on Computational Logic (to appear)

[BM08] Beyersdorff, O., Müller, S.: A tight Karp-Lipton collapse result in bounded arithmetic. In: Kaminski, M., Martini, S. (eds.) CSL 2008. LNCS, vol. 5213, pp. 199–214. Springer, Heidelberg (2008)

[BM09] Beyersdorff, O., Müller, S.: Does advice help to prove propositional tautologies? In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 65–72. Springer, Heidelberg (2009)

[BS09] Beyersdorff, O., Sadowski, Z.: Characterizing the existence of optimal proof systems and complete sets for promise classes. In: Frid, A., Morozov, A., Rybalchenko, A., Wagner, K.W. (eds.) CSR 2009. LNCS, vol. 5675, pp. 47–58. Springer, Heidelberg (2009)

[BSG03] Ben-Sasson, E., Galesi, N.: Space complexity of random formulae in resolution. Random Structures and Algorithms 23(1), 92–109 (2003)

[BSN08] Ben-Sasson, E., Nordström, J.: Short proofs be spacious: An optimal separation of space and length in resolution. In: Proc. 49th IEEE Symposium on the Foundations of Computer Science, pp. 709–718 (2008)

[BSN09] Ben-Sasson, E., Nordström, J.: Understanding space in resolution: Optimal lower bounds and exponential trade-offs. Technical Report TR09-034, Electronic Colloquium on Computational Complexity (2009)

[Cai07] Cai, J.-Y.: $S_2^p \subseteq ZPP^{NP}$. Journal of Computer and System Sciences 73(1), 25–35 (2007)

[CK07] Cook, S.A., Krajíček, J.: Consequences of the provability of NP $\subseteq$ P/poly. The Journal of Symbolic Logic 72(4), 1353–1371 (2007)

[CR79] Cook, S.A., Reckhow, R.A.: The relative efficiency of propositional proof systems. The Journal of Symbolic Logic 44(1), 36–50 (1979)

[DMS07] Dantchev, S.S., Martin, B., Szeider, S.: Parameterized proof complexity. In: Proc. 48th IEEE Symposium on the Foundations of Computer Science, pp. 150–160 (2007)

[EGM04] Esteban, J.L., Galesi, N., Messner, J.: On the complexity of resolution with bounded conjunctions. Theoretical Computer Science 321(2-3), 347–370 (2004)

[ET01] Esteban, J.L., Torán, J.: Space bounds for resolution. Information and Computation 171(1), 84–97 (2001)

[ET03] Esteban, J.L., Torán, J.: A combinatorial characterization of treelike resolution space. Information Processing Letters 87(6), 295–300 (2003)

[FS04] Fortnow, L., Santhanam, R.: Hierarchy theorems for probabilistic polynomial time. In: Proc. 45th IEEE Symposium on the Foundations of Computer Science, pp. 316–324 (2004)

[FST05] Fortnow, L., Santhanam, R., Trevisan, L.: Hierarchies for semantic classes. In: Proc. 37th ACM Symposium on Theory of Computing, pp. 348–355 (2005)

[GMR89] Goldreich, O., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing 18(2), 186–208 (1989)

[HI10] Hirsch, E.A., Itsykson, D.: On optimal heuristic randomized semidecision procedures, with application to proof complexity. In: Proc. 27th Symposium on Theoretical Aspects of Computer Science, pp. 453–464 (2010)

[Hir10] Hirsch, E.A.: Optimal acceptors and optimal proof systems. In: Proc. 7th Conference on Theory and Applications of Models of Computation. Springer, Heidelberg (2010)

[Jeř09] Jeřábek, E.: Approximate counting by hashing in bounded arithmetic. Journal of Symbolic Logic 74(3), 829–860 (2009)

[Kad89] Kadin, J.: $\mathrm{P}^{\mathrm{NP}[\log n]}$ and sparse Turing-complete sets for NP. Journal of Computer and System Sciences 39, 282–298 (1989)

[KL80] Karp, R.M., Lipton, R.J.: Some connections between nonuniform and uniform complexity classes. In: Proc. 12th ACM Symposium on Theory of Computing, pp. 302–309. ACM Press, New York (1980)

[KMT03] Köbler, J., Messner, J., Torán, J.: Optimal proof systems imply complete sets for promise classes. Information and Computation 184(1), 71–92 (2003)

[KP89] Krajíček, J., Pudlák, P.: Propositional proof systems, the consistency of first order theories and the complexity of computations. The Journal of Symbolic Logic 54(3), 1063–1079 (1989)

[Kra95] Krajíček, J.: Bounded Arithmetic, Propositional Logic, and Complexity Theory. Encyclopedia of Mathematics and Its Applications, vol. 60. Cambridge University Press, Cambridge (1995)

[KvM08] Kinne, J., van Melkebeek, D.: Space hierarchy results for randomized models. In: Proc. 25th Symposium on Theoretical Aspects of Computer Science, pp. 433–444 (2008)

[KW98] Köbler, J., Watanabe, O.: New collapse consequences of NP having small circuits. SIAM Journal on Computing 28(1), 311–324 (1998)

[NH08] Nordström, J., Håstad, J.: Towards an optimal separation of space and length in resolution. In: Proc. 40th ACM Symposium on Theory of Computing, pp. 701–710 (2008)

[Nor06] Nordström, J.: Narrow proofs may be spacious: separating space and width in resolution. In: Proc. 38th ACM Symposium on Theory of Computing, pp. 507–516 (2006)

[OKSW94] Orponen, P., Ko, K., Schöning, U., Watanabe, O.: Instance complexity. Journal of the ACM 41(1), 96–121 (1994)

[Pud09] Pudlák, P.: Quantum deduction rules. Annals of Pure and Applied Logic 157(1), 16–29 (2009)

[Seg07] Segerlind, N.: The complexity of propositional proofs. Bulletin of Symbolic Logic 13(4), 417–481 (2007)

[Sho97] Shor, P.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing 26(5), 1484–1509 (1997)

[vMP07] van Melkebeek, D., Pervyshev, K.: A generic time hierarchy with one bit of advice. Computational Complexity 16(2), 139–179 (2007)

Algebraic Proofs over Noncommutative Formulas

Iddo Tzameret*

Mathematical Institute, Academy of Sciences of the Czech Republic,
Žitná 25, 115 67, Prague 1, Czech Rep.
`http://www.math.cas.cz/~tzameret`

Abstract. We study possible formulations of algebraic propositional proofs operating with noncommutative polynomials written as algebraic noncommutative formulas. First, we observe that a simple formulation of such proof systems gives rise to systems at least as strong as Frege—yielding also a semantic way to define a Cook-Reckhow (i.e., polynomially verifiable) algebraic variant of Frege proofs, different from that given before in [8,11]. We then turn to an apparently weaker system, namely, Polynomial Calculus (PC) where polynomials are written as ordered formulas (*PC over ordered formulas*, for short). This is an algebraic propositional proof system that operates with noncommutative polynomials in which the order of products in all monomials respects a fixed linear ordering on the variables, and where proof-lines are written as noncommutative formulas. We show that the latter proof system is strictly stronger than resolution, polynomial calculus and polynomial calculus with resolution (PCR) and admits polynomial-size refutations for the pigeonhole principle and the Tseitin's formulas. We conclude by proposing an approach for establishing lower bounds on PC over ordered formulas proofs, and related systems, based on properties of lower bounds on noncommutative formulas.

The motivation behind this work is developing techniques incorporating rank arguments (similar to those used in algebraic circuit complexity) for establishing lower bounds on propositional proofs.

1 Introduction

This work attempts to gather certain facts about algebraic proof systems establishing propositional tautologies, in which proof lines are written as algebraic noncommutative formulas (noncommutative formulas, for short). Our main motivation is to develop a method to lower bound the size of certain propositional proofs via a rank argument, similar to that used in algebraic circuit complexity. For this purpose, the choice of noncommutative formulas is natural, since such formulas constitute a rather weak circuit class, and moreover, the proof, given

* Supported by The Eduard Čech Center for Algebra and Geometry and The John Templeton Foundation.

© Springer-Verlag Berlin Heidelberg 2010

by Nisan [14], of exponential-size lower bounds on noncommutative formulas is an especially transparent rank argument.

Research into the complexity of algebraic propositional proofs is a central line in proof complexity (cf. [15,24] for general expositions). Another prominent line of research is that dedicated to connections between circuit classes and the propositional proofs based on these classes. In particular, considerable efforts were made to borrow techniques for lower bounding certain circuit classes, and utilize them to show lower bounds on *proofs* operating with circuits from the given classes. For example, bounded depth Frege proofs can be viewed as propositional logic operating with AC^0 circuits, and lower bounds on bounded depth Frege proofs use techniques borrowed from AC^0 circuits lower bounds (cf. [1,13,16]). Pudlák et al. [17,4] studied proofs based on monotone circuits—motivated by exponential lower bounds on monotone circuits. Raz and the author [22,21,24] investigated algebraic proof systems operating with multilinear formulas—motivated by lower bounds on multilinear formulas for the determinant, permanent and other explicit polynomials [19,18]. Atserias et al. [5], Krajíček [12] and Segerlind [23] have considered proofs operating with ordered binary decision diagrams (OBDDs). The current work is a contribution to this line of research, where the circuit class is noncommutative formulas.

Note: This is an extended abstract. More details, full proofs and full definitions appear in the full version of this paper, available at: `http://www.math.cas.cz/~tzameret/nonComm.pdf` .

1.1 Results and Related Works

We concentrate on algebraic proofs establishing propositional contradictions where polynomials are written as noncommutative formulas. We deal with two kinds of proof systems—both are variants (and extensions) of the polynomial calculus (PC) introduced in [9]. In PC we start from a set of initial polynomials from $\mathbb{F}[x_1, \ldots, x_n]$, the ring of polynomials with coefficients from $\mathbb{F}$ (the intended semantics of a proof-line p is the equation $p = 0$ over $\mathbb{F}$). We derive new proof-lines by using two basic algebraic inference rules: from two polynomials p and q, we can deduce $\alpha \cdot p + \beta \cdot q$, where α, β are elements of $\mathbb{F}$; and from p we can deduce $x_i \cdot p$, for a variable x_i ($i = 1, \ldots, n$). We also have Boolean axioms $x_i^2 - x_i = 0$, for all $i = 1, \ldots, n$, expressing that the variables get the values 0 or 1. A *PC refutation of* Q is a proof of 1 (which is interpreted as $1 = 0$, that is the unsatisfiable equation standing for false) from Q. Our two proof systems extend PC as follows:

1. PC over noncommutative formulas: NFPC. This proof system operates with noncommutative polynomials over a field, written as (arbitrarily chosen)[1] noncommutative formulas. The rules of addition and multiplication are

[1] This means that if a proof-line consists of the polynomial p, then one can choose to write *any* formula that computes p in the proof. In other words, this is a "semantic" proof system.

similar to PC, except that multiplication is done *either from left or right.* We also add a a Boolean axiom $x_ix_j - x_jx_i$ that expresses the fact that for $0, 1$ values to the variables, multiplication is in fact commutative.
2. PC over ordered formulas: OFPC. This proof system is PC operating with noncommutative polynomials in which the order of products in all monomials respects a fixed linear ordering on the variables, and where proof-lines are written as (arbitrarily chosen) noncommutative formulas.

Both proof systems are shown to be Cook-Reckhow systems (that is, *polynomial verifiable*, sound and complete proof systems for propositional tautologies).

(1) The first proof system NFPC is shown to polynomially simulate Frege (this is partly because of the choice of Boolean axioms). This gives a semantic definition of a Cook-Reckhow proof system operating with algebraic formulas, simpler in some way from that proposed by Grigoriev and Hirsch [11]: the paper [11] aims at formulating a formal propositional proof system for establishing propositional tautologies (that is, a Cook-Reckhow proof system), which is an algebraic counterpart of the Frege proof system. In order to make their system polynomially-verifiable, the authors augment it with a set of auxiliary rewriting rules, intended to derive algebraic formulas from previous algebraic formulas via the polynomial-ring axioms (that is, associativity, commutativity, distributivity and the zero and unit elements rules). In this framework algebraic formulas are treated as syntactic terms, and one must explicitly apply the polynomial-ring rewrite rules to derive one formula from another. Our proof system NFPC is simpler in the sense that we achieve (apparently) the same strength as the system in [11], while adding no rewriting rules. The idea is that the only "hard to verify" rewrite rule is the commutativity axiom; and since we show how to efficiently simulate this rule we do not need the other polynomial-ring rewrite rules (like distributivity, associativity, etc.) to make the proof system polynomial verifiable: we can just use the deterministic polynomial identity testing algorithm for noncommutative formulas devised by Raz and Shpilka [20].

(2) For the second system, OFPC, we show that, despite its apparent weakness, it polynomially simulates PCR (and hence PC and resolution), as well as a proof system operating with restricted forms of disjunctions of linear equalities called $\mathrm{R}^0(\mathrm{lin})$ (introduced in [21]). The latter implies polynomial-size refutations for the pigeonhole principle and the Tseitin graph formulas, due to corresponding upper bounds demonstrated in [21].

We then propose a simple lower bound approach for OFPC, based on properties of products of ordered formulas (these properties are proved in a similar manner to Nisan's lower bound on noncommutative formulas, by lower bounding the rank of certain matrices associated with noncommutative polynomials). We show certain sufficient conditions yielding super-polynomial lower bounds on OFPC proofs. We also demonstrate certain obstacles to this approach in the form of a general upper-bound criterion.

2 Preliminaries

2.1 Noncommutative Polynomials and Formulas

Let $\mathbb{F}$ be a field and let $\mathbb{F}\langle x_1,\ldots,x_n\rangle$ denote the *noncommutative* ring of polynomials with coefficients from $\mathbb{F}$ and variables $x_1,\ldots,x_n$. In other words, $\mathbb{F}\langle x_1,\ldots,x_n\rangle$ is the ring of polynomials (where a polynomial is a formal sum of products of variables and field elements) conforming to all the polynomial-ring axioms excluding the commutativity of multiplication axiom. For instance, if x_i, x_j are two different variables, then $x_i \cdot x_j$ and $x_j \cdot x_i$ are two different polynomials in $\mathbb{F}\langle x_1,\ldots,x_n\rangle$ (note that variables do commute with field elements). We say that $\mathcal{A}$ is an *algebra over* $\mathbb{F}$, or an $\mathbb{F}$-*algebra*, if $\mathcal{A}$ is a vector space over $\mathbb{F}$ together with a distributive multiplication operation; where multiplication in $\mathcal{A}$ is associative (but it need not be commutative) and there exists a multiplicative unity in $\mathcal{A}$.

Definition 1 (Noncommutative formula). *Let $\mathbb{F}$ be a field and $x_1, x_2, \ldots$ be variables. A noncommutative algebraic formula is an ordered*[2] *labeled tree, with edges directed from the leaves to the root, and with fan-in at most two. Every leaf of the tree (namely, a node of fan-in zero) is labeled either with an input variable x_i or a field $\mathbb{F}$ element. Every other node of the tree is labeled either with $+$ or $\times$ (in the first case the node is a plus gate and in the second case a product gate). We assume that there is only one node of out-degree zero, called* the root*. An algebraic formula* computes *a noncommutative polynomial in the ring of noncommutative polynomials $\mathbb{F}\langle x_1,\ldots,x_n\rangle$ in the following way. A leaf computes the input variable or field element that labels it. A plus gate computes the sum of polynomials computed by its incoming nodes. A product gate computes the* noncommutative *product of the polynomials computed by its incoming nodes* according to the order of the edges*. The output of the formula is the polynomial computed by the root.*

The **size** of an algebraic formula (and noncommutative formula) f is the total number of nodes in its underlying tree, and is denoted $|f|$. Raz and Shpilka [20] showed that there is a deterministic polynomial identity testing (PIT) algorithm that decides whether two noncommutative formulas compute the same noncommutative polynomial:

Theorem 1 (PIT for noncommutative formulas [20]). *There is a deterministic polynomial-time algorithm that decides whether a given noncommutative formula over a field $\mathbb{F}$ computes the zero polynomial* 0.[3]

Recall the definition of PC in Section 1.1. The *degree* of a PC-proof is the maximal degree of a polynomial in the proof. The **size** of a *PC* proof is the total number of monomials (with nonzero coefficients) in all the proof-lines.

[2] This means that there is an order on the edges coming into a node.

[3] We assume here that the field $\mathbb{F}$ can be efficiently represented (e.g., the field of rationals).

Important note: The size of PC proofs can be defined as the total formula sizes of all proof-lines, where polynomials are written as *sums of monomials*, or more formally, as (unbounded fan-in depth-2) $\Sigma\Pi$ circuits.[4] This complexity measure is equivalent—up to a factor of n—to the usual complexity measure counting the total number of monomials appearing in the proofs.

The proof system PCR is an extension of PC, where we add new variables $\bar{x}_i$ for every original variable x_i and also the axiom $x_i\bar{x}_i - 1$. Thus, $\bar{x}_i$ stands for the negation of x_i (see [2] for the definition of PCR).

2.2 Proof Systems and Simulations

Let $L \subseteq \Sigma^*$ be a language over some alphabet Σ. A *proof system for a language* L is a polynomial-time algorithm A that receives $x \in \Sigma^*$ and a string π over some finite alphabet ("the (proposed) proof" of x), such that there exists a π with $A(x, \pi) = \mathsf{true}$ if and only if $x \in L$. Following [10], a *Cook-Reckhow proof system* is a proof system for the language of propositional tautologies in the De Morgan basis $\{\mathsf{true}, \mathsf{false}, \vee, \wedge, \neg\}$.

Assume that $\mathcal{P}$ is a proof system for the language L, where L is not the set of propositional tautologies in De Morgan's basis. In this case we can still consider $\mathcal{P}$ as a proof system for propositional tautologies by fixing a translation between L and the set of propositional tautologies in De Morgan basis (such that, $x \in L$ iff the translation of x is a propositional tautology). If two proof systems $\mathcal{P}_1$ and $\mathcal{P}_2$ establish two different languages L_1, L_2, respectively, then for the task of comparing their relative strength we fix a translation from one language to the other. In most cases, we shall confine ourselves to proofs establishing propositional tautologies or unsatisfiable CNF formulas.

We can speak of a propositional proof system as a *refutation* system (that is, a system for establishing contradictions and not tautologies), by translating every unsatisfiable propositional formula into its negation (since the latter is a tautology).

Definition 2. *Let $\mathcal{P}_1, \mathcal{P}_2$ be two proof systems for the same language L (in case the proof systems are for two different languages we fix a translation from one language to the other, as described above). We say that $\mathcal{P}_2$* polynomially simulates *$\mathcal{P}_1$ if given a $\mathcal{P}_1$ proof (or refutation) π of a F, then there exists a proof (respectively, refutation) of F in $\mathcal{P}_2$ of size polynomial in the size of π. In case $\mathcal{P}_2$ polynomially simulates $\mathcal{P}_1$ while $\mathcal{P}_1$ does not polynomially simulates $\mathcal{P}_2$ we say that $\mathcal{P}_2$ is* strictly stronger *than $\mathcal{P}_1$.*

3 Polynomial Calculus over Noncommutative Formulas

3.1 Discussion

In this section we propose a possible formulation of algebraic propositional proof systems that operate with noncommutative polynomials. We observe that

[4] That is, circuits whose roots are plus gates having product gates as children (and a bottom level containing only variables).

dealing with *propositional* proofs—that is, proofs whose variables range over $0, 1$ values—makes the variables "semantically" commutative. Therefore, for the proof systems to be complete (for unsatisfiable collections of noncommutative polynomials over $0, 1$ values), one may need to introduce rules or axioms expressing commutativity. We show that such a natural formulation of proofs operating with noncommutative formulas polynomially simulate the entire Frege system.

This justifies—if one is interested in concentrating on propositional proof systems weaker than Frege (and especially on concrete lower bounds questions)—our formulation in Section 4 of algebraic proofs operating with noncommutative algebraic formulas with a *fixed product order* (called *ordered formulas*). The latter system can be viewed as operating with commutative polynomials over a field precisely like PC, while the complexity of proofs is measured by the total sizes of ordered formulas needed to write the polynomials in the proof. In other words, the role played by the noncommutativity is only in measuring the sizes of proofs: while in PC-proofs the size measure is defined as the number of monomials appearing in the proofs—or equivalently, the total size of formulas in proofs in which formulas are written as depth-2 $\Sigma\Pi$ formulas—the proof system developed in Section 4 is measured by the total ordered formula size.

3.2 The Proof System NFPC

We now define a proof system operating with noncommutative polynomials written as noncommutative algebraic formulas.

In algebraic proof systems like the polynomial calculus we transform unsatisfiable propositional formulas into a collection Q of polynomials having no solution over a field $\mathbb{F}$. Accordingly, in the noncommutative setting we translate unsatisfiable propositional formulas into a collection Q of noncommutative polynomials from $\mathbb{F}\langle x_1, \ldots, x_n\rangle$ that have no solution over any noncommutative $\mathbb{F}$-algebra (e.g., the matrix algebra with entries from $\mathbb{F}$). Although our "Boolean" axioms will not force only $0, 1$ solutions over noncommutative $\mathbb{F}$-algebras, they will be sufficient for our purpose: every unsatisfiable propositional formula translates (via a standard polynomial translation) into a collection Q of noncommutative polynomials from $\mathbb{F}\langle x_1, \ldots, x_n\rangle$, for which Q and the Boolean axioms have no (common) solution in any noncommutative $\mathbb{F}$-algebra. Furthermore, *the Boolean axioms will in fact force commutativity of variables product*—as required for variables that range over $0, 1$ values (although, again, the Boolean axioms do not force only $0, 1$ values when variables range over noncommutative $\mathbb{F}$-algebras).

Definition 3 (Polynomial calculus over noncommutative formulas: NFPC). *Fix a field $\mathbb{F}$ and let $Q := \{q_1, \ldots, q_m\}$ be a collection of noncommutative polynomials from $\mathbb{F}\langle x_1, \ldots, x_n\rangle$. Let the set of axiom polynomials, called the* ***Boolean axioms****, be:*

$$x_i \cdot (1 - x_i) \quad \textit{for all } i \in [n], \qquad x_i \cdot x_j - x_j \cdot x_i \quad \textit{for all } i \neq j \in [n]\,.$$

Let $\pi = (p_1, \ldots, p_\ell)$ be a sequence of noncommutative polynomials from $\mathbb{F}\langle x_1, \ldots, x_n\rangle$, such that for each $i \in [\ell]$, either $p_i = q_j$ for some $j \in [m]$, *or p_i is a Boolean axiom, or p_i was deduced by one of the following inference rules using p_j, p_k , for $j, k < i$:*

$$\textbf{\textit{Left/right product:}} \quad \frac{p}{x_r \cdot p}\,, \qquad \frac{p}{p \cdot x_r} \quad \textit{for } r \in [n]\,.$$

$$\textbf{\textit{Addition:}} \quad \frac{p \qquad q}{a \cdot p + b \cdot q} \quad \textit{for } a, b \in \mathbb{F}$$

We say that π is an NFPC proof of p_ℓ from Q *if all proof-lines in π are written as noncommutative formulas. (The semantics of an* NFPC *proof-line $p_i = 0$ is the polynomial equation $p_i = 0$.) An* NFPC refutation of Q *is a proof of the polynomial* 1 *from Q. The* ***size*** *of an* NFPC *proof π is defined as the total sizes of all noncommutative formulas in π and is denoted by $|\pi|$.*

Remark 1. (i) The Boolean axioms might have roots different from $0, 1$ over noncommutative $\mathbb{F}$-algebras. (ii) The Boolean axioms are true for $0, 1$ assignments: $x_i \cdot x_j - x_i \cdot x_j = 0$, for all $x_i, x_j \in \{0, 1\}$.

We now show that NFPC is a sound and complete Cook-Reckhow proof system. First note that we have defined NFPC with no rules expressing the polynomial-ring axioms. Nevertheless, due to the deterministic polynomial-time PIT procedure for noncommutative formulas (Theorem 1) the proof system defined will be a Cook-Reckhow system (that is, verifiable in polynomial-time [whenever the base field and its operations can be efficiently represented]).

Proposition 1. *There is a deterministic polynomial-time algorithm that decides whether a given string is an* NFPC*-proof (over efficiently represented fields).*

For the next statements we use the algebraic propositional proof system $\mathcal{F}$-$\mathcal{PC}$ introduced by Grigoriev and Hirsch [11] as an algebraic counterpart of the Frege system. We refer the reader to [11] for definitions.

Proposition 2. *The systems* NFPC *is sound and complete. Specifically, let Q be a collection of noncommutative polynomials from $\mathbb{F}\langle x_1, \ldots, x_n\rangle$. Assume that for every $\mathbb{F}$-algebra, there is no $0, 1$ solution for Q (that is, an $0, 1$ assignment to variables that gives all polynomials in Q the value 0), then the contradiction $1 = 0$ can be derived in* NFPC *from Q.*

Theorem 2. NFPC *(over any field) polynomially-simulates Frege. Specifically,* NFPC *polynomially-simulates $\mathcal{F}$-$\mathcal{PC}$.*

4 Polynomial Calculus over Ordered Formulas

In this section we formulate an algebraic proof system OFPC that operates with noncommutative polynomials from $\mathbb{F}\langle x_1, \ldots, x_n\rangle$, in which every monomial is a product of variables in *nondecreasing order* (from left to right; and according to some fixed linear ordering on the variables), and where polynomials in proofs are written as noncommutative formulas.

Let $X = \{x_1, \ldots, x_n\}$ be a set of variables and let $\mathbb{F}$ be a field. Let $\prec$ be a linear order on the variables X. Let $f = \sum_{j \in J} b_j \mathcal{M}_j$ be a commutative polynomial from $\mathbb{F}[x_1, \ldots, x_n]$, where the b_j's are coefficient from $\mathbb{F}$ and the $\mathcal{M}_j$'s are monomials in the X variables. We define $[\![f]\!] \in \mathbb{F}\langle x_1, \ldots, x_n \rangle$ to be the (unique) noncommutative polynomial $\sum_{j \in J} b_j \cdot [\![\mathcal{M}_j]\!]$, where $[\![\mathcal{M}_j]\!]$ is the (noncommutative) product of all the variables in $\mathcal{M}_j$ such that the order of multiplications respects $\prec$. We denote the image of the map $[\![\cdot]\!] : \mathbb{F}[x_1, \ldots, x_n] \to \mathbb{F}\langle x_1, \ldots, x_n \rangle$ by $\mathcal{G}$.

Definition 4 (Ordered formula). *The class of noncommutative formulas computing polynomials from $\mathcal{G}$ is called the class of* ordered formulas *(under the given fixed linear order $\prec$). We say that an ordered formula F computes the* commutative *polynomial $f \in \mathbb{F}[x_1, \ldots, x_n]$ whenever F computes $[\![f]\!]$.*

Definition 4 enables us to define OFPC in a convenient way, that is, without referring to noncommutative polynomials: the system OFPC is defined similarly to PC, except that the proof-lines are written as ordered formulas:

Definition 5 (PC over ordered formulas (OFPC)). *Let $\pi = (p_1, \ldots, p_m)$ be a PC proof of p_m from some set of initial polynomials Q (that is, p_i are commutative polynomials from the ring of polynomials $\mathbb{F}[x_1, \ldots, x_n]$), and let $\prec$ be some linear order on the variables. The sequence $(p_1, \ldots, p_m)$ in which each p_i is written as an ordered formula (according to the ordering $\prec$), is called an* OFPC *proof of p_m from Q. The* ***size*** *of an* OFPC *proof is the total size of all the ordered formulas appearing in it.*

Similar to the proof system NFPC we define OFPC with no rules expressing the polynomial-ring axioms. The system OFPC will constitute a Cook-Reckhow proof system (that is, verifiable in polynomial-time [whenever the base field and its operations can be efficiently represented]):

Proposition 3. *For any linear order on the variables,* OFPC *is a sound, complete and polynomially-verifiable refutation system for establishing that a collection of polynomial equations over a field does not have* $0, 1$ *solutions. In other words (considering only refutations of polynomial translations of Boolean contradictions)* OFPC *is a Cook-Reckhow proof system.*

5 Simulations, Short Proofs and Separations for OFPC

In this section we are concerned with the relative strength of OFPC. Specifically, we show that OFPC is strictly stronger than the polynomial calculus, polynomial calculus with resolution (PCR) and resolution. For this purpose, we show first that, for any linear order on the variables, OFPC polynomially simulates PCR. Since PCR polynomially simulates both PC and resolution, we get that OFPC also polynomially simulates PC and resolution. Second, we show that OFPC admits polynomial-size refutations of tautologies (formally, families of unsatisfiable collections of polynomial equations) that are hard for PCR.

Proposition 4. *For any linear order on the variables,* OFPC *polynomially simulates PCR (and PC and resolution).*

OFPC polynomially simulates R^0(lin). We now state that OFPC can simulate the proof system R^0(lin) introduced in [21]. This will be used below to establish the OFPC upper bounds. In [21] a refutation system R(lin) was introduced. R(lin) is a refutation system extending resolution to work with disjunctions of linear equations instead of disjunction of literals. R^0(lin) is defined to be a subsystem of R(lin) in which certain restrictions are put on the possible disjunctions of linear equations allowed in a proof. For the precise definition of R(lin) and R^0(lin) we refer the reader to [21]. The main simulation of this subsection is:

Theorem 3. *For any linear order on the variables,* OFPC *polynomially simulates* R^0(lin) *(over large enough fields). Moreover, we can assume that all formulas appearing in the* OFPC *proofs simulating* R^0(lin) *are depth-*3 *ordered formulas.*

Corollaries: short proofs and separations. For natural numbers $m > n$, denote by $\neg\mathrm{FPHP}^m_n$ the following unsatisfiable collection of polynomials: Pigeons $:\forall i \in [m],\ (1 - x_{i,1})\cdots(1 - x_{i,n})$; Functional $:\forall i \in [m]\,\forall k < \ell \in [n],\ x_{i,k}\cdot x_{i,\ell}$; Holes $:\forall i < j \in [m]\,\forall k \in [n],\ \ x_{i,k}\cdot x_{j,k}$.

As a corollary of the polynomial simulation of R^0(lin) by OFPC, and the upper bounds on R^0(lin) proofs demonstrated in [21], we get the following results:

Corollary 1. *For any linear order on the variables, and for any $m > n$ there are polynomial-size (in n)* OFPC *refutations of the m to n pigeonhole principle* FPHP^m_n *(over large enough fields).*

By known lower bounds, we get that OFPC is strictly stronger than resolution. (It can be shown that Corollary 1 implies also a separation of OFPC from PC.) The Tseitin graph tautologies were proved to be hard tautologies for several propositional proof system. We refer the reader to Definition 6.5 in [21] for the precise definition of the Tseitin formulas. We have the following:

Corollary 2. *Let G be an r-regular graph with n vertices, where r is a constant, and fix some modulus p. Then there are polynomial-size (in n)* OFPC *refutations of the corresponding Tseitin* mod p *formulas (over large enough fields).*

This stems from the R^0(lin) polynomial-size refutations of the Tseitin mod p formulas demonstrated in [21]. From the known exponential lower bounds on PCR (and PC and resolution) refutation size of Tseitin mod p tautologies (when the underlying graphs are appropriately expanding; cf. [7,6,3]), we conclude that OFPC is strictly stronger than PCR.

6 Towards Lower Bounds on OFPC Proofs and Related Systems

6.1 Lower Bounds on Product Formulas

We show that the ordered formula size of certain polynomials can increase exponentially when multiplying the polynomials together. In the next section, we use this to suggest an approach for lower bounding the size of OFPC proofs.

Proposition 5. *Let $\mathbb{F}$ be a field, $X := \{x_1, \dots, x_n\}$ be a set of variables and $\prec \subseteq (X \times X)$ be any linear order. Then, for any natural numbers $m \le n$ and $d \le \lfloor n/m \rfloor$, there exist polynomials $f_1, \dots, f_d$ from $\mathbb{F}[x_1, \dots, x_n]$, such that every f_i can be computed by an ordered formula of size $O(m)$ and every ordered formula computing $\prod_{i=1}^{d} f_i$ has size $2^{\Omega(d)}$.*

6.2 A Lower Bound Approach

Here we discuss a simple approach intended to establish lower bounds on OFPC proofs, roughly, by reducing the lower bounds to PC degree lower bounds and using the bound in Section 6.1.

Let $Q_1(\bar{x}), \dots, Q_m(\bar{x})$ be a collection of constant degree (independent of n) polynomials from $\mathbb{F}[x_1, \dots, x_n]$ with no common solutions in $\mathbb{F}$, such that m is polynomial in n. Let $f_1(\bar{y}), \dots, f_n(\bar{y})$ be m homogenous polynomials of the same degree from $\mathbb{F}[y_1, \dots, y_\ell]$, such that the ordered formula size of each $f_i(\bar{y})$ (for some linear order on the variables) is polynomial in n and such that the $f_i(\bar{y})$'s do not have common variables (that is, each $f_i(\bar{y})$ is over disjoint set of variables from $\bar{y}$). Assume that for any distinct $i_1, \dots, i_d \in [n]$ the ordered formula size of $\prod_j^d f_{i_j}(\bar{y})$ is $2^{\Omega(d)}$.

Note 1. By the proof of Proposition 5, the conditions above are easy to achieve. See the full version for details.

Consider the polynomials $Q_1(\bar{x}), \dots, Q_m(\bar{x})$ after applying the substitution:

$$x_i \mapsto f_i(\bar{y}) \,. \tag{1}$$

In other words, consider

$$Q_1(f_1(\bar{y}), \dots, f_n(\bar{y})), \dots, Q_m(f_1(\bar{y}), \dots, f_n(\bar{y})) \,. \tag{2}$$

Note that (2) is also unsatisfiable over $\mathbb{F}$. We suggest to lower bound the OFPC refutations size of (2), based on the following simple idea: it is known that some families of unsatisfiable collections of polynomials require linear $\Omega(n)$ *degree* PC refutations. In other words, every refutation of these polynomials must contain some polynomial of linear degree. By definition, also every OFPC refutation of these polynomials must contain some polynomial of linear degree.

Thus, assume that the initial polynomials $Q = \{Q_1(\bar{x}), \dots, Q_m(\bar{x})\}$ in the $x_1, \dots, x_n$ variables, require linear degree refutations—in fact, an $\omega(\log n)$ degree lower bound would suffice. Thus, every PC refutation contains some polynomial h of degree $\omega(\log n)$. Then, we might expect that every PC refutation of (2) contains a polynomial $g \in \mathbb{F}[\bar{y}]$ which is a substitution instance (under the substitution (1)) of an $\omega(\log n)$-degree polynomial in the $\bar{x}$ variables. This, in turn, leads (under some conditions; see below for an example of such conditions) to a lower bound on OFPC refutations. Specifically, an example of sufficient conditions for super-polynomial OFPC lower bounds, is as follows: every PC refutation of (2) contains a polynomial g so that one of g's homogenous components is a substitution instance (under the substitution (1)) of a degree $\omega(\log n)$ multilinear polynomial from $\mathbb{F}[x_1, \dots, x_n]$. We formalize this argument:

Proposition 6 (Example: conditional OFPC **size lower bounds).** *(Assume the above notations and conditions.)*

IF: *every PC refutation of* (2) *contains a polynomial* $g \in \mathbb{F}[y_1, \ldots, y_\ell]$ *such that for some* $t \leq \deg(g)$*, the* t*-th homogenous component* $g^{(t)}$ *of* g *(that is, the sum of all monomials of total degree* t *in* g*) is a* substitution instance *(under the substitution* (1)*) of a degree* $\omega(\log n)$ *multilinear polynomial from* $\mathbb{F}[x_1, \ldots, x_n]$*;*
THEN: *every* OFPC *refutation of* (2) *is of super-polynomial size (in* n*).*

Note 2. Proposition 6 should serve only as an *example* of the reduction. It is possible that the condition itself, as stated, never holds.

6.3 Obstacles

We can show that the lower bound approach illustrated in the previous subsection does not work for certain substitutions $x_i \mapsto f_i(\bar{y})$. In fact, we can give a general upper bound criterion for OFPC refutations. This criterion is based on the polynomial simulation of $\mathrm{R}^0(\mathrm{lin})$ by OFPC (Theorem 3). We refer the reader to the full paper for the details.

Acknowledgments

I wish to thank Emil jeřabek, Sebastian Müller, Pavel Pudlák and Neil Thapen for helpful discussions on issues related to this paper. I also wish to thank Ran Raz for suggesting this research direction, and Jan Krajíček for inviting me to give a talk at TAMC 2010 on this subject.

References

1. Ajtai, M.: The complexity of the pigeonhole principle. In: Proceedings of the IEEE 29th Ann. Symp. on Found. of Comp. Sci, pp. 346–355 (1988)
2. Alekhnovich, M., Ben-Sasson, E., Razborov, A.A., Wigderson, A.: Space complexity in propositional calculus. SIAM J. Comput. 31(4), 1184–1211 (2002)
3. Alekhnovich, M., Ben-Sasson, E., Razborov, A.A., Wigderson, A.: Pseudorandom generators in propositional proof complexity. SIAM J. Comput. 34(1), 67–88 (2004)
4. Atserias, A., Galesi, N., Pudlák, P.: Monotone simulations of non-monotone proofs. J. Comput. System Sci. 65(4), 626–638 (2002)
5. Atserias, A., Kolaitis, P.G., Vardi, M.Y.: Constraint propagation as a proof system. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 77–91. Springer, Heidelberg (2004)
6. Ben-Sasson, E., Impagliazzo, R.: Random CNF's are hard for the polynomial calculus. In: Proceedings of the IEEE 40th Annual Symposium on Foundations of Computer Science, New York, pp. 415–421. IEEE, Los Alamitos (1999)
7. Buss, S.R., Grigoriev, D., Impagliazzo, R., Pitassi, T.: Linear gaps between degrees for the polynomial calculus modulo distinct primes. J. Comput. System Sci. 62(2), 267–289 (2001)
8. Buss, S.R., Impagliazzo, R., Krajíček, J., Pudlák, P., Razborov, A.A., Sgall, J.: Proof complexity in algebraic systems and bounded depth Frege systems with modular counting. Comput. Complex. 6(3), 256–298 (1996/1997)

9. Clegg, M., Edmonds, J., Impagliazzo, R.: Using the Groebner basis algorithm to find proofs of unsatisfiability. In: Proceedings of the 28th Ann. ACM Symp. on the Theory of Comput, pp. 174–183 (1996)
10. Cook, S.A., Reckhow, R.A.: The relative efficiency of propositional proof systems. The Journal of Symbolic Logic 44(1), 36–50 (1979)
11. Grigoriev, D., Hirsch, E.A.: Algebraic proof systems over formulas. Theoret. Comput. Sci. 303(1), 83–102 (2003)
12. Krajíček, J.: An exponential lower bound for a constraint propagation proof system based on ordered binary decision diagrams. J. Symbolic Logic 73(1), 227–237 (2008)
13. Krajíček, J., Pudlák, P., Woods, A.: An exponential lower bound to the size of bounded depth Frege proofs of the pigeonhole principle. Random Structures Algorithms 7(1), 15–39 (1995)
14. Nisan, N.: Lower bounds for non-commutative computation. In: Proceedings of the 23th Annual ACM Symposium on the Theory of Computing, pp. 410–418 (1991)
15. Pitassi, T.: Algebraic propositional proof systems. In: Descriptive complexity and finite models. DIMACS Ser. Discrete Math. Theoret. Comput. Sci, vol. 31, pp. 215–244. AMS, Providence (1997)
16. Pitassi, T., Beame, P., Impagliazzo, R.: Exponential lower bounds for the pigeonhole principle. Comput. Complexity 3(2), 97–140 (1993)
17. Pudlák, P.: On the complexity of the propositional calculus. In: Sets and proofs. London Math. Soc. Lecture Note Ser., vol. 258, pp. 197–218. Cambridge Univ. Press, Cambridge (Leeds 1997)
18. Raz, R.: Separation of multilinear circuit and formula size. Theory of Computing 2, article 6 (2006)
19. Raz, R.: Multi-linear formulas for permanent and determinant are of super-polynomial size. J. ACM 56(2) (2009)
20. Raz, R., Shpilka, A.: Deterministic polynomial identity testing in non commutative models. Computational Complexity 14(1), 1–19 (2005)
21. Raz, R., Tzameret, I.: Resolution over linear equations and multilinear proofs. Ann. Pure Appl. Logic 155(3), 194–224 (2008)
22. Raz, R., Tzameret, I.: The strength of multilinear proofs. Computational Complexity 17(3), 407–457 (2008)
23. Segerlind, N.: Nearly-exponential size lower bounds for symbolic quantifier elimination algorithms and OBDD-based proofs of unsatisfiability. Electronic Colloquium on Computational Complexity, TR07-009 (January 2007)
24. Tzameret, I.: Studies in Algebraic and Propsitional Proof Complexity. PhD thesis, Tel Aviv University (2008)

Nonlocal Quantum XOR Games for Large Number of Players

Andris Ambainis, Dmitry Kravchenko, Nikolajs Nahimovs, and Alexander Rivosh

Faculty of Computing, University of Latvia

Abstract. Nonlocal games are used to display differences between classical and quantum world. In this paper, we study nonlocal games with a large number of players. We give simple methods for calculating the classical and the quantum values for symmetric XOR games with one-bit input per player, a subclass of nonlocal games. We illustrate those methods on the example of the N-player game (due to Ardehali [Ard92]) that provides the maximum quantum-over-classical advantage.

1 Introduction

Nonlocal games provide a simple framework for studying the differences between quantum mechanics and classical theory. A nonlocal game is a cooperative game of two or more players. Given some information, the players must find a solution, but with no direct communication between any of them.

We can view nonlocal games as games between a *referee* and some number of *players*, where all communication is between the referee and players. Referee chooses settings of the game by telling some information (or *input*) x_i to each of the player. After that each player independently must give back some answer (or *output*) y_i. The rules of the game define a function $f(x_1, x_2, \ldots, y_1, y_2, \ldots)$ which determines whether the players have won or lost.

The most famous example is so called CHSH game [CHSH69]. This is a game between referee from one side and players (Alice and Bob) from the other side. Referee gives one bit to each player. Then he expects equal answers if at least one input bit was 0. If both input bits were 1, he expects different answers. Formally, the rules of this game could be expressed by the table:

$INPUT$	Right answer
$0,0$	$0,0$ or $1,1$
$0,1$	$0,0$ or $1,1$
$1,0$	$0,0$ or $1,1$
$1,1$	$0,1$ or $1,0$

or by the formula:

$$XOR\,(OUTPUT) = AND\,(INPUT)$$

© Springer-Verlag Berlin Heidelberg 2010

Assume that the referee gives to players randomized (uniformly distributed) inputs from $\{(0,0),(0,1),(1,0),(1,1)\}$. For any pair of fixed (deterministic) players' strategies

$$(A:\ \{0,1\} \to \{0,1\}, \quad B:\ \{0,1\} \to \{0,1\})$$

sum of their answers for all 4 different inputs

$$\Big(A(0)+B(0)\Big)+\Big(A(0)+B(1)\Big)+\Big(A(1)+B(0)\Big)+\Big(A(1)+B(1)\Big)$$

is evidently even. But, since sum of Right answers must be odd, any strategy pair will lead to at least one error in these 4 cases. (One may think that some kind of randomized strategies could give better results; the answer is no: an average result of a randomized strategy is calculated as an average result of some set of fixed strategies.) So, provably best average result is $\frac{3}{4} = 0.75$. It can be achieved by answering 0 and ignoring input.

Surprisingly, there is the way to improve this result by permitting players to use an entangled quantum system before start of the game. In this case, correlations between measurement outcomes of different parts of quantum system (in physics, *nonlocality*) can help players to achieve result $\frac{1}{2} + \frac{1}{2\sqrt{2}} = 0.853553\ldots$ [Cir80]. Such games are called *nonlocal* or *entangled*.

In general, the maximum winning probability in a nonlocal game is hard to compute. It is NP-hard to compute it for 2-player games with quantum inputs and outputs or 3-player games classically [Kem08].

XOR games are the most widely studied class of nonlocal games. In a XOR game, players' outputs $y_1, y_2, \ldots, y_N$ are 0-1 valued. The condition describing whether the players have won can depend only on $x_1, x_2, \ldots, x_N$ and $y_1 \oplus y_2 \oplus \ldots \oplus y_N$. XOR games include the CHSH game described above.

For two player XOR games (with inputs $x_1, x_2, \ldots, x_N$ being from an arbitrary set), we know that the maximum success probability of players can be described by a semidefinite program [Cir80] and, hence, can be calculated in polynomial time [CHTW04]. In contrast, computing the classical success probability is NP-hard.

For XOR games with more than two players, examples of specific games providing a quantum advantage are known [Mer90, Ard92, PW+08] and there is some theory in the framework of Bell inequalities [WW01, WW01a, ZB02]. This theory, however, often focuses on questions other than computing classical and quantum winning probabilities — which is our main interest.

In this paper, we consider a restricted case of symmetric multi-player XOR games. For this restricted case, we show that both classical and quantum winning probabilities can be easily calculated. We then apply our methods to the particular case of Ardehali's inequality [Ard92]. The results coincide with [Ard92] but are obtained using different methods (which are more combinatorial in their nature). The advantage of our methods is that they can be easily applied to any symmetric XOR game while those of [Ard92] are tailored to the particular XOR game.

In this paper we will consider only those games, where each player should receive exactly one bit of input and answer exactly one bit of output, and is allowed to operate with one qubit of N-qubit quantum system.

2 Nonlocal XOR Games

A nonlocal N-player game is defined by a sequence of 2^N elements

$$(I_{00\ldots0}, I_{00\ldots1}, \ldots, I_{11\ldots1}),$$

where each element corresponds to some of 2^N inputs and describes all right answers for this input: $I_{x_1\ldots x_N} \subseteq \{0,1\}^N$. Players receive a uniformly random input $x_1, \ldots, x_N \in \{0,1\}$ with the i^{th} player receiving x_i. The i^{th} player then produces an output $y_i \in \{0,1\}$. No communication is allowed between the players but they can use shared randomness (in the classical case) or quantum entanglement (in the quantum case). Players win if $y_1 \ldots y_N \in I_{x_1\ldots x_N}$ and lose otherwise.

For each $I_{x_1\ldots x_N}$, there are 2^{2^N} possible values. Therefore, there are $\left(2^{2^N}\right)^{\left(2^N\right)} = 2^{2^{2N}}$ different games. This means 65536 games for $N = 2$, $\approx 1.8 \cdot 10^{19}$ games for $N = 3$ and practically not enumerable for $N > 3$.

We will concentrate on those of them, which are symmetrical with respect to permuting the players and whose outcome depends only on parity of the sum of the output (or Hamming weight of the output), i.e. on $XOR(|OUTPUT|)$. (Actually, this decision was based not on strict analytics, but rather on the results of numerical experiments: XOR games seem to be the most interesting in their quantum versions.)

Each such XOR game can be described as a string of $N+1$ bits: $P_0P_1\ldots P_N$, where each bit P_i represents the correct right parity of the output sum in the case when the sum of input is i. Typical and important XOR game is the CHSH game: in our terms it can be defined as "$++-$" (even answer if $|INPUT| = 0$ or 1 and odd answer if $|INPUT| = 2$).

3 Methods for Analyzing Nonlocal Games

3.1 Classical XOR Games

In their classical versions XOR games are a good object for analysis and in most cases turn out to have a little outcome for players.

Imagine a classical version of XOR game, for which we want to find optimal classical strategies for players. Each player has 4 different choices — (00), (01), (10), (11). (1^{st} bit here represents the answer on input 0, and 2^{nd} bit represents the answer on input 1. Thus, (ab) denotes a choice to answer a on input 0 and answer b on input 1).

Definition 1 (Classical normalized strategy). *Classical normalized strategy for N-player XOR game is one of the following $2N + 2$ choice sequences:*

$$\begin{array}{l} (00)^{N-k}\ (01)^{k} \\ (00)^{N-1}\ (11) \\ (00)^{N-k}\ (01)^{k-1}\ (10) \end{array}$$

Theorem 1. *For any classical strategy for N-player XOR game there exists a normalized strategy, such that these strategies on equal input answer equal parity.*

Proof. First of all, remember, that we consider only symmetrical games with respect to players permutation. Therefore, we always will order players by their choices.

The second step is choice inversion for a pair of players. If we take any pair of choices and invert both of them, the parity of the output will not change. Thus, we can find the following pairs of choices and make corresponding inversions:

$$\begin{array}{l} (11)\ (11) \to (00)\ (00) \\ (11)\ (10) \to (00)\ (01) \\ (11)\ (01) \to (00)\ (10) \\ (10)\ (10) \to (01)\ (01) \end{array}$$

If it is impossible to find such pair, there is clearly no more than one choice from the set $\{(10)\,,\ (11)\}$, and presence of choice (11) follows that all other choices are (00). In other words, this strategy is normalized.

This trick allows very efficient search for an optimal strategy for classical version of a XOR game. Strategy of form

$$(00)^{N-k}\ (01)^{k}$$

has outcome probability

$$O\left((00)^{N-k}\ (01)^{k}\right) = \frac{\sum\limits_{\substack{0\le i\le N \\ 0\le j\le i \\ (j\equiv 0 (\bmod 2))\veebar(I_i=+)}} \binom{N-k}{i-j}\binom{k}{j}}{2^N}.$$

All other normal strategies has outcomes computable as

$$O\left((00)^{N-1}\ (11)\right) = 1 - O\left((00)^{N}\right)$$
$$O\left((00)^{N-k}\ (01)^{k-1}\ (10)\right) = 1 - O\left((00)^{k}\ (01)^{N-k}\right)$$

(These formulae are for illustration purposes only and won't be refered in this paper.)

3.2 Quantum XOR Games

Consider a possibly non-symmetric XOR game. Let $x_1, \ldots, x_N$ be the inputs to the players. Define $c_{x_1,\ldots,x_N} = 1$ if, to win for these inputs, players must output $y_1, \ldots, y_N$ with XOR being 1 and $c_{x_1,\ldots,x_N} = -1$ if players must output $y_1, \ldots, y_N$ with XOR being 0.

Werner and Wolf [WW01, WW01a] have shown that, for any strategy in quantum version of an XOR game, its bias (the difference between the winning probability p_{win} and the losing probability p_{los}) is equal to

$$f(\lambda_1, \lambda_2, \ldots, \lambda_N) = \left| \frac{1}{2^N} \sum_{x_1,\ldots,x_N \in \{0,1\}} c_{x_1,\ldots,x_N} \lambda_1^{x_1} \lambda_2^{x_2} \ldots \lambda_N^{x_N} \right| \quad (1)$$

for some $\lambda_1, \ldots, \lambda_N$ satisfying $|\lambda_1| = |\lambda_2| = \ldots = |\lambda_N| = 1$. Conversely, for any such $\lambda_1, \ldots, \lambda_N$, there is a winning strategy with the bias being $f(\lambda_1, \ldots, \lambda_N)$.

Lemma 1. *For symmetric XOR games, the maximum of $f(\lambda_1, \ldots, \lambda_N)$ is achieved when $\lambda_1 = \ldots = \lambda_N$.*

Proof. We fix all but two of λ_i. To simplify the notation, we assume that $\lambda_3, \ldots, \lambda_N$ are the variables that have been fixed. Then, (1) becomes

$$a + b\lambda_1 + c\lambda_2 + d\lambda_1\lambda_2$$

for some a, b, c, d. Because of symmetry, we have $b = c$. Thus, we have to maximize

$$a + b(\lambda_1 + \lambda_2) + d\lambda_1\lambda_2. \quad (2)$$

Let $\lambda_1 = e^{i\theta_1}$ and $\lambda_2 = e^{i\theta_2}$. Let $\theta_+ = \frac{\theta_1+\theta_2}{2}$ and $\theta_- = \frac{\theta_1-\theta_2}{2}$. Then, (2) becomes

$$a + be^{i\theta_+}(e^{i\theta_-} + e^{-i\theta_-}) + de^{2i\theta_+} = A + B\cos\theta_-$$

where $A = a + de^{2i\theta_+}$ and $B = 2be^{i\theta_+}$. If we fix θ_+, we have to maximize the expression $A+Bx$, $x \in [-1, 1]$. For any complex A, B, $A+Bx$ is either maximized by $x = 1$ (if the angle between A and B as vectors in the complex plane is at most $\frac{\pi}{2}$) or by $x = -1$ (if the angle between A and B is more than $\frac{\pi}{2}$). If $x = 1$, we have $\lambda_1 = \lambda_2 = \theta_+$. If $x = -1$, we have $\lambda_1 = \lambda_2 = -\theta_+$.

Thus, if $\lambda_1 \neq \lambda_2$, then the value of (1) can be increased by keeping the same $\theta_+ = \frac{\theta_1+\theta_2}{2}$ but changing λ_1 and λ_2 so that they become equal. The same argument applies if $\lambda_i \neq \lambda_j$. □

Thus, we can find the value of a symmetric XOR game by maximizing

$$f(\lambda) = \left| \frac{1}{2^N} \sum_{k=0}^{N} \binom{N}{k} c_k \lambda^k \right| \quad (3)$$

where $c_k = 1$ if $P_k = 1$ and $c_k = -1$ if $P_k = 0$. The maximal $f(\lambda)$ is the maximum possible gap $p_{win} - p_{los}$ between the winning probability p_{win} and the losing probability p_{los}. We have $p_{win} = \frac{1+f(\lambda)}{2}$ and $p_{los} = \frac{1-f(\lambda)}{2}$.

4 Ardehali Game

There are 4 games (equivalent to each other up to the input and/or output inversion), which give the biggest gap between "classical" and "quantum" outcomes. Those games were discovered in the context of Bell inequalities (physics notion closely related to nonlocal games) by Ardehali [Ard92], building on an earlier work by Mermin [Mer90].

They can be described as follows:

$\lvert INPUT \rvert$	0	1	2	3	$\cdots$	N
$XOR(\lvert OUTPUT \rvert)$	$+$	$+$	$-$	$-$	$\cdots$	$\begin{cases} + \text{ if } N \bmod 4 \in \{0,1\} \\ - \text{ otherwise} \end{cases}$

(4)

$\lvert INPUT \rvert$	0	1	2	3	$\cdots$	N
$XOR(\lvert OUTPUT \rvert)$	$+$	$-$	$-$	$+$	$\cdots$	$\begin{cases} + \text{ if } N \bmod 4 \in \{0,3\} \\ - \text{ otherwise} \end{cases}$

$\lvert INPUT \rvert$	0	1	2	3	$\cdots$	N
$XOR(\lvert OUTPUT \rvert)$	$-$	$-$	$+$	$+$	$\cdots$	$\begin{cases} + \text{ if } N \bmod 4 \in \{2,3\} \\ - \text{ otherwise} \end{cases}$

$\lvert INPUT \rvert$	0	1	2	3	$\cdots$	N
$XOR(\lvert OUTPUT \rvert)$	$-$	$+$	$+$	$-$	$\cdots$	$\begin{cases} + \text{ if } N \bmod 4 \in \{1,2\} \\ - \text{ otherwise} \end{cases}$

For each of those games, the maximum winning probability is $p_q = \frac{1}{2} + \frac{1}{2\sqrt{2}}$ for a quantum strategy and $p_c = \frac{1}{2} + \frac{1}{2^{N/2}}$ for a classical strategy [Ard92]. Thus, if we take the ratio $\frac{p_q - 1/2}{p_c - 1/2}$ as the measure of the quantum advantage, these games achieve the ratio of $2^{N/2}$. Similar ratio was earlier achieved by Mermin [Mer90] for a partial XOR game:

$\lvert INPUT \rvert$	0	1	2	3	$\cdots$	N
$XOR(\lvert OUTPUT \rvert)$	$+$	any	$-$	any	$\cdots$	$\begin{cases} + \text{ if } N \bmod 4 \in \{0\} \\ - \text{ if } N \bmod 4 \in \{2\} \end{cases}$

In this game, the input of the players is chosen uniformly at random among all inputs with an even number of 1s. Werner and Wolf [WW01] have shown that this ratio is the best possible.

We now derive the winning probabilities for Ardehali's game using our methods.

4.1 Classical Case

As all of them are symmetrical to each other, we will consider the 1st game only (4). Any normalized strategy for a classical version of such game can be further simplified. Once there exists two players with choices (01) and (01), they can be conversed into (00) and (11) with average outcome remaining the same.

Proof. Let us compare a strategy outcome before and after simplification of type (01) (01) $\rightarrow$ (00) (11). Imagine the situation, where the referee has already produced inputs for all except two players, whose choices are being changed. And he is ready to toss up his coin twice in order to decide, what input to give to remaining players.

If the coin will produce different inputs for these players, their answers will be the same: one will answer 0 and other will answer 1, so the outcome will remain unchanged.

If the coin will produce equal inputs for players — 00 or 11 — it is more tricky case. Let us notice first that the rules of the game require different answers for 00 and for 11. This can be seen from the Table 4: changing $|INPUT|$ by 2, Right answer changes to opposite value. The second fact to notice is that the strategy before the simplification resulted in equal answers on input 00 and on input 11, that is in one correct and one incorrect answer. The third fact is that the strategy after the simplificaion will do the same (but in opposite sequence): one answer will be incorrect and other will be correct.

So, the total average is equal for both strategies.

When none of the simplifications can be applied to the strategy, then this strategy is one from the following set:

$$\begin{array}{l} (00)^N \\ (00)^{N-1}\ (01) \\ (00)^{N-1}\ (10) \\ (00)^{N-1}\ (11) \end{array}$$

For $N = 8n$ an optimal strategy is always $(00)^N$. To show this fact, one can check outcome for all 4 simplified normal strategies. But here we will calculate only the first of them.

Imagine the players are gambling with the referee: they receive 1 in case of win and pay 1 in case of loss. Expected value of their gains after 2^N rounds of the game can be calculated by formula:

$$Outcome\left((00)^{8n}\right) \times 2^{8n} =$$

$$= \sum_{k=0}^{2n-1} \left(\binom{8n}{4k} + \binom{8n}{4k+1} - \binom{8n}{4k+2} - \binom{8n}{4k+3} \right) + \binom{8n}{8n}$$

As one could notice, these four summands inside Σ are approximately equal to each other (and to $\pm\frac{2^N}{4}$), so the total value of the sum is not far from 0. But let us be completely consequent and find precise results.

First, each summand on the odd position has its negation on the same position starting from the end of the sum:

$$+\binom{8n}{1} - \binom{8n}{3} + \binom{8n}{5} - \ldots - \binom{8n}{8n-5} + \binom{8n}{8n-3} - \binom{8n}{8n-1} = 0$$

So, remaining expression (with fake summand $-\binom{8n}{8n+2} = 0$ appended for the reason of simplicity) is the following:

$$Outcome\left((00)^{8n}\right) \times 2^{8n} = \sum_{k=0}^{2n}\left(\binom{8n}{4k} - \binom{8n}{4k+2}\right) \tag{5}$$

Further precise calculations consist mainly of $\binom{N}{K}$ replacements with $\binom{N-2}{K-2} + 2\binom{N-2}{K-1} + \binom{N-2}{K}$ (this equality is not quite evident, but can be proved trivially by induction).

$$\begin{aligned}
&\sum_{k=0}^{2n}\left(\binom{8n}{4k} - \binom{8n}{4k+2}\right)\\
&\quad= \sum_{k=0}^{2n}\left(\binom{8n-2}{4k-2} + 2\binom{8n-2}{4k-1} + \binom{8n-2}{4k} - \right.\\
&\qquad\qquad \left. - \binom{8n-2}{4k} - 2\binom{8n-2}{4k+1} - \binom{8n-2}{4k+2}\right)\\
&\quad= 2\sum_{k=0}^{2n}\left(\binom{8n-2}{4k-1} - \binom{8n-2}{4k+1}\right)
\end{aligned}$$

On the last step we again removed antipode summands $\binom{8n-2}{4k-2}$ and $-\binom{8n-2}{4k+2}$. Remaining expression can be further transformed to

$$\begin{aligned}
&2\sum_{k=0}^{2n}\left(\binom{8n-2}{4k-1} - \binom{8n-2}{4k+1}\right) =\\
&\quad= 2\sum_{k=0}^{2n}\left(\binom{8n-4}{4k-3} + 2\binom{8n-4}{4k-2} + \binom{8n-4}{4k-1} - \right.\\
&\qquad\qquad \left. - \binom{8n-4}{4k-1} - 2\binom{8n-4}{4k} - \binom{8n-4}{4k+1}\right) =\\
&\quad= 4\sum_{k=0}^{2n}\left(\binom{8n-4}{4k-2} - \binom{8n-4}{4k}\right)
\end{aligned}$$

On the last step we again removed antipode summands $\binom{8n-4}{4k-3}$ and $-\binom{8n-4}{4k+1}$. The resulting sum after removing some zero summands becomes

$$4\sum_{k=0}^{2(n-\frac{1}{2})}\left(\binom{8(n-\frac{1}{2})}{4k+2} - \binom{8(n-\frac{1}{2})}{4k}\right)$$

It turns out to be equal to (5) (for n decreased by $\frac{1}{2}$) multiplied by -4. Exactly the same technique shows that

$$Outcome\left((00)^{8n}\right) \times 2^{8n} = (-4)^2 Outcome\left((00)^{8(n-1)}\right) \times 2^{8(n-1)}$$

and thus immediately provides an induction step for proving the claim:

$$Outcome\left((00)^{8n}\right) \times 2^{8n} = 16^n$$

Replacing n with $\frac{N}{8}$ and dividing all expression by the number of rounds 2^N, the best expected outcome in classical version of Ardehali game is

$$Outcome\left((00)^{N}\right) = \left(\frac{1}{\sqrt{2}}\right)^N$$

While the particular manipulations above are specific to Ardehali's game, the overall method of evaluating a sum of binomial coefficients applies to any symmetric XOR game.

4.2 Quantum Case

The value of the Ardehali's game can be obtained by maximizing the one-variable expression in equation (3). In the case of Ardehali's game, the maximum of this expression is $\frac{1}{\sqrt{2}}$ and it is achieved by $\lambda = e^{i\theta}$ where $\theta = \frac{(2N+1) \bmod 8}{N}\pi + k\frac{2\pi}{N}$.

The result $f(\lambda) = \frac{1}{\sqrt{2}}$ corresponds to the winning probability of $p_{win} = \frac{1}{2} + \frac{1}{2\sqrt{2}}$. The winning strategy can be obtained by reversing the argument of [WW01] and going from λ to transformations for the N players. There are infinitely many possible sets of strategies for each of the given θ. One of these strategies is described in [Ard92]. We include example of another strategy in the appendix.

The optimality of $p_{win} = \frac{1}{2} + \frac{1}{2\sqrt{2}}$ can be shown by a very simple argument, which does not involve any of the machinery above.

Theorem 2. $\frac{1}{2} + \frac{1}{2\sqrt{2}}$ *is the best possible probability for quantum strategy.*

Proof. We modify the game by providing the inputs and the outputs of the first $N-2$ players to the $(N-1)^{\text{st}}$ and N^{th} players. Clearly, this makes the game easier: the last two players can still use the previous strategy, even if they have the extra knowledge.

Let k be the number of 1s among the first $N-2$ inputs. Then, we have the following dependence of the result on the actions of the last two players.

$x_1 + x_2 + \ldots + x_{N-2}$	$x_{N-1} + x_N$
	0 1 2
k	$+ + -$ if $k \bmod 4 = 0$
k	$+ - -$ if $k \bmod 4 = 1$
k	$- - +$ if $k \bmod 4 = 2$
k	$- + +$ if $k \bmod 4 = 3$

In either of the 4 cases, we get a game (for the last two players) which is equivalent to the CHSH game and, therefore, cannot be won with probability more than $\frac{1}{2} + \frac{1}{2\sqrt{2}}$.

The fact that $\frac{1}{2} + \frac{1}{2\sqrt{2}}$ is the best winning probability has been known before [Ard92]. But it appears that we are the first to observe that this follows by a simple reduction to the CHSH game.

References

[Ard92] Ardehali, M.: Bell inequalities with a magnitude of violation that grows exponentially with the number of particles. Physical Review A 46, 5375–5378 (1992)

[CHSH69] Clauser, J., Horne, M., Shimony, A., Holt, R.: Phys. Rev. Lett. 23, 880 (1969)

[CHTW04] Cleve, R., Høyer, P., Toner, B., Watrous, J.: Consequences and limits of nonlocal strategies. In: Proceedings of the 19th IEEE Conference on Computational Complexity (CCC 2004), pp. 236–249 (2004)

[Cir80] Tsirelson, B.S.: Quantum generalizations of Bell's inequality. Letters in Mathematical Physics 4(2), 93–100 (1980)

[Kem08] Kempe, J., Kobayashi, H., Matsumoto, K., Toner, B., Vidick, T.: Entangled Games are Hard to Approximate. In: Proceedings of FOCS 2008, pp. 447–456 (2008)

[Mer90] Mermin, D.: Extreme Quantum Entanglement in a Superposition of Macroscopically Distinct States. Physical Review Letters 65, 15 (1990)

[PW+08] Perez-Garcia, D., Wolf, M.M., Palazuelos, C., Villanueva, I., Junge, M.: Unbounded violation of tripartite Bell inequalities. Communications in Mathematical Physics 279, 455 (2008)

[WW01] Werner, R.F., Wolf, M.M.: Bell inequalities and Entanglement. Quant. Inf. Comp. 1(3), 1–25 (2001)

[WW01a] Werner, R.F., Wolf, M.M.: All multipartite Bell correlation inequalities for two dichotomic observables per site. Phys. Rev. A 64, 032112 (2001)

[ZB02] Zukowski, M., Bruckner, C.: Bell's theorem for general N-qubit states. Phys. Rev. Lett. 88, 210401 (2002)

A Appendix

A common behavior for a player in quantum nonlocal game is to perform some local operation on his qubit, perform a measurement in the standard basis and answer the result of the measurement. In other words, a choice for a player can be expressed with two matrices: one for input 0 and other for input 1. In fact, players may use equal strategies to achieve the best outcome. So optimal strategy for any quantum XOR game can be found and proved with numerical optimization quite simply. For quantum version of Ardehali game, the two matrices for all players look like the following:

For input 0 For input 1

$$\frac{1}{\sqrt{2}}\begin{pmatrix} e^{i\left(\frac{\pi}{2}+\gamma\right)} & 1 \\ -1 & e^{-i\left(\frac{\pi}{2}+\gamma\right)} \end{pmatrix} \qquad \frac{1}{\sqrt{2}}\begin{pmatrix} e^{i\gamma} & 1 \\ -1 & e^{-i\gamma} \end{pmatrix}$$

with angle γ depending on the number of players: in fact, it is $\frac{(2N+1) \bmod 8}{4N}\pi$. Assuming x as input bit, these two matrices can be described as one:

$$\frac{1}{\sqrt{2}}\begin{pmatrix} e^{i\left(\frac{\pi}{2}\cdot(1-x)+\gamma\right)} & e^{0} \\ e^{i\pi} & e^{i\left(-\frac{\pi}{2}\cdot(1-x)-\gamma\right)} \end{pmatrix} \tag{6}$$

Consider a quantum system in starting state Ψ_{GHZ}. Lets express all local operations, that players apply to their qubits during the game, as a tensor product of N matrices $M = C_1 \otimes \ldots \otimes C_N$. Each cell of M can be calculated as follows:

$$M_{[j_1 \ldots j_N,\, i_1 \ldots i_N]} = \prod_{k=1}^{N} C_{k\ [j_k,\, i_k]}$$

where $i_1, \ldots, i_N,\ j_1, \ldots, j_N \in \{0,1\}$.

After all local operations are complete, lets express the final state directly as sum of its amplitudes

$$\sum_{y_1,\ldots,y_N \in \{0,1\}} \alpha_{y_1 \ldots y_N} |y_1 \ldots y_N\rangle = M\left(\tfrac{1}{\sqrt{2}}|00\ldots0\rangle + \tfrac{1}{\sqrt{2}}|11\ldots1\rangle\right)$$

Consider a value of arbitrary amplitude $\alpha_{y_1 \ldots y_N}$. As there are only two nonzero amplitudes in the starting state, any $\alpha_{y_1 \ldots y_N}$ will consist of two summands:

$$\alpha_{y_1 \ldots y_N} = \tfrac{1}{\sqrt{2}} \prod_{k=1}^{N} C_{k\ [0,\, y_k]} + \tfrac{1}{\sqrt{2}} \prod_{k=1}^{N} C_{k\ [1,\, y_k]}$$

Assuming players got N-bit input $x_1 \ldots x_N$, lets substitute values from (6) for each C_k:

$$\begin{aligned} \alpha_{y_1 \ldots y_N} &= \tfrac{1}{\sqrt{2}} \prod_{k=1}^{N} \tfrac{1}{\sqrt{2}} e^{i\left(\left(\gamma+\frac{\pi}{2}(1-x_k)\right)\cdot(1-y_k)\right)} + \\ &\quad + \tfrac{1}{\sqrt{2}} \prod_{k=1}^{N} \tfrac{1}{\sqrt{2}} e^{i\left(\pi+\left(\frac{\pi}{2}-\gamma+\frac{\pi}{2}\cdot x_k\right)\cdot y_k\right)} = \\ &= \left(\tfrac{1}{\sqrt{2}}\right)^{N+1} e^{i\sum_{k=1}^{N}\left(\gamma+\frac{\pi}{2}(1-x_k)\right)\cdot(1-y_k)} \\ &\quad + \left(\tfrac{1}{\sqrt{2}}\right)^{N+1} e^{i\sum_{k=1}^{N}\left(\pi+\left(\frac{\pi}{2}-\gamma+\frac{\pi}{2}\cdot x_k\right)\cdot y_k\right)} \end{aligned}$$

Now we are interested mainly in difference between rotation angles on the complex plane for these two summands. That is, in

$$\begin{aligned} &\sum_{k=1}^{N} \left[\left(\gamma+\tfrac{\pi}{2}(1-x_k)\right)(1-y_k) - \left(\pi+\left(\tfrac{\pi}{2}-\gamma+\tfrac{\pi}{2}x_k\right)y_k\right)\right] \\ &= \sum_{k=1}^{N} \left[\left(\gamma+\tfrac{\pi}{2}-\tfrac{\pi}{2}x_k-\gamma y_k-\tfrac{\pi}{2}y_k+\tfrac{\pi}{2}x_k y_k\right) - \left(\pi+\tfrac{\pi}{2}y_k-\gamma y_k+\tfrac{\pi}{2}x_k y_k\right)\right] \\ &= \sum_{k=1}^{N} \left(\gamma+\tfrac{\pi}{2}-\tfrac{\pi}{2}x_k-\tfrac{\pi}{2}y_k-\pi-\tfrac{\pi}{2}y_k\right) \\ &= \sum_{k=1}^{N} \left(\gamma-\tfrac{\pi}{2}-\tfrac{\pi}{2}x_k-\pi y_k\right) \end{aligned}$$

Let us concentrate now on the case $N = 4n$ and $\gamma = \frac{1}{4N}\pi$ (but similar reasoning stays for any N). In this case the difference is expressed by (throwing out $\frac{\pi}{2} \times N \equiv 0 \pmod{2\pi}$, which is now redundant)

$$\tfrac{1}{4}\pi - \tfrac{1}{2}\pi \sum_{k=1}^{N} (x_k + 2y_k)$$

By modulus 2π it is equal to value from Table 1.

Table 1. Amplitude angle values for different inputs

$\lvert INPUT\rvert$	$\lvert OUTPUT\rvert = y_1 + y_2 + \ldots + y_N$				
$= x_1 + \ldots + x_N$	0	1	2	3	$\ldots$
0	$\boxed{\frac{1}{4}\pi}$	$-\frac{3}{4}\pi$	$\boxed{\frac{1}{4}\pi}$	$-\frac{3}{4}\pi$	$\cdots$
1	$\boxed{-\frac{1}{4}\pi}$	$\frac{3}{4}\pi$	$\boxed{-\frac{1}{4}\pi}$	$\frac{3}{4}\pi$	$\cdots$
2	$-\frac{3}{4}\pi$	$\boxed{\frac{1}{4}\pi}$	$-\frac{3}{4}\pi$	$\boxed{\frac{1}{4}\pi}$	$\cdots$
3	$\frac{3}{4}\pi$	$\boxed{-\frac{1}{4}\pi}$	$\frac{3}{4}\pi$	$\boxed{-\frac{1}{4}\pi}$	$\cdots$
4	$\boxed{\frac{1}{4}\pi}$	$-\frac{3}{4}\pi$	$\boxed{\frac{1}{4}\pi}$	$-\frac{3}{4}\pi$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

If angle between two summands (both of the same length $\left(\frac{1}{\sqrt{2}}\right)^{N+1}$) is $\boxed{\pm\frac{1}{4}\pi}$, then their sum is

$$\left(\frac{1}{\sqrt{2}}\right)^{N-1} \cos\frac{\pi}{8} = \left(\frac{1}{\sqrt{2}}\right)^{N-1} \frac{\sqrt{2+\sqrt{2}}}{2} \tag{7}$$

If angle between two summands (both of the same length $\left(\frac{1}{\sqrt{2}}\right)^{N+1}$) is $\pm\frac{3}{4}\pi$, then their sum is $\left(\frac{1}{\sqrt{2}}\right)^{N-1} \cos\frac{3\pi}{8} = \left(\frac{1}{\sqrt{2}}\right)^{N-1} \frac{\sqrt{2-\sqrt{2}}}{2}$.

As one can see from the Table 1, bigger amplitudes always correspond to correct answers (and smaller amplitudes correspond to incorrect answers). Sum of the squares of formula (7), i.e. the measurement result for any fixed input, will give the probability of right answer:

$$\sum_{Angle=\boxed{\pm\frac{1}{4}\pi}} \left(\left(\frac{1}{\sqrt{2}}\right)^{N-1} \frac{\sqrt{2+\sqrt{2}}}{2}\right)^2 = \frac{1}{2} + \frac{1}{2\sqrt{2}}$$

Note that this probability is stable: it remains the same for all possible inputs from the referee.

Nontriviality for Exponential Time w.r.t. Weak Reducibilities

Klaus Ambos-Spies and Timur Bakibayev

University of Heidelberg, Im Neuenheimer Feld 294, D-69120 Heidelberg, Germany

Abstract. A set A is nontrivial for the linear exponential time class $\mathrm{E} = \mathrm{DTIME}(2^{lin})$ if $A \in \mathrm{E}$ and the sets from E which can be reduced to A are not from a single level $\mathrm{DTIME}(2^{kn})$ of the linear exponential hierarchy. Similarly, a set A is nontrivial for the polynomial exponential time class $\mathrm{EXP} = \mathrm{DTIME}(2^{poly})$ if $A \in \mathrm{EXP}$ and the sets from EXP which can be reduced to A are not from a single level $\mathrm{DTIME}(2^{n^k})$ of the polynomial exponential hierarchy (see [1]). Here we compare the strength of the nontriviality notions with respect to the underlying reducibilities where we consider the polynomial-time variants of many-one, bounded truth-table, truth-table, and Turing reducibilities. Surprisingly, the results obtained for E and EXP differ. While the above reducibilities yield a proper hierarchy of nontriviality notions for E, nontriviality for EXP under many-one reducibility and truth-tab! le reducibility coincides.

1 Introduction

A set A is *nontrivial* for $\mathrm{E} = \mathrm{DTIME}(2^{lin})$ (or E-*nontrivial* for short) if there are arbitrarily complex sets from E which can be reduced to A, i.e., if for any $k \geq 1$ there is a set $B \in \mathrm{E}$ reducible to A which is 2^{kn}-complex (i.e., $B \notin \mathrm{DTIME}(2^{kn})$). Similarly, a set A is *nontrivial* for $\mathrm{EXP} = \mathrm{DTIME}(2^{poly})$ if for any $k \geq 1$ there is a set $B \in \mathrm{EXP} \setminus \mathrm{DTIME}(2^{n^k})$ which can be reduced to A. Nontriviality which was introduced by the authors in [1] was inspired by Lutz's concept of weak completeness. While a set $A \in \mathrm{C}$ is complete for a complexity class C in the classical sense if *all* problems in C can be reduced to A, Lutz [9] proposed to call a set $A \in \mathrm{C}$ *weakly complete* for C if a *nonnegligible part* of problems in C can be reduced to A. Lutz formalized the idea of weak completeness for the exponential time classes E and EXP by introducing some resource bounded (pseudo) measures on these classes and by calling a subclass of E and EXP negligible if it has measure 0 in E and EXP, respectively. As one can easily show, weakly complete sets for E and EXP in the sense of Lutz [9] are E-nontrivial and EXP-nontrivial, respectively, and in [1] it is argued that E-nontriviality and EXP-nontriviality are the *weakest* meaningful weak completeness notions for the corresponding exponential time classes.

The classical approach for generalizing completeness is to generalize (i.e., to weaken) the underlying reducibility. So one might replace the polynomial time

© Springer-Verlag Berlin Heidelberg 2010

bounded many-one reducibility (p-m-reducibility for short) on which completeness is usually based by more general polynomial-time reducibilities like the polynomial time variants of bounded truth-table reducibility (p-btt) or truth-table reducibility (p-tt) or Turing reducibility (p-T). As Watanabe [10] has shown, these more general reducibilities also yield more general completeness notions for the exponential time classes. For Lutz's weak completeness notions for E and EXP the corresponding separations have been obtained by Ambos-Spies, Mayordomo and Zheng [3].

Here we address the corresponding questions for nontriviality where we also consider the question of possible trade-offs: If arbitrarily complex sets from E - or even all sets from E - can be reduced to a set $A \in$ E by some weaker reducibility, can we also reduce arbitrarily complex sets from E to A by some stronger reducibility (and, similarly, for EXP)?

For the investigation of these questions, the following phenomenon has to be taken into account. While, by a simple padding argument, hardness for E and EXP coincide (whence a set $A \in$ E is E-complete if and only if it is EXP-complete), surprisingly, for Lutz's weak completeness only one direction holds. Namely any weakly E-complete set is weakly EXP-complete but there are sets in E which are weakly EXP-complete but not weakly E-complete (see Juedes and Lutz [8]). For the still weaker nontriviality notions, E-nontriviality and EXP-nontriviality are in fact independent (see Ambos-Spies and Bakibayev [2]), i.e., there are sets in E which are E-nontrivial but not EXP-nontrivial and vice versa.

This difference in the nontriviality notions for E and EXP is also manifested in a quite surprising way in our results here. While for E the hierarchy of the nontriviality notions under the weak polynomial time reducibilities completely mirrors Watanabe's separation results for the corresponding completeness notions, for EXP, nontriviality under truth-table reducibility and nontriviality under many-one reducibility coincide.

The outline of the paper is as follows. In Section 2 we show that, for E and EXP, nontriviality under truth-table reducibility is stronger than nontriviality under Turing reducibility. In fact we show that there is a T-complete set A for E which is neither tt-nontrivial for E nor tt-nontrivial for EXP. So the fact that *all* sets in E can be recovered from a set A by a Turing reduction does not imply that there are arbitrarily complex sets in E which can be recovered from A by some truth-table reductions. In Section 4 we give corresponding separations of many-one, bounded truth-table and truth-table reducibility for E whereas in Section 3 we prove the coincidence of EXP-nontriviality under many-one reducibility with EXP-nontriviality under truth-table reducibility. Due to lack of space, some of the proofs are only sketched or omitted.

We assume familiarity with the basic notions of structural complexity theory (see e.g. Balcázar et al. [4] and [5] for unexplained notation). In the following we let $\mathrm{E}_k = \mathrm{DTIME}(2^{kn})$ and $\mathrm{EXP}_k = \mathrm{DTIME}(2^{k^n})$. Then a set $A \in$ E is r-E-*nontrivial* if, for any $k \geq 1$, there is a set $B_k \in \mathrm{E} \setminus \mathrm{E}_k$ such that $B_k \leq^p_r A$; and A is r-E-*trivial* otherwise. Similarly, $A \in$ EXP is r-EXP-*nontrivial* if, for any $k \geq 1$, there is a set $B_k \in \mathrm{EXP} \setminus \mathrm{EXP}_k$ such that $B_k \leq_r^p A$; and A is r-EXP-*trivial* otherwise.

2 Turing Completeness vs. Truth-Table Nontriviality

We start with separating nontriviality (for E and EXP) under Turing and truth-table reducibility.

Theorem 1. *There is a T-*E*-complete set A such that A is tt-trivial for* E *and* EXP*.*

Proof. Fix an m-complete set C for E such that $C \in \mathrm{E}_1$. It suffices to define a set A such that

$$C \leq_T^p A \tag{1}$$

$$A \in \mathrm{E}_1, and \tag{2}$$

$$\forall\, B\ (B \leq_{tt}^p A \Rightarrow B \in \mathrm{E}_6) \tag{3}$$

hold. Namely, (1) and (2) guarantee that A is T-complete for E while, by (3), A is tt-trivial for E and EXP.

We first describe a framework for constructing sets which will guarantee (1) and (2), and then we define a set A within this framework which will satisfy condition (3) too.

In order to guarantee (1) we define a p-Turing reduction of C to A as follows. For any string $z \neq \lambda$, let

$$CODE(z) = \{\langle z, y\rangle : |y| \leq 3|z|^2 + 1\}$$

be the set of z-*codes* where the pairing function $\langle , \rangle$ is defined by $\langle z, y\rangle = 0^{4|z|}1zy$. Then, in the course of the construction of A, we define a string $code(z)$ of length $3|z|^2 + 1$ such that the last bit of $code(z)$ is the value of $C(z)$, i.e.,

$$C(z) = code(z)(3|z|^2), \tag{4}$$

and we put a z-code $\langle z, y\rangle$ into A if and only if y is an initial segment of $code(z)$ thereby guaranteeing

$$A \cap CODE(z) = \{\langle z, y\rangle : y \sqsubseteq code(z)\}. \tag{5}$$

Obviously, this ensures that $C \leq_T^p A$ since, by (5), A can compute $code(z)$ by a standard prefix search, and, by (4), $code(z)$ gives the value of $C(z)$.

Strings will be put into A only by the above coding procedure. So

$$A = \bigcup_{z \in \{0,1\}^+} \{0^{4|z|}1zy : y \sqsubseteq code(z)\} = \bigcup_{z \in \{0,1\}^+} \{\langle z, y\rangle : y \sqsubseteq code(z)\}. \tag{6}$$

Now, for a string z of length $n \geq 1$, $\mathit{code}(z)$ will consist of n segments of length $3n$ and the final coding bit, i.e.,

$$\mathit{code}(z) = v_1^z \ \ldots \ v_n^z \ C(z) \quad \text{where } n = |z| \text{ and } |v_1^z| = \cdots = |v_n^z| = 3n. \tag{7}$$

Moreover, these segments will be chosen so that

$$v_1^z \ldots v_m^z \ (1 \leq m \leq |z|) \ \text{ can be computed in } O(\mathit{poly}(|z|) \cdot 2^{4m}) \text{ steps.} \tag{8}$$

Note that, by $C \in \mathrm{E}_1$, (7) and (8) guarantee that

$$\mathit{code}(z) \text{ can be computed in } O(\mathit{poly}(|z|) \cdot 2^{4|z|}) \leq O(2^{5|z|}) \text{ steps.} \tag{9}$$

This allows us to argue that (2) holds, i.e., that $A \in \mathrm{E}_1$, as follows. Given a string x, it follows from (6) that x is in A if and only if there is a string z such that $x = 0^{4|z|}1zy$ for some initial segment y of $\mathit{code}(z)$. But, by the above observation on the complexity of $\mathit{code}(z)$ and by $|x| \geq 5|z|$, this can be decided in $O(\mathit{poly}(|x|) + 2^{5|z|}) \leq O(2^{|x|})$ steps.

Having described the frame for the construction, we now show how, for given z of length $n \geq 1$, the segments v_m^z $(1 \leq m \leq n)$ of $\mathit{code}(z)$ can be chosen so that (8) is satisfied and such that, for the corresponding set A defined according to (6) and (7), A satisfies condition (3). Since, by the preceding discussion, A will satisfy (1) and (2) too, this will complete the proof.

We start with some notation. Fix a standard enumeration $\{M_e : e \geq 0\}$ of the polynomial-time bounded oracle Turing machines such that, for any oracle X, the run time of M_e^X on inputs of length n is bounded by $p_e(n)$ (uniformly in e and n) where the polynomials p_e are chosen such that $n \leq p_e(n) \leq p_{e+1}(n)$ and $p_e(n)^2 < 2^n$ for all e and n with $e \leq n$. Let $Q_e(x)$ be the set of oracle queries made by $M_e^{\emptyset}$ on input x. Note that, for e and x such that $e \leq |x|$, $Q_e(x)$ consists of less than $p_e(|x|) < 2^{|x|}$ strings, each having length less than $p_e(|x|) < 2^{|x|}$, and $Q_e(x)$ can be computed in time $p_e(|x|) < 2^{|x|}$. Finally, note that if M_e describes a p-tt-reduction then M_e is nonadaptive, i.e., the query set of M_e on input x does not depend on the oracle set whence $Q_e(x)$ is the query set of $M_e^A(x)$.

Now, given a string z of length $n \geq 1$, the segments $v_1^z, \ldots, v_n^z$ of $\mathit{code}(z)$ are inductively defined as follows. Given m with $1 \leq m \leq n$ and the strings $v_1^z, \ldots v_{m-1}^z$, let v_m^z be the least string v of length $3n$ such that

$$\forall\, e < m \,\forall\, x \in \{0,1\}^m \,\forall\, y \in Q_e(x)\ (0^{4|z|}1zv_1^z \ldots v_{m-1}^z v \not\sqsubseteq y). \tag{10}$$

In order to show that v_m^z is well defined (i.e., that there is a string v satisfying (10)) and that the segments v_m^z of $\mathit{code}(z)$ satisfy (8), we first observe that the set $Q = \bigcup_{e<m,|x|=m} Q_e(x)$ of the strings y which are not allowed to extend $0^{4|z|}1zv_1^z \ldots v_{m-1}^z v_m^z$ has cardinality less than 2^{2m} and can be listed in time $O(2^{2m})$. Note that there are m numbers $e < m$ and 2^m strings x of length m. Moreover, as observed above, for each such e and x, $|Q_e(x)| < p_e(m)$. So, by choice of the polynomials p_e (and by $e < m$),

$$|Q| < m \cdot 2^m \cdot p_e(m) \leq p_m(m)^2 \cdot 2^m \leq 2^{2m}.$$

The existence of a listing of Q in time $O(2^{2m})$ follows by a similar argument from the observation that each of the sets $Q_e(x)$ can be listed in time $\leq p_e(m)$.

Now the existence of a string v of length $3n$ as in (10) is immediate since there are 2^{3n} strings v of length $3n$ but there are are only less than 2^{2n} strings y which have to be avoided as extensions of $0^{4|z|}1zv_1^z \dots v_{m-1}^z v$.

Condition (8) is established by induction on m. Given m and $v_1^z, \dots, v_{m-1}^z$ it suffices to show that v_m^z can be computed in $O(poly(n) \cdot 2^{4m})$ steps. Since Q can be listed in time $O(2^{2m})$ and since $z, v_1^z, \dots, v_{m-1}^z$ are given, in $poly(n) \cdot 2^{2m}$ steps we can list the set Q' of all strings w of length $3n$ such that $0^{4|z|}1zv_1^z \dots v_{m-1}^z w$ is an initial segment of a string y in Q. So, by sorting Q', in $O(poly(n) \cdot 2^{4m})$ steps we can find the least string v of length $3n$ such that $v \notin Q'$ and, obviously, v_m^z is the least such string.

It remains to show that (3) is satisfied. So fix a set B such that $B \leq_{tt}^p A$. We have to show that $B \in \mathrm{E}_6$.

Fix e such that M_e is nonadaptive and $B = M_e^A$. Then, given a string x where w.l.o.g. $e < |x|$, $B(x)$ can be computed in $O(2^{6|x|})$ steps by simulating $M_e^A(x)$ as follows. Since M_e is nonadaptive, $Q_e(x)$ is the query set of this computation. So knowing $A(y)$ for all strings $y \in Q_e(x)$ allows us to compute $M_e^A(x)$ in polynomial time. Hence, by $|Q_e(x)| \leq 2^{|x|}$, it suffices to compute $A(y)$ for a given $y \in Q_e(x)$ in $O(2^{5|x|})$ steps.

In order to compute $A(y)$, first decide whether y is an element of a code set $CODE(z)$ and if so compute the unique z such that $y \in CODE(z)$ and the unique w such that $y = 0^{4|z|}1zw$. Since $|y| < p_e(|x|)$, this can be done in $poly(|x|)$ steps. Now if y is not in any code set then, by (6), $y \notin A$. If $y = 0^{4|z|}1zw$ is a z-code then, by (6), $y \in A$ iff $y \sqsubseteq 0^{4|z|}1z code(z)$. For deciding the latter, distinguish the following two cases. If $|z| \leq |x|$ then, by (9), $code(z)$ can be computed in $O(2^{5|z|}) \leq O(2^{5|x|})$ steps. Finally, if $|x| < |z|$ then, by $e < |x| < |z|$ and by choice of $v_{|x|}^z$ (see (10)), $y \sqsubseteq 0^{4|z|}1z code(z)$ if and only if $y \sqsubseteq 0^{4|z|}1zv_1^z \dots v_{|x|}^z$. Moreover, by (8), $v_1^z \dots v_{|x|}^z$ can be computed in $O(poly(|z|) \cdot 2^{4|x|})$ steps, and, by $|z| < |x|$, $O(poly(|z|) \cdot 2^{4|x|}) \leq O(2^{5|x|})$.

This completes the proof.

3 Collapse of Truth-Table Nontriviality for EXP

In contrast to the hierarchy theorems for EXP-completeness by Watanabe [10] and for weak EXP-completeness by Ambos-Spies, Mayordomo and Zheng [3], here we show that tt-nontriviality for EXP and m-nontriviality for EXP coincide.

Theorem 2. *For any set $A \in$ EXP the following are equivalent.*

(i) A is m-nontrivial for EXP.
(ii) A is tt-nontrivial for EXP.

Proof. For a proof of the nontrivial direction assume that A is tt-nontrivial for EXP and fix $k \geq 1$. It suffices to show that there is a set B such that $B \leq_m^p A$

and $B \notin \mathrm{EXP}_k$. (Note that, by $A \in \mathrm{EXP}$ and by downward closure of EXP under $\leq^p_m$, $B \leq^p_m A$ will imply that $B \in \mathrm{EXP}$.)

By tt-nontriviality of A, fix a set C such that $C \in \mathrm{EXP} \backslash \mathrm{EXP}_{k+1}$ and $C \leq^p_{tt} A$. Moreover, fix a nonadaptive oracle Turing machine M such that $C \leq^p_{tt} A$ via M and let p be a polynomial time-bound for M. For any input string x let $q(x,1), \ldots, q(x,n_x)$ be the list of oracle queries of M on input x (with empty oracle) in order of appearance. Finally, define the set B by

$$B = \{\langle x, z_n\rangle : n \leq n_x \ \&\ q(x,n) \in A\}$$

where z_n is the nth string with respect to the length-lexicographical ordering and the coded pair $\langle x,y\rangle$ is defined by $\langle x,y\rangle = 1^{|x|}0xy$.

We claim that B has the required properties.

Obviously, $B \leq^p_m A$ via f where f is defined by

$$f(y) = \begin{cases} q(x,n) & \text{if } y = 1^{|x|}0xz_n \ \&\ n \leq n_x \\ 0 & \text{otherwise.} \end{cases}$$

It remains to show that $B \notin \mathrm{EXP}_k$. For a contradiction assume $B \in \mathrm{EXP}_k$. Then, for given x, $C(x)$ can be computed by the following procedure.

– Compute the queries $q(x,0), \ldots, q(x,n_x)$ by running $M^{\emptyset}$ on input x. (This can be done in $p(|x|)$ steps.)

– For $n \leq n_x$ compute $A(q(x,n))$ by using the identity

$$A(q(x,n)) = B(\langle x, z_n\rangle) = B(1^{|x|}0xz_n).$$

(Note that $n \leq n_x < p(|x|)$ and that the length of z_n is logarithmic in n whence $|1^{|x|}0xz_n| \leq 3|x| + O(1)$. So, by assumption on B, this part of the procedure can be completed in $O(p(|x|) \cdot 2^{(3|x|)^k})$ steps.)

– Finally, using the values $A(q(x,n))$ $(n \leq n_x)$, simulate the computation of M with oracle A on input x in order to get $C(x) = M^A(x)$. (This can be done in $p(|x|)$ steps.)

So $C(x)$ can be computed in

$$O(p(|x|) \cdot 2^{(3|x|)^k}) \leq O(2^{|x|^{k+1}})$$

steps. It follows that $C \in \mathrm{EXP}_{k+1}$ contrary to assumption.

This completes the proof.

For a tt-E-nontrivial set $A \in \mathrm{E}$ we can modify the above argument as follows. Given $k \geq 1$, take a set C such that $C \in \mathrm{E} \setminus \mathrm{E}_{4k}$ and $C \leq^p_{tt} A$, and let B be the set obtained from C as above. Then one can show as above that $B \leq^p_m A$ and $B \notin \mathrm{E}_k$. We cannot argue, however, that B is in E. So the above proof of Theorem 2 cannot be converted into a proof of the corresponding claim for E in place of EXP. In fact, as we will show next, Theorem 2 fails for E.

4 Separating Nontriviality for E under Truth-Table Type Reducibilities

We now separate the nontriviality notions for E under the different truth-table type reducibilities. In order to separate E-nontriviality under bounded truth-table reducibility from E-nontriviality under many-one reducibility, we give some stronger separation results by showing that E-nontriviality (or even E-completeness) under bounded truth-table reductions of norm $k+1$ does not imply E-nontriviality under bounded truth-table reductions of norm k.

Theorem 3. *(a) Let $k \geq 1$. There is a $(k+1)$-tt-complete set in* E *which is k-tt-*E*-trivial.*

(b) There is a tt-complete set in E *which is btt-*E*-trivial.*

Proof (Idea). In order to explain the basic ideas underlying the proof we consider only a special case of part (a), namely the case $k=1$, for which the proof is typical for the general argument but less technical.

By a slow diagonalization argument, we construct a set $A \in O(2^{n^2})$ such that

$$\text{A is 2-tt-hard for E} \tag{11}$$

and

$$\forall B \in \mathrm{E}\ (B \leq^p_{1-tt} A \Rightarrow B \in DTIME(2^n)). \tag{12}$$

Then any set $\hat{A} \in \mathrm{E}$ with $\hat{A} =^p_m A$ will be 2-*tt*-complete for E but 1-*tt*-E-trivial. (Note that, by padding, for any set $A \in \mathrm{EXP}$ there is a set $\hat{A} \in \mathrm{E}_1$ such that $\hat{A} =^p_m A$.)

Condition (11) is satisfied as follows. Given an E-complete set $C \in \mathrm{E}_1$, it suffices to ensure that $C \leq^p_{2-tt} A$. This is achieved by guaranteeing

$$x \in C \Leftrightarrow |A \cap \{x0, x1\}| = 1. \tag{13}$$

Our strategy for satisfying (12) is much less straightforward, and, for implementing it, we will need a speed-up argument. We start with some notation. Let $\{E_e : e \geq 0\}$ be an enumeration of E such that, for x with $|x| > e$, $E_e(x)$ can be (uniformly) computed in time $2^{e|x|}$, and let $\{(g_e, h_e) : e \geq 0\}$ be an enumeration of the p-1-*tt*-reductions - where $g_e : \{0,1\}^* \rightarrow \{0,1\}^*$ is a selector function and $h_e : \{0,1\}^* \times \{0,1\} \rightarrow \{0,1\}$ is the corresponding evaluator function - such that, for a common (uniform) polynomial time bound p_e of g_e and h_e, $p_e((|x|+1)^2) \leq 2^{|x|}$ for all x with $|x| > e$. Finally, call a string x *e-critical*, if $h_e(x,0) \neq h_e(x,1)$.

Now, in order to satisfy (12), we will ensure

$$\forall\ \alpha > 0\ (A \in \mathrm{DTIME}(2^{\alpha \cdot n^2})) \tag{14}$$

(where α is a real number), and, for $e \geq 0$ where $e = \langle e_0, e_1 \rangle$, we will meet the requirements

$$\Re_e :\ E_{e_0} \leq^p_{1-tt} A \text{ via } (g_{e_1}, h_{e_1}) \Rightarrow \forall^\infty x\ (x \text{ is } e_1\text{-critical } \Rightarrow |x| > 2^{-e} \cdot |g_{e_1}(x)|^2).$$

(Intuitively, if, for the p-1-tt-reduction (g_{e_1}, h_{e_1}), there are infinitely many x such that the query $g_{e_1}(x)$ is “long” and relevant then $\Re_e$ requires that (g_{e_1}, h_{e_1}) is not a reduction from E_{e_0} to A.)

In order to show that this will guarantee (12), let $B \in \mathrm{E}$ be given such that $B \leq^p_{1-tt} A$. Fix $e = \langle e_0, e_1 \rangle$ such that $B = E_{e_0}$ and $B \leq^p_{1-tt} A$ via (g_{e_1}, h_{e_1}). Then, by requirement $\Re_e$, we may fix n_0 such that, for all e_1-critical x with $|x| \geq n_0$, $|x| > 2^{-e}|g_{e_1}(x)|^2$ holds. So, for x with $|x| \geq n_0$,

$$B(x) = E_{e_0}(x) = h_{e_1}(x, A(g_{e_1}(x))) = h_{e_1}(x, y)$$

where

$$y = \begin{cases} A(g_{e_1}(x)) & \text{if } |x| > 2^{-e}|g_{e_1}(x)|^2 \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, by (14) (for $\alpha = 2^{-e}$), the string y can be computed in $O(2^{|x|})$ steps. Obviously this implies $B \in \mathrm{E}_1$.

We now turn to the construction of A.

At stage s of the construction, we define $A(z_s0)$ and $A(z_s1)$ for the sth string z_s w.r.t. to the length-lexicographical ordering. We say that requirement $\Re_e$ *requires attention* at stage s if $e < |z_s|$, $\Re_e$ is not satisfied at any stage $t < s$, and

$$\exists x\ (|x| \leq 2^{-e}(|z_s|+1)^2\ \&\ g_{e_1}(x) \in \{z_s0, z_s1\}\ \&\ x \text{ is } e_1\text{-critical}). \tag{15}$$

Now, if no requirement requires attention then let $A \cap \{z_s0, z_s1\} = \emptyset$ if $C(z_s) = 0$ and $A \cap \{z_s0, z_s1\} = \{z_s0\}$ if $C(z_s) = 1$. Otherwise, fix e minimal such that $\Re_e$ requires attention. Call $\Re_e$ *active* and *satisfied* at stage s and proceed as follows. Fix x minimal as in (15), fix the unique $i, j \leq 1$ such that $g_{e_1}(x) = z_si$ and $E_{e_0}(x) \neq h_{e_1}(x, j)$ (note that such a j exists since x is e_1-critical) and define A on $\{z_s0, z_s1\}$ by

$$A \cap \{z_s0, z_s1\} = \begin{cases} \emptyset & \text{if } C(z_s) = 0 \text{ and } j = 0 \\ \{z_s0, z_s1\} & \text{if } C(z_s) = 0 \text{ and } j = 1 \\ \{z_s(1-i)\} & \text{if } C(z_s) = 1 \text{ and } j = 0 \\ \{z_si\} & \text{if } C(z_s) = 1 \text{ and } j = 1. \end{cases}$$

This completes the construction.

Obviously, the construction ensures (13). Moreover, all requirements $\Re_e$ are met. In order to show this, first observe that any requirement requires attention at most finitely often. So if there are infinitely many e_1-critical strings x such that $|x| \leq 2^{-e} \cdot |g_{e_1}(x)|^2$ then $\Re_e$ will eventually become active and satisfied. But if $\Re_e$ becomes satisfied then $\Re_e$ is met since, at the stage at which $\Re_e$ is satisfied, it is ensured that E_{e_0} is not p-1-tt-reducible to A via (g_{e_1}, h_{e_1}).

It remains to show that (14) holds. Given $e \geq 1$, it suffices to show that $A \in \mathrm{DTIME}(2^{2^{-e} \cdot n^2})$.

Fix a stage s and let $n = |z_s 0| = |z_s 1|$. Note that the complexity of computing $A(z_s 0)$ and $A(z_s 1)$ according to the above construction is essentially determined by the complexity of the procedure which tells us which requirements $\Re_e$ require attention (and if some $\Re_e$ requires attention, how $\Re_e$ wants $A(z_s 0)$ and $A(z_s 1)$ to be defined in case that it becomes active). Now, as one can easily show, the latter can be done in $O(2^{2^{-(e-1)} \cdot n^2})$ steps. (Note that the crucial part of the procedure of checking whether $\Re_e$ requires attention at stage s is to check whether there is a string x of length $\leq 2^{-e} n^2$ as in (15).)

So the complexity of checking whether $\Re_e$ requires attention is decreasing in e. Since each requirement requires attention only finitely often, this observation helps us to speed up the construction of A in order to get the required time bound for A. Namely, by using a finite table containing the information on the effect of the requirements $\Re_{e'}$, $e' \leq e + 1$, on the construction of A, it suffices to check which of the requirements $\Re_{e''}$ with $e'' > e + 1$ require attention. It easily follows from the above observation on the complexity of checking whether a requirement requires attention, that the thus modified construction witnesses that $A \in \mathrm{DTIME}(2^{2^{-e} \cdot n^2})$.

This completes the proof.

We conclude our analysis of E-nontriviality under the weak reducibilities with the observation that 1-*tt*-nontriviality for E coincides with *m*-nontriviality for E. The corresponding observations for E-completeness and weak E-completeness have been made by Homer et al. [7] and Ambos-Spies et al. [3], respectively. We omit the easy proof.

Lemma 1. *Let $A \in$ E be 1-tt-nontrivial for* E. *Then A is m-nontrivial for* E.

5 Summary of Results

Our results on the relations among completeness and nontriviality under the common polynomial-time reducibilities can be summarized as follows.

Theorem 4. *For $A \in$ E the following and (up to transitive closure) only the following implications hold in general:*

$$\begin{tikzcd}
A\ m\text{-E-}\mathit{complete} \arrow[d, Leftrightarrow] \arrow[r, Rightarrow] & A\ m\text{-E-}\mathit{nontrivial} \arrow[d, Leftrightarrow] \\
A\ 1\text{-}\mathit{tt}\text{-E-}\mathit{complete} \arrow[d, Rightarrow] & A\ 1\text{-}\mathit{tt}\text{-E-}\mathit{nontrivial} \arrow[d, Rightarrow] \\
A\ \mathit{btt}\text{-E-}\mathit{complete} \arrow[d, Rightarrow] \arrow[r, Rightarrow] & A\ \mathit{btt}\text{-E-}\mathit{nontrivial} \arrow[d, Rightarrow] \\
A\ \mathit{tt}\text{-E-}\mathit{complete} \arrow[d, Rightarrow] \arrow[r, Rightarrow] & A\ \mathit{tt}\text{-E-}\mathit{nontrivial} \arrow[d, Rightarrow] \\
A\ T\text{-E-}\mathit{complete} \arrow[r, Rightarrow] & A\ T\text{-E-}\mathit{nontrivial}
\end{tikzcd}$$

Theorem 5. *For $A \in \mathrm{EXP}$ the following and (up to transitive closure) only the following implications hold in general:*

$$\begin{tikzcd}
A\ m\text{-EXP-}\textit{complete} \arrow[d, Leftrightarrow] & A\ m\text{-EXP-}\textit{nontrivial} \arrow[d, Leftrightarrow] \\
A\ 1\text{-}tt\text{-EXP-}\textit{complete} \arrow[d, Rightarrow] & A\ 1\text{-}tt\text{-EXP-}\textit{nontrivial} \arrow[d, Leftrightarrow] \\
A\ btt\text{-EXP-}\textit{complete} \arrow[d, Rightarrow] & A\ btt\text{-EXP-}\textit{nontrivial} \arrow[d, Leftrightarrow] \\
A\ tt\text{-EXP-}\textit{complete} \arrow[d, Rightarrow] \arrow[r, Rightarrow] & A\ tt\text{-EXP-}\textit{nontrivial} \arrow[d, Rightarrow] \\
A\ T\text{-EXP-}\textit{complete} \arrow[r, Rightarrow] & T\text{-EXP-}\textit{nontrivial}
\end{tikzcd}$$

Here we have not looked at E- or EXP-nontriviality under the strong reducibilities. Berman [6] has shown that E-completeness under many-one reducibility coincides with E-completeness under length-increasing one-one reducibility and the corresponding fact for weak E- (and EXP-) completeness has been shown in [3]. It can be easily shown that E- (and EXP-) nontrivality under many-one reducibility coincides with E- (and EXP-) nontrivality under length-increasing many-one reducibility. The question whether nontriviality under many-one reducibility and nontriviality under one-one reducibility coincide, however, is open. We have obtained such a collapse only under the strong hypothesis that P = PSPACE.

References

1. Ambos-Spies, K., Bakibayev, T.: Weak completeness notions for exponential time (to appear)
2. Ambos-Spies, K., Bakibayev, T.: Comparing nontriviality for E and EXP (to appear)
3. Ambos-Spies, K., Mayordomo, E., Zheng, X.: A Comparison of Weak Completeness Notions. In: Proceedings of the 11th Annual IEEE Conference on Computational Complexity, pp. 171–178 (1996)
4. Balcázar, J.L., Díaz, J., Gabarró, J.: Structural complexity I, 2nd edn. Springer, Berlin (1995)
5. Balcázar, J.L., Díaz, J., Gabarró, J.: Structural complexity II. Springer, Berlin (1990)
6. Berman, L.: On the structure of complete sets: almost everywhere complexity and infinitely often speedup. In: Proceedings of the 17th Annual Symposium on Foundations of Computer Science, pp. 76–80 (1976)
7. Homer, S., Kurtz, S., Royer, J.: On 1-truth-table-hard languages. Theoret. Comput. Sci. 115, 383–389 (1993)
8. Juedes, D.W., Lutz, J.H.: Weak completeness in E and E_2. Theoret. Comput. Sci. 143, 149–158 (1995)
9. Lutz, J.H.: Weakly hard problems. SIAM J. Comput. 24, 1170–1189 (1995)
10. Watanabe, O.: A comparison of polynomial time completeness notions. Theoret. Comput. Sci. 54, 249–265 (1987)

Streaming Algorithms for Some Problems in Log-Space

Ajesh Babu, Nutan Limaye, and Girish Varma

Tata Institute of Fundamental Research, Mumbai, India
{ajesh,nutan,girish}@tcs.tifr.res.in

Abstract. In this paper, we give streaming algorithms for some problems which are known to be in deterministic log-space, when the number of passes made on the input is unbounded. If the input data is massive, the conventional deterministic log-space algorithms may not run efficiently. We study the complexity of the problems when the number of passes is bounded.

The first problem we consider is the membership testing problem for deterministic linear languages, DLIN. Extending the recent work of Magniez et al.[11](to appear in STOC 2010), we study the use of fingerprinting technique for this problem. We give the following streaming algorithms for the membership testing of DLINs: a randomized one pass algorithm that uses $O(\log n)$ space (one-sided error, inverse polynomial error probability), and also a p-pass $O(n/p)$-space deterministic algorithm. We also prove that there exists a language in DLIN, for which any p-pass deterministic algorithm for membership testing, requires $\Omega(n/p)$ space. We also study the application of fingerprinting technique to visibly pushdown languages, VPLs.

The other problem we consider is, given a degree sequence and a graph, checking whether the graph has the given degree sequence, Deg-Seq. We prove that, any p-pass deterministic algorithm that takes as its input a degree sequence, followed by an adjacency list of a graph, requires $\Omega(n/p)$ space to decide Deg-Seq. However, using randomness, for a more general input format: degree sequence, followed by a list of edges in any arbitrary order, Deg-Seq can be decided in $O(\log n)$ space. We also give a p-pass, $O(n/p)$-space deterministic algorithm for Deg-Seq.

1 Introduction

Conventional computational models such as Turing machines do not restrict the number of passes on the input. We wish to understand the conventional space bounded models of computation when the number of passes made on the input is restricted. The model of computation where the number of passes is bounded has been studied extensively [1,14]. In this paper we study problems that are already known to be in deterministic log-space with no restrictions on the number of passes and re-analyze their complexity for a bounded number of passes.

© Springer-Verlag Berlin Heidelberg 2010

The paper is divided into two parts. In the first part, we consider the membership problem for subclasses of context-free languages, CFLs. The membership problem, for a fixed language L is: given a string w checking whether w belongs to L. We consider some subclasses of CFLs for which log-space algorithms are already known. We study the number of passes versus space and randomness trade-offs for these problems.

Recently, Magniez et al. [11] studied the membership problem for Dyck_2 (the set of balanced strings over two types of parentheses). They proved that there is a $O(\sqrt{n}\log n)$ space, one pass randomized algorithm for Dyck_2. They also proved that any randomized algorithm that makes one pass on the input must use $\Omega(\sqrt{n \log n})$ space.

Using their ideas of finger-printing the stack, we study the problem of membership testing of deterministic linear languages, DLIN, the class of languages generated by deterministic linear grammars for which the right hand side of every rule has at most one non-terminal. The language Dyck_2 does not belong to DLIN. The membership testing for languages in DLIN is in deterministic log-space when there is no restriction on the number of passes [8]. The most obvious log-space algorithm makes multiple passes on the input. We ask whether adding randomness leads to an algorithm with fewer passes. We prove the following theorem:

Theorem 1. *For any $L \in \mathsf{DLIN}$ there exists a constant c and a randomized one pass algorithm A_L that uses $O(\log n)$ space such that $\forall w \in \Sigma^*$ if $w \in L$, $\Pr[A_L(w) = 1] = 1$ and if $w \notin L$ $\Pr[A_L(w) = 1] \leq \frac{1}{n^c}$.*

We also analyze the deterministic streaming complexity of the membership problem of DLIN. We prove the following two theorems:

Theorem 2. *There is a deterministic p-passes, $O(n/p)$-space algorithm for membership testing for languages in* DLIN.

Theorem 3. *Any deterministic algorithm that makes p-passes over the input, will require $\Omega(n/p)$ space for membership testing for languages in* DLIN.

We analyze the algorithm used to prove Theorem 1, and apply it to a subclass of visibly pushdown languages, VPLs. VPLs were defined by Mehlhorn [12] and Alur et al. [2]. Their membership testing was studied in [12,5,6]. Brahnmuhl et al. gave the first log-space algorithm for membership testing of VPLs and later Dymond settled its complexity to NC^1. VPLs are defined over tri-partitioned alphabet $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_l$. A grammar form for VPLs was given in [3]. A sub-class of VPLs, well-matched VPLs was defined in [2]. It is believed that an efficient algorithm for membership testing of wVPLs can help in XML type checking. For large XML documents, it may be inefficient to store the whole document to perform the type checking. It is therefore interesting to consider the model where the membership testing for languages in wVPL can be done using fewer passes on the input.

We answer the question in a more restricted setting. We consider a class of languages generated by grammars more restrictive than the grammar form for wVPLs. We denote such grammars by rest-VPG.

Let w be an input string over the input alphabet of L. And let $n = |w|$. An index $i \in [n]$ is said to be a *reversal* if $w[i-1] \in \Sigma_r$ and $w[i] \in \Sigma_c \cup \Sigma_l$.

Let $rev(L, n)$ be defined as maximum value of $rev(w)$ over all strings $w \in \Sigma^n$ such that $w \in L$. We denote this by $rev(n)$ if L is clear from the context.

Theorem 4. *For any L generated by rest-VPG, there exists a constant c and a randomized one pass algorithm A_L that uses $O(rev(L,n)\log n)$ space such that $\forall w \in \Sigma^*$ if $w \in L$, $\Pr[A_L(w) = 1] = 1$ and if $w \notin L$ $\Pr[A_L(w) = 1] \leq \frac{1}{n^c}$.*

In the second part of the paper we consider the following graph problem:

Degree-Sequence, Deg-Seq:
Given a degree sequence and a directed graph; check if vertices $v_1, v_2, \ldots, v_n$ have out-degrees $d_1, d_2, \ldots, d_n$, respectively.

This problem is known to be in log-space (in fact in TC^0(see for example [15])). The obvious log-space algorithm for this problem makes multiple passes on the input, as for the membership testing of DLIN.

It has been observed [4,7] that the complexity of graph problems changes drastically depending on the order in which the input is presented to the streaming algorithm. If the input to Deg-Seq is such that the degree of a vertex along with all the edges out of that vertex are listed one after the other, then checking whether the graph has the given degree sequence is trivial. If the degrees sequence is listed first, followed by the adjacency list of the graph then we observe that a one pass deterministic algorithm needs $\Omega(n)$ space to compute Deg-Seq. For a more general ordering of the input where the degree sequence is followed by a list of edges in an arbitrary order, we prove the following theorem:

Theorem 5. *If the input is a degree sequence followed by a list of edges in an arbitrary order, then* Deg-Seq *can be solved: (1) by a one pass, $O(\log n)$ space randomized streaming algorithm such that if vertices $v_1, v_2, \ldots, v_n$ have out-degrees $d_1, d_2, \ldots, d_n$, respectively, then the algorithm accepts with probability 1 and rejects with probability $1-n^{-c}$, otherwise. (2) by a p-passes, $O((n \log n)/p)$-space deterministic streaming algorithm.*

The rest of the paper is organized as follows: In Section 2 we prove Theorem 1 and Theorem 2 . Here we also give a randomness efficient version of the algorithm used to prove Theorem 1. In Section 3 we use the algorithm from Section 2.1 and prove Theorem 4. In Section 4, we analyze the complexity of Deg-Seq. and prove Theorem 5.

2 Streaming Algorithms for Membership Testing of DLINs

In this section we give streaming algorithms for the membership testing of languages in DLIN (Theorem 1). (See [9], for the basic definitions regarding context-free grammars, sentential forms and derivations.) We start with some definitions.

Definition 1 ([10,8]). *A* ***deterministic linear CFG****,* DL-CFG*, is a CFG (Σ, N, P, S) for which, every production is of the form $A \longrightarrow a\omega$, where $a \in \Sigma$*

and $\omega \in (N \cup \epsilon)\Sigma^*$ and for any two productions, $A \longrightarrow a\omega$ and $B \longrightarrow b\omega'$, if $A = B$ then $a \neq b$, where $a, b \in \Sigma$ and $\omega, \omega' \in (N \cup \epsilon)\Sigma^*$.

Definition 2. ***Deterministic linear CFLs,*** *DLIN, is the class of languages for which there exists a $\mathsf{DL\text{-}CFG}$ generating it.*

The algorithm for membership testing of DLINs is a modification of the algorithm of [11].

2.1 A Randomized Streaming Algorithm

We observe a property of 1-turn-Dyck_2, a language accepted by the following grammar: $S \longrightarrow (S_1 \mid [S_2,\ S_1 \longrightarrow (S_1) \mid [S_1] \mid),\ S_2 \longrightarrow (S_2) \mid [S_2] \mid]$, which can be generalized for DLIN to obtain an efficient algorithm.

Observation 6. *For any string w in 1-turn-Dyck_2 and $i \in [\frac{n}{2}]$, the letter at location $w[i]$ completely determines the letter at location $w[n-i+1]$, where $n = |w|$.*

For a language in DLIN the observation is not immediately applicable. For example, $S \to aBc;\ B \to aSb$ is a valid $\mathsf{DL\text{-}CFG}$ but on letter a at location $i < \frac{n}{2}$, we do not know whether to expect b or c at $w[n-i+1]$. However, something similar to Observation 6 applies to $\mathsf{DL\text{-}CFG}$.

In this section, for the sake of simplicity, we restrict ourselves to $\mathsf{DL\text{-}CFG}$ grammars that have rules of the form $A \to uBv$, where $|u| = |v| = 1$. It is easy to see that this can be generalized to all of $\mathsf{DL\text{-}CFG}$. Let L be a DLIN. Any string in L is produced by repeated application of rules corresponding to the $\mathsf{DL\text{-}CFG}$ of L, say G_L. The sentential forms arising in the derivation of any $w \in L$ have at most one non-terminal in them. Let the current sentential form be uAu', where $u, u' \in \Sigma^*$, u and u' are prefix and suffix of w respectively, and $A \in N$. Let the next terminal symbol after u in w be a. Suppose there is a rule $A \to aBc$ then determinism forces that there is no other rule $A \to a'B'c'$ such that $a' = a$. Therefore, if the rule for A is to be applied and the letter to be generated is a, then the next sentential form is $uaBcu'$, i.e. A and a uniquely determine c.

Observation 7. *Let w be a string generated by a $\mathsf{DL\text{-}CFG}$ G that have rules only of the form $A \to uBv$, where $|u| = |v| = 1$. Let $i \in [\frac{n}{2}]$. The letter at location $w[i]$ and the rule that needs to be applied to produce it, completely determine the letter at location $w[n-i+1]$, where $n = |w|$.*

While processing $w[i]$, we add a monomial, a power of a variable, to the sum which is *expected* to be subtracted on reading $w[n-i+1]$.

In the case of 1-turn-Dyck_2 we could write down an explicit polynomial to be computed for a given input string. Here, the polynomial computation is more involved.

Let L be a DLIN, generated by a $\mathsf{DL\text{-}CFG}$, $G_L = (N, \Sigma, P, S)$. We describe the multi-variate polynomial that we come up with for the given input string such that this polynomial is the zero polynomial if and only if the given string is in L.

Let $\Sigma = \{a_1, a_2, \ldots, a_k\}$. Let $\{x_1, x_2, \ldots, x_k\}$ be formal variables. Let *type* and *next* be two functions such that $type : \Sigma \times N \to \{x_1, \ldots, x_k\}$ and $next : \Sigma \times N \to N$. If $A \to a_i B a_j$ is a rule in the grammar G_L, then $type(a_i, A)$ and $next(a_i, A)$ are defined to be x_j and B respectively. They are undefined otherwise. The determinism of the grammar ensures that for given A and a_i, x_j and B are unique.

We define the polynomial inductively using an extra variable *var* also maintained inductively.

Let $q_0(x_1, \ldots, x_k) = 0$ and $var_0 = S$. For $i \leq \frac{n}{2}$, we define:

$$\begin{aligned} q_i(x_1, \ldots, x_k) = &\quad q_{i-1}(x_1, \ldots, x_k) + (type(w[i], var_{i-1}))^i \\ var_i \quad = &\quad next(w[i], var_{i-1}). \end{aligned}$$

For $i > \frac{n}{2}$, define $q_i(x_1, \ldots, x_k)$ as $q_{i-1}(x_1, \ldots, x_k) - (map(w[i]))^{n-i+1}$, where $map(a_i) = x_i$.

It is easy to see that $q_n(x_1, \ldots, x_k)$ is the zero polynomial if and only if the given string is in L.

As in the case of 1-turn-Dyck$_2$, we will implicitly compute this polynomial. The idea is to maintain an evaluation of this polynomial at randomly points $\alpha_1 \cdots \alpha_k$ chosen from a enough field, $\mathbb{F}_p$. We are now ready to describe our algorithm for membership test of DLIN.

Algorithm 1. Randomized one pass algorithm

```
Pick $\alpha_1, \alpha_2, \ldots, \alpha_k$ uniformly at random from $\mathbb{F}_p$, where $k = |\Sigma|$.
$Sum \leftarrow 0$
$var \leftarrow S$
**for** $i = 1$ to $n/2$ **do**
  Let *index* be $j$ if $type(w[i], var) = x_j$
  $Sum \leftarrow (Sum + (\alpha_{index})^i)(mod\ p)$
  $var \leftarrow next(w[i], var)$
**end for**
**for** $i = n/2 + 1$ to $n$ **do**
  Let *index* be $j$ if $map(w[i]) = x_j$
  $Sum \leftarrow (Sum - (\alpha_{index})^{n-i+1})(mod\ p)$
**end for**
Output 'yes' if $Sum$ is zero and 'no' otherwise.
```

It is easy to see that the above algorithm can be generalized when the DL-CFG grammar has rules of the form $A \to aBv$, where $|v| = 0$ or $|v| > 1$. We need to maintain an extra variable to keep track of the power to which α_js will be raised. We denote this variable by h. Suppose the rule $A \to aBv$ such that $|v| > 1$ is being applied at step $i \leq n/2$, then obtain the type of each letter inside v. Say the types are $x_{t_1}, x_{t_2}, \ldots x_{t_l}$ where $l = |v|$. Now add $\sum_{j=1}^{l} \alpha_{t_j}^{h+(l-j)}$ to the sum and set h to $h + l$. For $|v| = 0$, h and Sum remain unchanged.

The algorithm makes one pass on the input. Also, $q_n(x_1, \ldots, x_k)$ is non-zero when the given string is not in L. However, the evaluation may still be zero.

Note that degree of $q_n(x_1, x_2, \ldots, x_k)$ is less than or equal to n. If the field size is chosen to be $n^{1+c} \leq p \leq n^{2+c}$, then due to Schwartz-Zippel lemma [13] the probability that Sum is zero but $q_n(x_1, x_2, \ldots, x_k)$ is non-zero is at most n/p which is at most n^{-c}. The amount of randomness used will be $(c|\Sigma| \log n)$. The algorithm keeps track of $\alpha_1, \ldots, \alpha_k$ and the value of Sum. Therefore, the space used is $(c|\Sigma| \log n)$.

2.2 Randomness Efficient Version

In [11], it was observed that the membership testing of Dyck_k $O(\log k)$-streaming reduces to membership testing of Dyck_2. We show that the membership testing for any language in DLIN $O(1)$-streaming reduces to membership testing in 1-turn-Dyck_k, where k is the alphabet size of the language.

As the membership testing for Dyck_2 requires only $(c \log n)$ random bits as opposed to $(ck \log n)$-bits used in Algorithm 1, this gives a randomness efficient algorithm. The main result in this section is stated below:

Theorem 8. *The membership testing for any language in* DLIN $O(\log |\Sigma|)$-*streaming reduces to membership testing in* 1-*turn*-Dyck_2, *where* Σ *is the alphabet of the language.*

2.3 A Deterministic Multi-pass Algorithm

In this section we give a deterministic multi-pass algorithm for the membership testing of any language in DLIN (Theorem 2).

As any language in DLIN $O(\log |\Sigma|)$-streaming reduces to 1-turn-Dyck_2(Theorem 8), it suffices to give a p-passes, $O(n/p)$-space deterministic algorithm for membership testing of 1-turn-Dyck_2.

The algorithm divides the string into blocks of length $n/2p$. Let the blocks be called B_0, B_1, ..., B_{2p-1} from left to right. (i.e. $B_i = w[i(n/2p)+1]\ w[i(n/2p)+2]\ \ldots w[(i+1)n/2p]$.) The algorithm considers a pair of blocks $(B_j,B_{2p-(j+1)})$ during the jth pass. Using the stack explicitly, the algorithm checks whether the string formed by the concatenation of B_j and $B_{2p-(j+1)}$ is balanced. If it is balanced, it proceeds to the next pair of blocks. The number of passes required is p. Each pass uses $O(n/p)$ space and the algorithm is deterministic. The above algorithm is optimal.

3 Streaming Algorithm for a Subclass of VPLs

In this section we prove Theorem 4. Visibly pushdown languages, VPLs, were defined by [12] and [2]. They are known to be a subclass of DCFLs. In [3], a grammar form for VPLs was defined. We denote the grammars generating VPLs by VPG. Here, we consider a restriction of VPG.

Consider a context-free grammar $G = (N, \Sigma, S, P)$ over a tri-partitioned alphabet $\Sigma = (\Sigma_c, \Sigma_r, \Sigma_l)$ having rules of the form $A \longrightarrow \epsilon$, $A \longrightarrow cB$, $A \longrightarrow aBb$

or $A \longrightarrow aBbD$, where $a \in \Sigma_c$ (called the call alphabet), $b \in \Sigma_r$ (called the return alphabet), $c \in \Sigma_l$ (called the local alphabet), and $D \longrightarrow \epsilon \notin P$ (i.e. if $A \rightarrow \epsilon$, then there is no rule in P that has A as its second non-terminal.) If $A \longrightarrow a\omega$ and $A \longrightarrow a'\omega'$ are two rules such that $\omega, \omega' \in (N \cup \Sigma)^*$, then $a \neq a'$. We denote such grammars by rest-VPG and the languages generated by it as rest-VPLs.

The example language generated by the rest-VPG, G: $S \longrightarrow aAbB$, $A \longrightarrow aAb \mid \epsilon, B \longrightarrow aAb$, is $\{a^n b^n a^m b^m | n \geq 1, m \geq 1\}$.

We coin a notation to address these rules of rest-VPG. Let a rule be called *linear* if it has one non-terminal on the right hand side. Let a rule be called *quadratic* if there are two non-terminals on the right hand side.

We first make one crucial observation about the derivation of a string in a language generated by rest-VPG.

Observation 9. *Let L be generated by a rest-*VPG*, say G_L. Let $w \in \Sigma^n$. If $w \in L$, then for any derivation d of w in G_L, the number of times a quadratic rule is applied is at most $rev(L, n)$.*

Proof. Recall that an index $i \in [n]$ is called a reversal if $w[i-1] \in \Sigma_r$ and $w[i] \in \Sigma_c \cup \Sigma_l$. As $w \in L$, and L is generated by G_L. Therefore, there is a derivation for w in G_L. Suppose there exists a derivation for w in G_L that needs more than $rev(L, n)$ applications of quadratic rules. Every time a quadratic rule is applied, it gives rise to one reversal. Therefore, the string w must have more than $rev(L, n)$ reversals, which is a contradiction.

Let L be a fixed rest-VPL and let L be generated by a rest-VPG G_L. Let w be the input string. For every application of a quadratic rule in the derivation of w, we perform a book keeping operation storing $O(\log n)$ bits in the memory. As the maximum number of times a quadratic rule is applied in the derivation of w is bounded by $rev(w)$ (due to Observation 9), we get the desired bound. (If the storage grows beyond $O(rev(n) \log n)$ the algorithm rejects and halts.)

We now describe the book keeping. Initially the expected sum is zero. As long as linear rules are applied, expected sums can be computed as in Algorithm 1. Consider the first time a quadratic rule is applied. The string w can be split into four parts at this stage. The first part consists of the string of length l that has been read so far, and the fourth part consists of the suffix of w of length l. The second (third) part consists of the substring of w generated by the first (second, respectively) non-terminal.

The expected sum accumulated during the first part needs to be used when the fourth part is being processed. The second non-terminal is needed when the third part is being processed. Thus, while starting to check the second part, the sum as well as the second non-terminal are stacked. Once, the second part is recursively checked, the stacked non-terminal is popped and it is recursively processed. After this, the algorithm needs to check the fourth part. During this stage, the sum that was stacked is popped and used as in Algorithm 1.

We start with some notation. Let *type* and $next_1, next_2$ be functions such that $type : \Sigma \times N \rightarrow \{x_1, \ldots, x_k\}$ $next_1 : \Sigma \times N \rightarrow N$ and $next_2 : \Sigma \times N \rightarrow N$.

If $A \to a_i B a_j D$ is a rule in G_L, then the values of $type(a_i, A)$, $next_1(a_i, A)$, and $next_2(a_i, A)$ are x_j, B and D, respectively. (Here $a_i \in \Sigma_c$ and $a_j \in \Sigma_r$. Also, if $D = \epsilon$ then $next_2(a_i, A)$ is ϵ).

If $A \to a_i B$ is a rule in the grammar G_L, then the values of $next_1(a_i, A)$, $next_2(a_i, A)$, are B and ϵ, respectively. (Note that in this case, $a_i \in \Sigma_l$.) They are undefined otherwise. The algorithm is given below.

4 Streaming Algorithms for Checking Degree Sequence of Graphs

The input to a graph streaming algorithm may be in the form of a sequence of edges. These may be provided in a specific order (eg: adjacency list), or in any arbitrary fashion. It has been observed [4,7] that the complexity of graph problems changes drastically depending on the order in which the edges are presented to the streaming algorithm. The usual setting is: the edges are assumed to be presented in any arbitrary order. If one is able to provide an efficient algorithm in such a setting, then of course this gives the most general algorithm. However, more recently Das Sarma et al. [4] observed that it is also useful to consider other orderings of the edges. They observed that, the algorithm can be considered as a resource bounded verifier and that the input is presented to the verifier by a prover. In this setting, two models can be considered: the adversarial model and the best order model [4]. In the former the prover may provide the edges in the worst possible order. In contrast to this, in the latter model, the prover orders the edges in such a manner that the verifier can easily check the property.

We consider the problem Deg-Seq which we defined in Section 1. Under various assumptions on its input, we analyze its complexity. If the input to Deg-Seq is such that the degree of a vertex along with all the edges out of that vertex are listed one after the other, then checking whether the graph has the given degree sequence is trivial. If the degrees sequence is listed first, followed by the adjacency list of the graph then we observe the following:

Lemma 1. *A one pass deterministic algorithm for* Deg-Seq *needs $\Omega(n)$ space when the input is: a degrees sequence, followed by the adjacency list.*

Here we present the proof of the first part of Theorem 5.

Proof (of part 1 of Theorem 5). We come up with a uni-variate polynomial from the given degree sequence and the set of edges such that the polynomial is identically zero if and only if the graph has the given degree sequence (assuming some predecided order on the vertices).

We do not store the polynomial explicitly. Instead, we evaluate this polynomial at a random point chosen from a large enough field and only maintain the evaluation of the polynomial. The Schwartz-Zippel lemma [13] gives us that with high probability the evaluation will be non-zero if the polynomial is non-zero. (If the polynomial is identically zero, its evaluation will also be zero.)

Algorithm 2. Streaming algorithm for membership testing of rest-VPLs

```
Pick $\alpha_1, \alpha_2, \ldots, \alpha_k$ uniformly at random from $\mathbb{F}_p$, where $k = |\Sigma|$.
$Sum \leftarrow 0$, $var \leftarrow S$, $h \leftarrow 0$, $Rev \leftarrow rev(n)$ (encoded along with the input)
for $i = 1$ to $n$ do
   if $w[i] \in (\Sigma_c \cup \Sigma_l)$ and $next_1(w[i], var)$ is undefined then
      reject and halt
   end if
   if $w[i] \in \Sigma_c$ then
      $v_2 \leftarrow next_2(w[i], var)$
      if $v_2 \neq \epsilon$ then
         if $Rev = 0$ then
            'reject' and halt
         else
            Stack.Push($v_2, Sum, h$)   (*Addressed as: StackTop.$Var$, StackTop.$Sum$,
            StackTop.$Height$*)
            $Sum = 0$
            $Rev \leftarrow Rev - 1$
         end if
      end if
      $h \leftarrow h + 1$
      Let $index$ be $j$ if $type(w[i], var) = x_j$
      $Sum \leftarrow Sum + \alpha^h_{index} (mod\ p)$ , $var \leftarrow next_1(w[i], var)$,
   else if $w[i] \in \Sigma_r$ then
      if $w[i-1] \in \Sigma_c$ then
         $var \leftarrow \epsilon$   (*This handles rules of the form $A \longrightarrow \epsilon$*)
      end if
      Let $index$ be $j$ if $w[i] = a_j$
      $Sum \leftarrow Sum - \alpha^h_{index} (mod\ p)$
      $h \leftarrow h - 1$
      if $h =$ StackTop.$Height$ then
         if $Sum \neq 0$ then
            reject and halt
         else if StackTop.$Var = \epsilon$ then
            $Sum \leftarrow$ StackTop.$Sum$
            Stack.Pop()
         else
            $var \leftarrow$ StackTop.$Var$
            StackTop.$Var \leftarrow \epsilon$
         end if
      end if
   else
      $var \leftarrow next_1(w[i], var)$ (*i.e. $w[i] \in \Sigma_l$*)
   end if
end for
```

The uni-variate polynomial can be constructed as follows: Let the vertices be labelled from 1 to n (in the order in which the degrees appear in the degree sequence). Let $l : V(G) \to [n]$ be a function such that $l(v)$ gives the label for $v \in V(G)$.

The uni variate polynomial we construct is:

$$q(x) = \sum_i d_i x^i - \sum_{\exists v:(u,v) \in input} x^{l(u)}$$

The algorithm can be now described as:

Algorithm 3. Randomized streaming algorithm for Deg-Seq

```
Pick $\alpha \in_R \mathbb{F}_p$. ($p$ will be fixed later)
$Sum \leftarrow 0$.
for $i = 1$ to $n$ do
    $Sum \leftarrow Sum + d_i \alpha^i$
end for
for $i = 1$ to $m$ (where $m$ number of edges) do
    $Sum \leftarrow Sum - \alpha^{l(u)}$, where $e_i = (u, v)$
end for
if $Sum = 0$ then
    output "yes"
else
    output "no"
end if
```

It is easy to note that the algorithm requires only log-space as long as p is $O(poly(n))$. The input is being read only once from left to right. For the correctness, note that if the given degree sequence corresponds to that of the given graph, then $q(x)$ is identically zero and the value of Sum is also zero for any randomly picked α. We know that $q(x)$ is non-zero when the given degree sequence does not correspond to that of the given graph. However, the evaluation may still be zero. Note that degree of $q(x)$ is n. If the field size is chosen to be $n^{1+c} \leq p \leq n^{2+c}$ then due to Schwartz-Zippel lemma [13] the probability that Sum is zero given that $q(x)$ is non-zero is at most n/p which is at most n^{-c}.

Acknowledgements. We thank Meena Mahajan and Jaikumar Radhakrishnan for useful discussions.

References

1. Alon, N., Matias, Y., Szegedy, M.: The space complexity of approximating the frequency moments. In: STOC 1996: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pp. 20–29 (1996)
2. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC, pp. 202–211 (2004)

3. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM 56(3) (2009)
4. Nanongkai, D., Sarma, A.D., Lipton, R.J.: Best-order streaming model. In: Chen, J., Cooper, S.B. (eds.) TAMC 2009. LNCS, vol. 5532, pp. 178–191. Springer, Heidelberg (2009)
5. Von Braunmühl, B., Verbeek, R.: Input-driven languages are recognized in $\log n$ space. In: Karpinski, M. (ed.) FCT 1983. LNCS, vol. 158, pp. 40–51. Springer, Heidelberg (1983)
6. Dymond, P.W.: Input-driven languages are in $\log n$ depth. Information Processing Letters 26, 247–250 (1988)
7. Feigenbaum, J., Kannan, S., Strauss, M., Viswanathan, M.: Testing and spot-checking of data streams. Algorithmica 34(1), 67–80 (2002)
8. Holzer, M., Lange, K.-J.: On the complexities of linear LL(1) and LR(1) grammars. In: Ésik, Z. (ed.) FCT 1993. LNCS, vol. 710, pp. 299–308. Springer, Heidelberg (1993)
9. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Addison-Wesley Longman Publishing Co., Inc., Boston (2006)
10. Ibarra, O.H., Jiang, T., Ravikumar, B.: Some subclasses of context-free languages in NC^1. Information Processing Letters 29, 111–117 (1988)
11. Magniez, F., Mathieu, C., Nayak, A.: Recognizing well-parenthesized expressions in the streaming model. In: STOC (2010)
12. Mehlhorn, K.: Pebbling mountain ranges and its application to DCFL recognition. In: de Bakker, J.W., van Leeuwen, J. (eds.) ICALP 1980. LNCS, vol. 85, pp. 422–432. Springer, Heidelberg (1980)
13. Motwani, R., Raghavan, P.: Randomized algorithms. ACM Comput. Surv. 28(1), 33–37 (1996)
14. Muthukrishnan, S.: Data streams: algorithms and applications. In: SODA 2003: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms, p. 413 (2003)
15. Vollmer, H.: Introduction to Circuit Complexity: A Uniform Approach. Springer, New York (1999)

Temperature Aware Online Scheduling with a Low Cooling Factor

Martin Birks and Stanley P.Y. Fung[*]

Department of Computer Science, University of Leicester,
Leicester LE1 7RH, United Kingdom
{mb259,pyfung}@mcs.le.ac.uk

Abstract. We consider the problem of scheduling jobs in a processor with temperature constraints. In our model, unit jobs arrive over time, each with a deadline and a heat contribution. The objective is to maximize the total number of completed jobs while keeping the system within a given temperature threshold. Our main result is the analysis of a large class of 'reasonable' algorithms. We analyse the competitive ratio of these algorithms as a function of the cooling factor of the system, and show that they are optimal by giving a matching lower bound. We also give some results concerning the multiple machines case.

1 Introduction

Background. Advances in microprocessor technology have given a huge increase in processing power in computing devices. Moreover, the need for mobile and embedded devices mean that these computing units are packed inside an even smaller space. These together give significant problems to thermal management in microprocessors. High temperatures are a problem for many reasons such as affecting reliability and incurring higher cooling costs; see e.g. [3]. Many devices have a *temperature threshold* that cannot be exceeded without causing problems or even permanent failure. A lot of research work has been done to address these issues. While the problem can be tackled on a micro-architecture level, it has also become apparent that algorithmic techniques can be used to manage temperature and energy; see e.g. the survey in [6].

The temperature created by a processor is related to its power use as well as its cooling mechanism. Power usage is a convex function of processor speed (see e.g. [6]), so one way of controlling the temperature is to slow down the processor when the temperature gets too high; this is known as *dynamic voltage scaling* (DVS). Algorithms using DVS have been designed and analysed in [1]. Where these algorithms are competitive in minimising the maximum temperature, it is more useful in some cases to give a fixed temperature threshold that cannot be exceeded, and then maximise throughput subject to this threshold. This is the model we consider in this paper. As for cooling, we follow the model in [1]

[*] Part of this work was done while the author was on a study leave granted by the University of Leicester.

© Springer-Verlag Berlin Heidelberg 2010

and Fourier's Law, which states that the rate of cooling is proportional to the difference between the temperature of the processor and its environment, and that the environment's temperature is assumed to be constant.

Our model. We consider a simplified model with unit-length jobs, where each job j has a release time r_j, a deadline d_j and a heat contribution h_j. The jobs arrive *online* which means that the algorithm will not know anything about the job until it is released and so cannot use information regarding future jobs to make any scheduling decisions. Time is divided into unit-length time steps, with an algorithm choosing to schedule at most one job at each time step. If the temperature of the system at a time t is τ_t, then the temperature at time $t+1$ is given by $\tau_{t+1} = (\tau_t + h_j)/R$, where j is the job scheduled at time t and R is the *cooling factor*, which is the factor that the temperature of a system reduces by at the end of each time step. The temperature τ_t at any time t must not exceed the thermal threshold T. Without loss of generality we can assume that the initial temperature is 0 and that T is 1. The objective is to compute a schedule that maximises the total number of completed jobs, while staying under the thermal threshold. Using standard notations this can be described as $1|\text{online-}r_i, h_i, p_i = 1|\sum U_i$ (where $U_i = 1$ if job i is completed on time and 0 otherwise). We measure the performance of online algorithms using competitive analysis. An algorithm is c-competitive if the value obtained by the online algorithm is at least $1/c$ of that obtained by an offline optimal algorithm, for any input instance.

Previous results. Although there is an abundance of literature at the more practical or hardware level (see e.g. [3,8] and the references therein), there are comparatively few results of analysing the problem in the framework of online competitive algorithms. Bansal et al. [1] considered the problem of minimizing the maximum temperature and gave online competitive algorithms. For the unit-length throughput model that we consider in this paper, the main previous result was from [2], where it was shown that a large class of algorithms called *reasonable algorithms* (to be defined precisely in the next section) are 2-competitive. A matching lower bound on the competitive ratio was also given, showing that this class of algorithms are optimal. They also showed that the offline case is NP-hard. In the paper they only considered the case where $R = 2$.

Motivation of this paper. The primary motivation of modelling the problem using unit-length jobs is to represent the job slices given to the processor by the operating system [2]. As such the actual cooling factor R relates to the length of this time quantum and the 'RC constant' of the processing unit (a thermal property related to how quickly the unit cools). Different systems appear to have very different values for these parameters (see e.g. [7]) and it is therefore important that we can design and analyse algorithms for different values of R.

Our results. For the case where $R > 2$, the results in [2] can be extended (giving the same competitive ratio) in a straightforward manner, which we omit in this conference version. Here we focus on the case $1 < R < 2$. We first show an upper bound for reasonable algorithms, for any $1 < R < 2$. We then give a matching lower bound, therefore showing that reasonable algorithms are in fact optimal for all values of R. The result also shows how the competitiveness depends on R: specifically, it increases as R gets smaller, and tends to infinity when R tends to 1. We also give some additional results on the multiple machines case and the weighted throughput case.

Because of space constraints, some proofs are omitted in this conference version and will appear in the full version of the paper.

2 The Algorithm and Its Analysis

2.1 The Algorithm

The algorithms take a decision at each time step deciding whether to schedule a job, and if it is decided to schedule a job, which job will be scheduled. A job j is *pending* at time u if j is released but not expired, $r_j \leq u < d_j$, and has not been scheduled before u. Job j is *admissible* at time u if it is pending and not too hot to be executed, i.e., $\tau_u + h_j \leq R$. Note that $h_j \leq R$ for all j since any hotter jobs can never be executed. As described in [2], a job j *dominates* another job k if $d_j \leq d_k$ and $h_j \leq h_k$. If at least one of the inequalities is strict then we say that j *strictly dominates* k.

An online algorithm is called *reasonable* if at each time step the algorithm schedules a job whenever there is one admissible, and if there are several admissible jobs, then any of these jobs that is not strictly dominated. If no jobs are admissible then the machine is left idle for a time step.

2.2 Competitive Analysis

Let OPT denote the offline optimal algorithm (the adversary) and $\mathcal{A}$ denote a reasonable algorithm. We use the same symbols to denote the schedules produced by the algorithms. The temperatures of OPT and $\mathcal{A}$ at a time t are denoted by τ'_t and τ_t, respectively.

Whenever, at some time u, $\mathcal{A}$ schedules a job j that is strictly hotter than a job k scheduled by OPT (with an idle time step in OPT being treated as though a job k with $h_k = 0$ is being executed) we call this a *relative-heating step*. Whenever $\tau_u > \tau'_u$ for some time u a relative-heating step must have occured before time u.

The general strategy to the competitive analysis is to map, or charge, the completed jobs in OPT to those in $\mathcal{A}$, such that (1) all jobs in OPT are charged to some jobs in $\mathcal{A}$, and (2) each job in $\mathcal{A}$ is being charged to by a bounded number of jobs in OPT. We use a charging scheme that is based on the one used in [2]. For a job j that has been scheduled by OPT at time u the charge of j is as follows:

Type-1 Charge: If $\mathcal{A}$ has also scheduled a job k at time u then charge j to k.

Type-2 Charge: If $\mathcal{A}$ is idle and hotter than OPT at time u but not at time $u+1$, that is $\tau_u > \tau'_u$ and $\tau_{u+1} \leq \tau'_{u+1}$, then there must have been a relative-heating step before time u. j is charged to the job k executed by $\mathcal{A}$ at the last relative heating step before u.

Type-3 Charges These charges are for the case when $\mathcal{A}$ is idle at time u and either $\mathcal{A}$ is cooler or the same as OPT at u, or $\mathcal{A}$ is hotter than OPT at time $u+1$ (and so must also have been hotter at time u), more formally $\tau_{u+1} > \tau'_{u+1}$ or $\tau_u \leq \tau'_u$. We divide Type-3 charges into two sub-types.

Type-3a: Type-3a charges are used when j has already been scheduled by $\mathcal{A}$ at some time $t < u$. In the case that $\tau_u \leq \tau'_u$ it must be that j has already been scheduled by $\mathcal{A}$ before u, because otherwise the fact that OPT has scheduled j means it cannot have expired, and as $\mathcal{A}$ is cooler than OPT if j had not been scheduled already it would have been admissible at u and so $\mathcal{A}$ would have chosen to schedule it.

To find a job to charge j to we construct a chain of jobs $j, j', \ldots, j^*$ like that constructed in [2]. The chain will be uniquely defined by j. If OPT is idle or schedules a job at time t that has a heat contribution equal to or greater than that of j then $j^* = j$, that is the last job in the chain is the copy of the job j in $\mathcal{A}$'s schedule. Otherwise OPT schedules some job j' at t that has a heat contribution that is strictly less than that of j. If j' is not scheduled by $\mathcal{A}$ at some time $t' < u$ then we also end the chain and have that $j^* = j$. Otherwise j' is added to the chain. If OPT is idle, schedules a hotter or equal job at t', or schedules a job j'' that is not scheduled by $\mathcal{A}$ at some time $t'' < u$ then we end the chain and have that $j^* = j'$. Otherwise we add j'' and the process continues. This process must end at some point as it deals with strictly cooler and cooler jobs and there are only a finite number of jobs.

Job j is then charged to j^* unless at the time t^* that j^* is scheduled in $\mathcal{A}$, OPT schedules a cooler job than j^* (that cannot be scheduled by $\mathcal{A}$ at any time $s < u$ by definition of the chain). In this case j is charged to the latest job already scheduled by $\mathcal{A}$, i.e., the job scheduled by $\mathcal{A}$ at the largest time s such that $s < u$. Such a time s must exist because at least the original job j must have already been scheduled by $\mathcal{A}$ by definition of a Type-3a charge. For simplicity we will refer to a Type-3a charge made through the chain as a Type-3a charge, and a Type-3a charge not made through the chain as a Type-3a* charge.

Type-3b: Type-3b charges are used when the job has not already been scheduled by $\mathcal{A}$. It must be the case that $\tau_{u+1} > \tau'_{u+1}$ because (as already argued) with the other Type-3 case of $\tau_u < \tau'_u$, j must have already been scheduled so a Type-3a charge must apply. With a Type-3b charge job j is charged to the latest job already scheduled by $\mathcal{A}$ before time u. Such a job must exist because there must have been at least one relative heating step in order for $\mathcal{A}$ to be hotter than OPT.

Although much of the charging scheme is the same as the one in [2], there are important differences that arise from the fact that $R < 2$, and new techniques are needed to handle them. In particular, the following lemma shows some properties of Type-3a* charges that are required in the analysis.

Lemma 1. *When there is a job j scheduled by OPT at time t that generates a Type-3a* charge the following are true:*

(1) $d_a > d_j$, where a is the job scheduled by OPT at the end of the Type-3a chain when $\mathcal{A}$ schedules j^*.
(2) $\tau'_t < \tau_t$ and $\tau'_{t+1} < \tau_{t+1}$.
(3) j would have been too hot to be scheduled by $\mathcal{A}$ at t if it were still pending.

Proof. (1) Let $j_0(= j), j_1, \ldots, j_{n-1}(= j^*), j_n(= a)$ be the chain. For simplicity we denote d_{j_i} by d_i and h_{j_i} by h_i. We claim that $d_n > d_i$ for all $0 \le i < n$; when $i = 0$ this gives $d_a > d_j$. The claim is proven by induction along the chain. By definition of the chain and the Type-3a* charge we have that $h_n < h_{n-1} < \ldots < h_1 < h_0$, and that a is not scheduled before t by $\mathcal{A}$. These two facts will be used repeatedly below.

First we show that $d_n > d_{n-1}$. We have that $h_a < h_{j^*}$ and that a is not scheduled before t by $\mathcal{A}$. This means that if $d_n \le d_{n-1}$ then $\mathcal{A}$ would have chosen to schedule a instead of j^* as j^* would be strictly dominated by a.

Now assume $d_n > d_{i'}$ for all $i \le i' \le n-1$. We show $d_n > d_{i-1}$. For each step in the chain we have a job j_{i+1} that is scheduled by OPT and a job j_i that is scheduled by $\mathcal{A}$ at the same time u, such that $h_{i+1} < h_i$. This leads to three cases that need to be covered in order to complete this proof.

In the first case job j_i is scheduled by OPT at some time $v < u$, as in Figure 1. $\mathcal{A}$ schedules some job j_{i-1} at time v as the next link in the chain. j_i was pending at v because it must have been released, as it was scheduled by OPT at this time, but $\mathcal{A}$ only schedules it at a later time u. Since $h_i < h_{i-1}$, it must be that $d_i > d_{i-1}$, otherwise j_i strictly dominates j_{i-1} and so would have been scheduled by $\mathcal{A}$ instead of j_{i-1} if this were the case. As we already know $d_n > d_i$ this gives $d_n > d_{i-1}$.

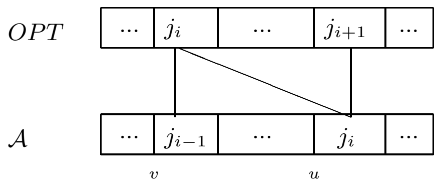


Fig. 1. $v < u$

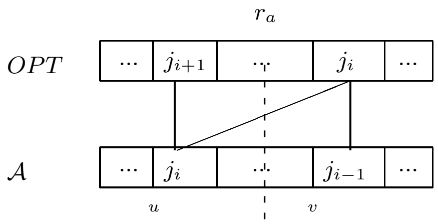


Fig. 2. $v > u$ and $v \ge r_a$

The second case is when j_i has been scheduled by OPT for the next link in the chain at some time v such that $v > u$ and $v \ge r_a$, as in Figure 2. This means that a must be pending in $\mathcal{A}$ at v. We also know that $h_a < h_{i-1}$. This means that $d_n > d_{i-1}$ otherwise job j_{i-1} would be strictly dominated by a and so $\mathcal{A}$ would have scheduled a instead.

The third case is when j_i has been scheduled by OPT for the next link in the chain at some time v such that $v > u$ and $v < r_a$, as in Figure 3. For this case to occur there must exist a job j_k such that $k > i$, that is j_k appears later in the chain than j_i, and j_k is scheduled by $\mathcal{A}$ at some time $w \ge r_a$ but is scheduled by OPT at some time $w' < v$. If this is not the case it would be impossible for the chain to have continued and ended at time $t^* \ge r_a$.

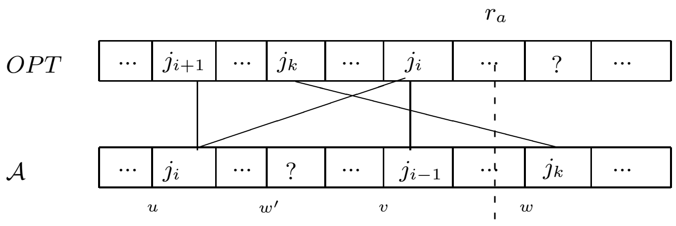

r_a
OPT
$\cdots$ j_{i+1} $\cdots$ j_k $\cdots$ j_i $\cdots$? $\cdots$
$\mathcal{A}$
$\cdots$ j_i $\cdots$? $\cdots$ j_{i-1} $\cdots$ j_k $\cdots$
u w' v w

Fig. 3. $v > u$ and $v < r_a$

As $k > i$ this means $h_{i-1} > h_i > h_k$. j_k must be pending at $\mathcal{A}$ at time v, as it has already been scheduled by OPT but not yet by $\mathcal{A}$, so it must be that $d_k > d_{i-1}$ otherwise j_k would have strictly dominated j_{i-1} and so been scheduled by $\mathcal{A}$ at v instead. By induction we can assume that $d_n > d_k$ so we can also conclude that $d_n > d_{i-1}$.

(2) This can be proven by contradiction. If $\tau'_t \geq \tau_t$, then as we know that a was pending at t and $h_a < h_j$, job a would have been admissible by $\mathcal{A}$ so would have been scheduled at t, contradicting that j generated a Type-3a* charge. Therefore $\tau'_t < \tau_t$. Now suppose $\tau'_{t+1} \geq \tau_{t+1}$ but $\tau'_t < \tau_t$. Then a Type-2 charge would have been generated at t contradicting the assumption that j generates a Type-3a* charge.

(3) At time t, since job a was pending, it must be too hot to be scheduled by $\mathcal{A}$, and as we know that $h_j > h_a$ it follows that even if j had not already been completed by $\mathcal{A}$ it would be too hot for $\mathcal{A}$ to schedule at t. □

Lemma 2. *If $\mathcal{A}$ remains idle at time t then $\tau_t - \tau'_t > \tau_{t+1} - \tau'_{t+1}$.*

Proof. Let $\alpha = \tau_t - \tau'_t$. Suppose OPT schedules a job j at time t (if OPT remains idle then treat it as executing a job with $h_j = 0$). Then $\tau_{t+1} - \tau'_{t+1} = \frac{\tau_t}{R} - \frac{\tau'_t + h_j}{R} = \frac{\alpha - h_j}{R} < \alpha$, since $R > 1$ and $h_j \geq 0$. □

Lemma 3. *Let N be the largest integer $n \geq 1$ such that $R^{n+1} < (n+2)R-(n+1)$ holds, for any fixed $1 < R < 2$. Consider any time interval when $\mathcal{A}$ is idle and the temperature of OPT is lower than that of $\mathcal{A}$ throughout this interval. Then during this time interval, OPT can schedule at most N jobs that are too hot to be scheduled by $\mathcal{A}$ in the same time step that they are scheduled in OPT.*

Proof. Suppose there are n jobs $j_1, j_2, \ldots, j_n$ in OPT that are scheduled consecutively by OPT but are too hot to be scheduled by $\mathcal{A}$, that $\mathcal{A}$ remains idle until all n jobs have been scheduled, and that after each job has been completed by OPT, the temperature of OPT is greater than that of $\mathcal{A}$. Suppose j_1 is scheduled at time u. See Figure 4. We will assume that between any of the jobs $j_1, \ldots, j_n$ in OPT's schedule, there are no idle time or jobs that do not satisfy the heat condition in the lemma statement. It will be shown that the inclusion of these will not make OPT able to schedule more 'hot' jobs (i.e. jobs that satisfy the heat condition of the lemma).

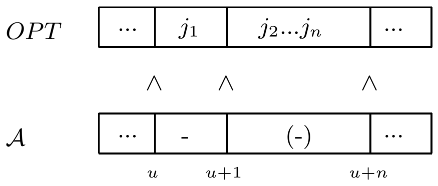


Fig. 4. The scenario with $\mathcal{A}$ remaining idle and OPT scheduling n jobs. The vertical inequality signs between the schedules show the relation between the temperatures.

Initially it must be that $\tau_u \leq 1$. As $\mathcal{A}$ stays idle for all n time steps we also have $\tau_{u+i} \leq 1/R^i$ for $i = 1, \ldots, n$. All of the jobs $j_1, \ldots, j_n$ are too hot to be scheduled by $\mathcal{A}$. Therefore, each job j_i, for every $1 \leq i \leq n$, has heat contribution

$$h_{j_i} > R - \tau_{u+i-1} \geq R - \frac{1}{R^{i-1}} \tag{1}$$

We now consider the temperature of OPT after executing these jobs. Initially $\tau'_u \geq 0$. After executing i jobs, $1 \leq i \leq n$, the temperature is given by the recursive formula $\tau'_{u+i} = \frac{\tau'_{u+i-1}+h_{j_i}}{R}$, which can be solved to give:

$$\tau'_{u+n} \geq \frac{R(1-R^{-n})}{R-1} - \frac{n}{R^n} \tag{2}$$

By the definition of the lemma we must have that after the n jobs have been completed, $\tau'_{u+n} < \tau_{u+n}$. Together with (1) and (2), this gives

$$\frac{R(1-R^{-n})}{R-1} - \frac{n}{R^n} < \frac{1}{R^n}$$

This is equivalent to:

$$R^{n+1} < (n+2)R - (n+1) \tag{3}$$

We now return to the assumption of OPT scheduling all of the n jobs consecutively. Our aim is to upper-bound the number of jobs that can be scheduled by OPT but are too hot to be scheduled $\mathcal{A}$ in a time period when $\mathcal{A}$ remains idle but is still hotter than OPT. If $j_1, \ldots, j_n$ are not consecutive then there are time steps in-between where OPT is idle or schedules a job that is not too hot to be scheduled by $\mathcal{A}$. By Lemma 2, such time steps will reduce the difference in temperature between OPT and $\mathcal{A}$, thus will not help OPT schedule more hot jobs. □

Theorem 1. *Any job scheduled by $\mathcal{A}$ can receive at most $N+2$ charges, where N is the largest integer $n \geq 1$ such that inequality (3) holds, therefore any reasonable algorithm $\mathcal{A}$ is $N+2$ competitive for any fixed $1 < R < 2$.*

Proof. Each job in $\mathcal{A}$ can receive at most one Type-1 charge. In addition to this each job in $\mathcal{A}$ receives at most one Type-2 charge. This is because in between any two time steps that satisfy Type-2 conditions there must be a relative-heating step, so Type-2 charges will be assigned to distinct relative-heating steps. A job can also receive at most one Type-3a charge through a chain because the chains formed are disjoint and uniquely defined by the original job that generated the Type-3a charge. If a job j receives a Type-1 charge it cannot receive both a Type-2 charge and a Type-3a charge through a chain. This is because a Type-3a charge through the chain cannot be made to a job at a relative heating step, otherwise the chain would either continue further, or a Type-3a* charge would be generated instead. This means that any job can receive at most 2 charges from the set {Type-1, Type-2, Type-3a (chain)}.

Lemma 3 can be used to show that a job can receive at most N Type-3b and Type-3a* charges, where N is the largest integer $n \geq 1$ such that inequality (3) holds. Both these charges are made to the job that was most recently scheduled by $\mathcal{A}$. To prove this we must now show that jobs that generate Type-3b and Type-3a* charges are both covered by Lemma 3, in that they are too hot to be scheduled by $\mathcal{A}$ at that time, and they also ensure that the temperature of $\mathcal{A}$ is greater than that of OPT after they have completed. Lemma 1 shows that jobs that generate Type-3a* charges meet these requirements, leaving us to show only that Type-3b charges also meet these conditions. By definition of a Type-3b charge it is already established that the temperature of $\mathcal{A}$ is greater than that of OPT after the job has completed. Also by definition of a Type-3b charge the job that generates it cannot have already been completed by $\mathcal{A}$ and $\mathcal{A}$ remains idle while OPT schedules the job. This means that the job must have been too hot to be scheduled by $\mathcal{A}$ at that time otherwise it would have been admissible to $\mathcal{A}$ so $\mathcal{A}$ would have had to schedule it, contradicting that $\mathcal{A}$ remains idle.

It has been shown that any job can receive at most N Type-3a* and Type-3b charges, and 2 charges from the set {Type-1, Type-2, Type-3a (chain)}. This means that each job can receive at most $N + 2$ charges. □

The table in Figure 5 shows the competitive ratio of any reasonable algorithm for different values of R. It can be seen that the competitive ratio increases as R decreases. The competitive ratio is still fairly reasonable for moderate values of R, but goes to infinity as R gets very close to 1.

3 Lower Bound Construction

In this section we show a lower bound for all deterministic algorithms showing that all reasonable algorithms are in fact optimal for all $1 < R < 2$.

Theorem 2. *Any deterministic algorithm has a competitive ratio of at least* $N + 1$ *for any fixed* $1 < R < 2$ *where* N *is the largest integer* $n \geq 1$ *such that* $R^n < (n+1)R - n$ *holds.*

Proof. Fix some deterministic algorithm $\mathcal{A}$. At $t = 0$ release a job j that has a very large deadline and a heat contribution of just below R, that is $d_j = 2D + N + 1$ for a large D and $h_j = R - \epsilon$ for a sufficiently small $\epsilon > 0$. D must

R	**Upper Bound**
$2 > R > \frac{-1+\sqrt{13}}{2} \approx 1.30278$	3
$1.30278 \geq R \gtrsim 1.15091$	4
$1.15091 \gtrsim R \gtrsim 1.09128$	5
$1.09128 \gtrsim R \gtrsim 1.0614$	6
$1.0614 \gtrsim R \gtrsim 1.04421$	7
$1.04421 \gtrsim R \gtrsim 1.03339$	8
$1.03339 \gtrsim R \gtrsim 1.02612$	9
$1.02612 \gtrsim R \gtrsim 1.02100$	10
1.01	15
1.001	45
1.0001	142

Fig. 5. Table of upper bounds

be large enough so that OPT's temperature will have cooled down from 1 to ϵ after D steps, i.e. $R^D \geq 1/\epsilon$. If $\mathcal{A}$ never schedules j then OPT does and $\mathcal{A}$ has an unbounded competitive ratio. Otherwise $\mathcal{A}$ schedules j at some time u. We consider two cases.

If $u < D$, then OPT chooses not to start j yet. At $t = u + i$ for $i = 1, 2, \ldots$ the temperature of $\mathcal{A}$ is given by $\tau_{u+i} = 1/R^{i-1}$. At each such t, as long as $\tau_{u+i} > \tau'_{u+i}$, the adversary releases a tight job j_i with $d_{j_i} = u + i + 1$ and the smallest possible heat contribution such that $h_{j_i} > R - 1/R^{i-1}$. $\mathcal{A}$ will be (just) too hot to schedule the job due to scheduling job j while OPT can and will schedule each of them. We will prove that this stops after N steps. OPT will then remain idle for D time steps until it is cool enough to schedule j. Since $u + 1 + N + D < 2D + N + 1$, OPT can still finish j before its deadline.

If $u \geq D$, then OPT can schedule j to start at $t = 0$. Since $u \geq D$, at time $u+1$ OPT's temperature will be less than ϵ, which can be made arbitrarily close to 0. Similar to the previous case, at each $t = u + i$ where $i = 1, 2, \ldots$ we release a tight job where $d_{j_i} = u + i + 1$ and $h_{j_i} > R - 1/R^{i-1}$. $\mathcal{A}$ will be just too hot to run any of these jobs, and this continues as long as $\tau_{u+i} > \tau'_{u+i}$, so OPT can finish all these jobs.

In both cases OPT will schedule $N + 1$ jobs (the N tight jobs that are generated in successive time steps plus j) whereas $\mathcal{A}$ will only ever be able to schedule j. Thus the competitive ratio is $N + 1$.

The temperature of OPT at time $u + i$ is given by:

$$\tau'_{u+1} \approx 0, \quad \tau'_{u+i} \approx \frac{\tau_{u+i-1} + R - \frac{1}{R^{i-2}}}{R} \text{ for } i > 1$$

This can be solved so we have:

$$\tau'_{u+i} \approx \frac{R}{R-1}\left(1 - \frac{1}{R^{i-1}}\right) - \frac{i-1}{R^{i-1}}$$

The sequence of jobs continues as long as $\tau'_{u+i} < \tau_{u+i}$. Hence N is the largest value of n such that $\tau'_{u+n} < \tau_{u+n}$. This gives

$$\frac{R}{R-1}\left(1-\frac{1}{R^{n-1}}\right)-\frac{n-1}{R^{n-1}}<\frac{1}{R^{n-1}}$$

which is equivalent to

$$R^n < (n+1)R - n \tag{4}$$

Thus the sequence indeed stops after N steps with N given by the formula above. □

By comparing the formulas in Theorems 1 and 2, it can be seen easily that the lower bound matches the upper bound, and thus reasonable algorithms are optimal for all ranges of values of R.

4 Extensions

4.1 Multiple Machines

All the above results consider the single processor case. New machines come with multicore processors where each core can process jobs independently. In the multicore or multiple machine setting, we assume jobs can be executed in any one of the $m > 1$ identical machines. Each machine has a separate temperature, each of which cannot be above the (same) threshold. For simplicity, we assume there is no heat transfer between different processors. It has been suggested that such lateral heat transfer is small (see e.g. [8] and the discussion therein), and such consideration would require information of the physical layout of the processing units. Thus the temperature of each machine is determined independently, in the same way as the single machine case.

It is straightforward to generalise the definition of reasonable algorithms to the multiple machine case: an algorithm is reasonable if it does not leave a machine idle if there are jobs admissible on that machine, and any job allocated to a machine is not dominated by some other pending jobs. An example reasonable algorithm would be to arrange the pending jobs in coolest first order, and allocate them one by one to any machine on which it is admissible.

The upper bound proof of Theorem 1 can be adapted to show that any reasonable algorithm also has the same competitive ratio, for any number of machines m. We omit the proof as it is very similar to Theorem 1.

Theorem 3. *All reasonable algorithms are $(N+2)$-competitive for any number of machines m, where N is the largest integer n such that inequality (3) holds.*

Next we show a lower bound for any deterministic algorithms in the multiple machine case.

Theorem 4. *Any deterministic algorithm has a competitive ratio of at least $2-\frac{1}{N}$ for any fixed $1 < R < 2$ where N is the largest integer $n \geq 2$ such that $R^n < (n+1)R - n$ holds, for any number of machines m.*

Proof. Fix any deterministic algorithm $\mathcal{A}$. At time 0 release m tight jobs J that has the maximum heat, that is for every $j \in J$, $d_j = 1$ and $h_j = R$. We set a threshold of λm for some $1 \geq \lambda \geq 0$ to be fixed later. This gives the following two possibilities.

If $\mathcal{A}$ schedules less than λm of the J jobs then OPT can schedule all m of the jobs and no further jobs are released. This gives a competitive ratio of at least $\frac{m}{\lambda m}$ which is equal to $\frac{1}{\lambda}$.

If $\mathcal{A}$ schedules at least λm of the J jobs then starting at time 1 we release m copies of the N tight jobs that are released in the single machine lower bound in Theorem 2. The value for N can be worked out for a fixed R as before, using inequality (4). These N jobs will then be scheduled by $\mathcal{A}$ on each of the $(1-\lambda)m$ machines that have not scheduled a J job, as any machine that schedules a J job will remain too hot to schedule any of the N jobs. OPT will remain idle for the first time step and be able to schedule N jobs on each of its m machines. This gives a competitive ratio of at least $\frac{Nm}{\lambda m+(1-\lambda)mN}$ which is equal to $\frac{N}{\lambda+(1-\lambda)N}$.

We must now find the best value for λ. As the ratio $\frac{1}{\lambda}$ decreases with an increase in λ, while $\frac{N}{\lambda+(1-\lambda)N}$ increases with λ, to find the optimal λ value we need to find the value for λ where these two ratios are equal for a given N. This gives $\lambda = \frac{N}{2N-1}$, and using this λ gives a lower bound of $\frac{1}{\lambda} = 2 - \frac{1}{N}$. $\square$

This bound is far lower than the (optimal) single machine lower bound and the multiple machine upper bound, thus it is entirely possible to give better algorithms in the multiple machine case. However we observe that in order to do this, some machines must stay idle even when there are admissible jobs:

Theorem 5. *For $1 < R < 2$, for any algorithm that always schedules all machines with jobs as long as an admissible one is available, the competitive ratio is at least the same as that given in Theorem 2, for any number of machines m.*

Proof. We simply consider a job instance which consists of m identical copies of jobs used in the lower bound construction of Theorem 2. All machines will then run the jobs at time 0 and thus miss all later jobs, while OPT remains idle at the first time step and schedule all $N+1$ jobs on each of its machines. $\square$

4.2 Weighted Throughput

A weighted version of this problem can be considered, where each job has a weight and the objective is to maximize the total weight of completed jobs. Note that without heat contributions, this weighted version of the problem is known as *unit job scheduling* which is by itself a very challenging problem (see e.g. [4]). If W is the ratio of maximum to minimum job weights, then any deterministic algorithm is $\Omega(W)$-competitive, and any randomized algorithm is $\Omega(\log W)$-competitive, for any value of R [5]. Matching upper bounds were obtained in [5], but only for $R \geq 2$, by combining the constant upper bound in the unweighted case [2] with the standard 'classify and randomly select' technique. Theorem 1 gives constant upper bounds for any fixed $R < 2$; hence, we have that the upper bounds for the weighted case are indeed $O(W)$ and $O(\log W)$ for the deterministic and randomized cases, respectively, for any constant value of R.

5 Final Comments

There are still many open problems for this area. Not much is known about randomized algorithms for this and similar problems. It is possible to prove a lower bound of 1.5 for the case where $R \geq 2$ [2] but so far no improvement on a 2-competitive algorithm has been shown.

Instead of idling when the threshold is reached, one could consider reducing the frequency of the processor. There could be discrete multiple frequencies, or it could be a continuous range of frequencies.

Another realistic scenario would be to study longer (non unit length) jobs, either with equal or with varying lengths. This leads to scheduling problems with preemption.

The multiple machines scenario would also be of interest. This could also be combined with preemption (for longer than unit jobs) to give migration of jobs to different machines to control temperature. Although Section 4.1 gave some preliminary results for the multiple machines case, the bounds are not tight, in particular it seems likely that the competitive ratio should be lower than the single machine case.

References

1. Bansal, N., Kimbrel, T., Pruhs, K.: Dynamic speed scaling to manage energy and temperature. In: FOCS 2004: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 520–529 (2004)
2. Chrobak, M., Dürr, C., Hurand, M., Robert, J.: Algorithms for temperature-aware task scheduling in microprocessor systems. In: Fleischer, R., Xu, J. (eds.) AAIM 2008. LNCS, vol. 5034, pp. 120–130. Springer, Heidelberg (2008)
3. Coskun, A.K., Rosing, T.S., Whisnant, K.: Temperature aware task scheduling in MPSoCs. In: Proceedings of 2007 Design, Automation and Test in Europe Conference, pp. 1659–1664 (2007)
4. Englert, M., Westermann, M.: Considering suppressed packets improves buffer management in QoS switches. In: SODA 2007: Proceedings of 18th ACM-SIAM Symposium on Discrete Algorithms, pp. 209–218 (2007)
5. Fung, S.P.Y.: Online algorithms for maximizing weighted throughput of unit jobs with temperature constraints. In: MAPSP 2009: 9th Workshop on Models and Algorithms for Planning and Scheduling Problems (2009)
6. Irani, S., Pruhs, K.: Algorithmic problems in power management. SIGACT News 36(2), 63–76 (2005)
7. Kursen, E., Cher, C.-Y., Buyuktosunoglu, A., Bose, P.: Investigating the effects of task scheduling on thermal behavior. In: 3rd Workshop on Temperature-Aware Computer Systems (2006)
8. Yang, J., Zhou, X., Chrobak, M., Zhang, Y., Jin, L.: Dynamic thermal management through task scheduling. In: Proceedings of IEEE Int. Symposium on Performance Analysis of Systems and Software, pp. 191–201 (2008)

On Solution Concepts for Matching Games

Péter Biró[1,⋆], Walter Kern[2], and Daniël Paulusma[3,⋆⋆]

[1] Department of Computing Science, Sir Alwyn Williams Building, University of Glasgow, Glasgow G12 8QQ, Scotland
pbiro@dcs.gla.ac.uk

[2] Faculty of Electrical Engineering, Mathematics and Computer Science, University of Twente, P.O. Box 217, NL-7500 AE Enschede
w.kern@math.utwente.nl

[3] Department of Computer Science, University of Durham, Science Laboratories, South Road, Durham DH1 3EY, England
daniel.paulusma@durham.ac.uk

Abstract. A matching game is a cooperative game (N, v) defined on a graph $G = (N, E)$ with an edge weighting $w : E \to \mathbb{R}_+$. The player set is N and the value of a coalition $S \subseteq N$ is defined as the maximum weight of a matching in the subgraph induced by S. First we present an $O(nm + n^2 \log n)$ algorithm that tests if the core of a matching game defined on a weighted graph with n vertices and m edges is nonempty and that computes a core allocation if the core is nonempty. This improves previous work based on the ellipsoid method. Second we show that the nucleolus of an n-player matching game with nonempty core can be computed in $O(n^4)$ time. This generalizes the corresponding result of Solymosi and Raghavan for assignment games. Third we show that determining an imputation with minimum number of blocking pairs is an NP-hard problem, even for matching games with unit edge weights.

1 Introduction

Consider a group N of tennis players that will participate in a doubles tennis tournament. Suppose that each pair of players can estimate the expected prize money they could win together if they form a pair in the tournament. Also suppose that each player is able to negotiate his share of the prize money with his chosen partner, and that each player wants to maximize his own prize money. Can the players be matched together such that no two players have an incentive to leave the matching in order to form a pair together? This is the example Eriksson and Karlander [7] used to illustrate the stable roommates problem with payments. In this paper we consider the situation in which groups of *possibly more than two* players in a doubles tennis tournament can distribute their total prize money among each other. For instance, suppose that three players always form pairs among themselves. Then they may decide that only two of them will

⋆ Supported by EPSRC grant EP/E011993/1 and by OTKA grant K69027.

⋆⋆ Supported by EPSRC grants EP/D053633/1 and EP/G043434/1.

© Springer-Verlag Berlin Heidelberg 2010

form a pair in this tournament, but that the third player will be compensated for his loss of income. Now the question is whether the players can be matched together such that no group of players will be better off when leaving the matching. This scenario is an example of a matching game. Matching games are well studied within the area of Cooperative Game Theory. In order to explain these games and how they are related to the first problem setting, we first state the necessary terminology and formal definitions.

A *cooperative game* (N, v) is given by a set N of n *players* and a *value function* $v : 2^N \to \mathbb{R}$ with $v(\emptyset) = 0$. A *coalition* is any subset $S \subseteq N$. We refer to $v(S)$ as the *value* of coalition S, i.e., the maximal profit or the minimal costs that the players in S achieve by cooperating with each other. In the first case we also speak of (N, v) as a *profit game* and in the second case we also say that (N, v) is a *cost game*. The v-values of many cooperative games are derived from solving an underlying discrete optimization problem (cf. Bilbao [3]). It is often assumed that the *grand coalition* N is formed, because in many games the total profit or costs are optimized if all players work together. The central problem is then how to allocate the *total value* $v(N)$ to the individual players in N. An *allocation* is a vector $x \in \mathbb{R}^N$ with $x(N) = v(N)$, where we adopt the standard notation $x(S) = \sum_{i \in S} x_i$ for $S \subseteq N$. A *solution concept* $\mathcal{S}$ for a class of cooperative games Γ is a function that maps each game $(N, v) \in \Gamma$ to a set $\mathcal{S}(N, v)$ of allocations for (N, v). These allocations are called $\mathcal{S}$-*allocations.*

The choice for a specific solution concept $\mathcal{S}$ not only depends on the notion of "fairness" specified within the decision model but also on certain computational aspects, such as the computational complexity of testing nonemptiness of $\mathcal{S}(N, v)$, or computing an allocation in $\mathcal{S}(N, v)$. Here we take the size of the underlying discrete structure as the natural input size, instead of the 2^n v-values themselves.

We will now define some well-known solution concepts in terms of profit games; see Owen [18] for a survey. First, the *core* of a game (N, v) consists of all allocations x with $x(S) \geq v(S)$ for all $S \in 2^N \backslash \{\emptyset, N\}$. Core allocations are fair in the sense that every nonempty coalition S receives at least its value $v(S)$. Therefore, players in a coalition S do not have any incentive to leave the grand coalition (recall the doubles tennis tournament). However, for many games, the core might be empty. Therefore, other solution concepts have been designed, such as the nucleolus and the nucleon, both defined below.

Let (N, v) be a cooperative game. The *excess* of a nonempty coalition $S \subsetneq N$ regarding an allocation $x \in \mathbb{R}^N$ expresses the satisfaction of S with x and is defined as $e(S, x) := x(S) - v(S)$. We order all excesses $e(S, x)$ into a non-decreasing sequence to obtain the *excess vector* $\theta(x) \in \mathbb{R}^{2^n - 2}$. The *nucleolus* of (N, v) is then defined as the set of allocations that lexicographically maximize $\theta(x)$ over all *imputations*, i.e., over all allocations $x \in \mathbb{R}^N$ with $x_i \geq v(\{i\})$ for all $i \in N$. The nucleolus is not defined if the set of imputations is empty. Otherwise, Schmeidler [20] showed it consists of exactly one allocation. Note that, by definition, the nucleolus lies in the core if the core is nonempty. The standard procedure for computing the nucleolus proceeds by solving up to n

linear programs, which have exponential size in general. We refer to Maschler, Peleg and Shapley [14] for more details. Taking multiplicative excesses $e'(S,x) = x(S)/v(S)$ in the definition of the nucleolus leads to the *nucleon* of a game.

Matching games. In a *matching game* (N,v), the underlying discrete structure is a finite undirected graph $G=(N,E)$ that has no loops and no multiple edges and that is *weighted*, i.e., on which an edge weighting $w : E \rightarrow \mathbb{R}_+$ has been defined. The players are represented by the vertices of G, and for each coalition S we define $v(S) = w(M) = \sum_{e \in M} w(e)$, where M is a maximum weight matching in the subgraph of G induced by S. If $w \equiv 1$, then $v(S)$ is equal to the size of a maximum matching and we call (N,E) a *simple* matching game. If there exists a weighting $w^* : N \rightarrow \mathbb{R}_+$ such that $w(uv) = w^*(u) + w^*(v)$ for all $uv \in E$, then we obtain a *node matching game*. Note that every simple matching game is a node matching game by choosing $w^* \equiv \frac{1}{2}$. Matching games defined on a bipartite graph are called *assignment games*.

The core of a matching game can be empty. In order to see this, consider a simple matching game (N,v) on a triangle with players a,b,c. An allocation x in the core must satisfy $x_a+x_b \geq 1$, $x_a+x_c \geq 1$, and $x_b+x_c \geq 1$, and consequently, $x(N) = x_a+x_b+x_c \geq \frac{3}{2}$. However, this is not possible due to $x(N) = v(N) = 1$. Shapley and Shubik [21] show that the core of an assignment game is always nonempty.

We will now discuss complexity aspects of solution concepts for matching games. First let us recall the following result of Gabow [9].

Theorem 1 ([9]). *A maximum weight matching of a weighted graph on n vertices and m edges can be computed in $O(nm + n^2 \log n)$ time.*

The following observation can be found in several papers, see e.g. [5,7,19]. Here, a *cover* of a graph $G=(N,E)$ with edge weighting w is a vertex mapping $c : N \rightarrow \mathbb{R}_+$ such that $c(u)+c(v) \geq w(uv)$ for each edge $uv \in E$. The *weight* of c is defined as $c(N) = \sum_{u \in N} c(u)$.

Observation 1. *Let (N,v) be a matching game on a weighted graph $G=(N,E)$. Then $x \in \mathbb{R}^N$ is in the core of (N,v) if and only if x is a cover of G with weight $v(N)$.*

Observation 1 and Theorem 1 imply that testing core nonemptiness can be done in polynomial time for matching games by using the ellipsoid method for solving linear programs [12]. Deng, Ibaraki and Nagamochi [5] characterize when the core of a simple matching game is nonempty. In this way they can compute a core allocation of a simple matching game in polynomial time, without having to rely on the ellipsoid method. Eriksson and Karlander [7] characterize the extreme points of the core of a matching game. Observation 1 implies that the size of the linear programs involved in the procedure of Maschler, Peleg and Shapley [14] is polynomial in the case that the matching game has a nonempty core [19]. Hence the nucleolus of such matching games can be computed in polynomial time by using the ellipsoid method at most n times. Solymosi and Raghavan [23] compute the nucleolus of an assignment game without making use of the ellipsoid method.

Matsui [15] present a faster algorithm for computing the nucleolus of assignment games defined on bipartite graphs that are unbalanced.

Theorem 2 ([23]). *The nucleolus of an n-player assignment game can be computed in $O(n^4)$ time.*

It is known [11] that the nucleolus of a simple matching game can be computed in polynomial time by using the standard procedure of Maschler, Peleg and Shapley [14], after reducing the size of the involved linear programs to be polynomial. This result is extended to node matching games [19]. For matching games, Faigle et al. [8] show how to compute an allocation in the nucleon in polynomial time. Determining the computational complexity of finding the nucleolus for general matching games is still open.

Connection to the stable roommates problem. We refer to a survey [4] for more on this problem. Here, we only define the variant with payments. Let $G = (N, E)$ be a graph with edge weighting w. Let $B(x) = \{(u, v) \mid x_u + x_v < w(uv)\}$ denote the set of *blocking pairs* of a vector $x \in \mathbb{R}^N$. A vector $p \in \mathbb{R}^N$ with $p_u \geq 0$ for all $u \in N$ is said to be a *payoff* with respect to a matching M in G if $p_u + p_v = w(uv)$ for all $uv \in M$, and $p_u = 0$ for each u that is not incident to an edge in M. The problem Stable Roommates with Payments tests if a weighted graph allows a *stable solution*, i.e., a pair (M, p), where p is a payoff with respect to matching M such that $B(p) = \emptyset$. We also call such a pair *stable*. This problem is polynomially solvable due to following well-known observation (cf. Eriksson and Karlander [7]).

Observation 2. *An allocation x of the matching game defined on a weighted graph G is a core allocation if and only if there exists a matching M in G such that (M, x) is stable.*

Any core allocation of a matching game is an imputation with an empty set of blocking pairs. If the core is empty, then we can try to minimize the number of blocking pairs. This leads to the following decision problem, which is trivially solvable for assignment games, because these games have a nonempty core [21].

Blocking Pairs
Instance: a matching game (N, v) and an integer $k \geq 0$.
Question: does (N, v) allow an imputation x with $|B(x)| \leq k$?

Our results and paper organization. In Section 2 we present an $O(nm + n^2 \log n)$ time algorithm that tests if the core of a matching game on a weighted n-vertex graph with m edges is nonempty and that computes a core allocation if it exists. Like the algorithm of Deng, Ibaraki and Nagamochi [5] for simple matching games, our algorithm does not rely on the ellipsoid method. By Observation 2 we can use this algorithm to find stable solutions for instances of Stable Roommates with Payments. In Section 3 we generalize Theorem 2 by showing that the nucleolus of an n-player matching game with nonempty core can be computed in $O(n^4)$ time. Klaus and Nichifor [13] investigate the relation of the core with other solution concepts for matching games and express the need of a comparison of matching games with nonempty core to assignment games. As the results in Sections 2 and 3 are based on a duplication technique yielding bipartite graphs, our paper gives such a comparison. In Section 4 we show that the BLOCKING PAIRS problem is NP-complete, even for simple matching games. We note that, in the context of stable matchings without payments, minimizing the number of blocking pairs is NP-hard as well [1]. This problem setting is quite different from ours, and we cannot use the proof of this result for our means.

2 The Core of a Matching Game

As mentioned in the previous section, Shapley and Shubik [21] showed that every assignment game has a nonempty core. However, they did not analyze the computational complexity of finding a core allocation. In this section we consider this question for matching games. First we introduce some terminology.

Let $G = (N, E)$ be a graph with edge weighting $w : E \to \mathbb{R}_+$. We write $v \in e$ if v is an end vertex of edge e. A *fractional matching* is an edge mapping $f : E \to \mathbb{R}_+$ such that $\sum_{e:v\in e} f(e) \leq 1$ for each $v \in N$. The weight of a fractional matching f is defined as $w(f) = \sum_{e\in E} w(e)f(e)$. We call f a *matching* if $f(e) \in \{0, 1\}$ for all $e \in E$, and we call f a *half-matching* if $f(e) \in \{0, \frac{1}{2}, 1\}$ for all $e \in E$.

We state two useful lemmas. The first lemma is a direct application of the Duality Theorem (cf. Schrijver [22]). The second lemma is a special case of Theorem 1 from Deng, Ibaraki and Nagamochi [5].

Lemma 3. *Let $G = (N, E)$ be a graph with edge weighting w. Let f be a fractional matching of G and let c be a cover of G. Then $w(f) \leq c(N)$, with equality if and only if f has maximum weight and c has minimum weight.*

Lemma 4 ([5]). *Let (N, v) be a matching game on a weighted graph $G = (N, E)$. Then the core of (N, v) is nonempty if and only if the maximum weight of a matching in G equals the maximum weight of a fractional matching in G.*

We also need the following theorem, which is a straightforward consequence of a result by Balinski [2].

Theorem 3 ([2]). *Let G be a weighted graph. Then the maximum weight of a half-matching of G is equal to the minimum weight of a cover of G.*

Lemma 3 and 4 together with Theorem 3 characterize the core of a matching game.

Proposition 1. *Let (N, v) be a matching game on a weighted graph $G = (N, E)$. The core of (N, v) is nonempty if and only if the maximum weight of a matching in G is equal to the maximum weight of a half-matching in G.*

Eriksson and Karlander [7] consider the problem STABLE ROOMMATES WITH PAYMENTS and characterize stable solutions in terms of forbidden minors. We can translate their characterization as follows. A weighted G allows a stable pair (M, x) if and only if the weight of M cannot be improved via any half-integer alternating path. Such an improvement is possible if and only if the maximum weight of a half-matching in G is greater than the maximum weight of a matching in G. Hence their result [7] follows directly from Observation 2 and Proposition 1.

We will use Proposition 1 in the proof of Theorem 5, in which we present a polynomial-time algorithm for finding a core allocation of a matching game, if such an allocation exists. We also need the following result by Egerváry [6], which holds for bipartite graphs.

Theorem 4 ([6]). *Let G be a weighted bipartite graph. Then the maximum weight of a matching in G is equal to the minimum weight of a cover of G.*

Theorem 4 can also be used in an alternative proof of Theorem 3 based on a duplication technique, as described by Nemhauser and Trotter [17]. Since we need this technique in our proof of Theorem 5, we discuss it below.

Let (N, v) be a matching game defined on graph $G = (N, E)$ with edge weighting w. We replace each vertex u by two copies u', u'' and each edge $e = uv$ by two edges $e' = u'v''$ and $e'' = u''v'$. We define edge weights $w^d(e') = w^d(e'') = \frac{1}{2}w(e)$ for each $e \in E$. This yields a weighted bipartite graph $G^d = (N^d, E^d)$ in $O(n^2)$ time. We call G^d the *duplicate* of G. We compute a maximum weight matching f^d of G^d in $O(nm + n^2 \log n)$ time due to Theorem 1. Given f^d, we compute a minimum weight cover c^d of G^d in the same time (cf. Theorem 17.6 from Schrijver [22]). We compute the half-matching f in G defined by $f(e) := \frac{f^d(e')+f^d(e'')}{2}$ for each $e \in E$ in $O(n^2)$ time and note that

$$w(f) = \sum_{e\in E} w(e)f(e) = \sum_{e\in E} \left(w^d(e')f^d(e') + w^d(e'')f^d(e'')\right) = w^d(f^d).$$

We define $c : N \to \mathbb{R}_+$ in $O(n)$ time by $c(u) := c^d(u') + c^d(u'')$ for all $u \in N$ and deduce that $c(u)+c(v) = c^d(u')+c^d(u'')+c^d(v')+c^d(v'') \geq w^d(u'v'')+w^d(u''v') = \frac{1}{2}w(uv)+\frac{1}{2}w(uv) = w(uv)$. This means that c is a cover of G with $c(N) = c^d(N^d)$, and by Theorem 4, we deduce that

$$w(f) = w^d(f^d) = c^d(N^d) = c(N). \qquad (1)$$

We observe that f is a maximum weight half-matching due to Lemma 3. Hence equation (1) implies Theorem 3. We compute a maximum weight matching f^* of G in $O(nm + n^2 \log n)$ time due to Theorem 1. Then, by Proposition 1, we just need to check whether $w(f^*) = w(f)$ in order to find out if the core of (N, v) is nonempty. Suppose the core of (N, v) is nonempty. Since $w(f) = c(N)$ and $w(f^*) = v(N)$, we obtain

$$c(N) = v(N). \qquad (2)$$

Hence, c is a core member due to Observation 1, and we get the following result.

Theorem 5. *There exists an $O(nm + n^2 \log n)$ time algorithm that tests if the core of a matching game on a graph with n vertices and m edges is nonempty and that computes a core allocation in the case that the core is nonempty.*

For simple matching games, we improve the running time of the algorithm in Theorem 5 to $O(\sqrt{n}m)$ by using the $O(\sqrt{n}m)$ time algorithm of Micali and Vazirani [16] for computing a maximum matching in a graph with n vertices and m edges.

3 The Nucleolus of a Matching Game with Nonempty Core

We start with some extra terminology. For a matching game (N, v) defined on a weighted graph $G = (N, E)$ we define its *duplicate* as the assignment game (N^d, v^d) defined on G^d with edge weights w^d. The *duplicate* of a vector $x \in \mathbb{R}^N$ is the *symmetric* vector x^d given by $x^d_{u'} = x^d_{u''} = \frac{1}{2}x_u$ for all $u \in N$.

Lemma 5. *Let (N, v) be a matching game with nonempty core. Then the nucleolus of (N^d, v^d) is the duplicate of the nucleolus of (N, v).*

Proof. Let η^* be the nucleolus of (N^d, v^d). Define $\overline{\eta}$ by $\overline{\eta}_{u'} = \eta^*_{u''}$ and $\overline{\eta}_{u''} = \eta^*_{u'}$ for all $u \in N$. Then $\theta(\eta^*) = \theta(\overline{\eta})$. Because η^* is unique as shown by Schmeidler [20], we find that η^* must be symmetric. Let η be such that $\eta^d = \eta^*$. We observe that $\theta(x) \preceq \theta(y)$ if and only if $\theta(x^d) \preceq \theta(y^d)$ for two imputations x and y of (N, v). Hence, we are left to prove that η is an imputation.

Note that $\eta_u \geq 0$ for all $u \in N$. We now show $\eta(N) = v(N)$. By definition, η^d is in the core of (N^d, v^d). Observation 1 implies that η^d is a cover of G^d with weight $\eta^d(N^d) = v^d(N^d) = w^d(f^d)$, where f^d is a maximum weight matching of G^d. By Lemma 3, η^d is a minimum weight cover of (N^d, v^d). Then, by equation (2), we derive $\eta(N) = v(N)$. Hence, η is indeed an imputation of (N, v). □

Theorem 6. *The nucleolus of an n-player matching game with nonempty core can be computed in $O(n^4)$ time.*

Proof. Let (N, v) be an n-player matching game with nonempty core that is defined on a graph G with edge weighting w. We create G^d and w^d in $O(n^2)$ time. Note that $|N^d| = 2n$. By Theorem 2 we compute the nucleolus η^d of (N^d, v^d) in $O(n^4)$ time. From η^d we construct η in $O(n^2)$ time. By Lemma 5 we find that η is the nucleolus of (N, v). This finishes the proof of Theorem 6. □

4 Blocking Pairs in a Matching Game

Fixing parameter k makes the BLOCKING PAIRS problem polynomially solvable. This can be seen as follows. We choose a set B of k blocking pairs. Then we use the ellipsoid method to check in polynomial time if there exists an imputation x with $x_u + x_v \geq w(uv)$ for all pairs $uv \notin B$. Because k is fixed, the total number of choices is bounded by a polynomial in n. What happens when k is part of the input? Before we present our main result, we start with a useful lemma.

Lemma 6. *Let K be a complete graph with vertex set $\{1, \ldots, \ell\}$ for some odd integer ℓ, and let $x \in \mathbb{R}^K_+$. If $x(K) < \frac{\ell}{2}$ then $|B(x)| \geq \frac{\ell-1}{2}$ holds.*

Proof. Write $\ell = 2q + 1$ and use induction on q. If $q = 1$ the statement holds. Suppose $q \geq 2$. We assume without loss of generality that $x_1 \leq x_2 \leq \cdots \leq x_{2q+1}$ holds. Since $x(K) < \frac{\ell}{2}$, we have $x_1 < \frac{1}{2}$. If $x_1 + x_{2q+1} < 1$ then $x_1 + x_i < 1$ for $2 \leq i \leq 2q+1$. Hence, we have at least $2q$ blocking pairs. Suppose $x_1 + x_{2q+1} \geq 1$. Then $x_2 + \cdots + x_{2q} < \frac{2q-1}{2}$. By induction this yields $q - 1$ blocking pairs. Note that $x_2 < \frac{1}{2}$ holds. Hence $x_1 + x_2 < 1$, and we have at least q blocking pairs. □

Theorem 7. Blocking Pairs *is* NP*-complete, even for simple matching games.*

Proof. Clearly, this problem is in NP. To prove NP-completeness, we reduce from Independent Set, which tests if a graph G contains an *independent* set of size at least k, i.e., a set U (with $|U| \geq k$) such that there is no edge in G between any two vertices of U. Garey, Johnson and Stockmeyer [10] show that the Independent Set problem is already NP-complete for the class of *3-regular* connected graphs, i.e., graphs, in which all vertices are of degree three. So, we may assume that G is 3-regular and connected. Let $n = |V|$. We may without loss of generality assume that $k \leq \frac{1}{2}n$; otherwise G does not have an independent set of size at least k.

From G we construct the following graph. First, we introduce a set Y of np new vertices for some integer p, the value of which we will determine later. We denote the vertices in Y by $y_1^u, \ldots, y_p^u$ for each $u \in V$. We connect each y_i^u (only) to its associated vertex u. This yields a graph G^*, in which all vertices of G now have degree $3 + p$, and all vertices of Y have degree one. Second, let K be a complete graph on ℓ vertices, where ℓ is some *odd* integer larger than np, the value of which will be made clear later on. We add $2(n-k)$ copies $K^1, \ldots, K^{2(n-k)}$ of K to G^* without introducing any further edges. The resulting graph consists of $2(n-k)+1$ components and is denoted by $G' = (N, E')$. We denote the simple matching game on G' by (N, v). We observe that $\{uy_1^u \mid u \in V\}$ is a maximum matching in G^* of size n. Because of this and because ℓ is odd, we obtain that $v(N) = \frac{1}{2}(\ell-1)2(n-k)+n = \ell(n-k)+k$. We show that the following statements are equivalent for suitable choices of ℓ and p, hereby proving Theorem 7.

(i) G has an independent set U of size $|U| \geq k$.
(ii) $|B(x)| \leq (n-k)p + \frac{3}{2}n - 3k$ for some imputation x of (N, v).

"(i) ⇒ (ii)" Suppose G has an independent set U of size $|U| \geq k$. We define an imputation x as follows, $x \equiv \frac{1}{2}$ on each K^h, $x \equiv 1$ on U' for some subset $U' \subseteq U$ of size $|U'| = k$ and $x \equiv 0$ otherwise. Then the set of blocking pairs is

$$\{(u, y_i^u) \mid u \in V \backslash U', 1 \leq i \leq p\} \cup \{(u, v) \mid u, v \in V \backslash U' \text{ and } uv \in E\}.$$

We observe that $|\{(u, y_i^u) \mid u \in V \backslash U', 1 \leq i \leq p\}| = (n-k)p$. Furthermore, because G is 3-regular, $U' \subseteq U$ is an independent set and $k \leq \frac{1}{2}n$, we find that $|\{(u,v) \mid u, v \in V \backslash U'\}| = |E| - 3k = \frac{3}{2}n - 3k \geq 0$. Hence, $|B(x)| = (n-k)p + \frac{3}{2}n - 3k$.

"(ii) ⇒ (i)" Suppose $|B(x)| \leq (n-k)p + \frac{3}{2}n - 3k$ for some imputation x of (N, v). We may without loss of generality assume that x has minimum number of blocking pairs. We first prove a number of claims.

Claim 1. We may without loss of generality assume that $x_i \leq 1$ for all $i \in N$.

We prove Claim 1 as follows. Suppose $x_i = 1 + \alpha$ for some $\alpha > 0$ for some $i \in N$. We set $x_i := 1$ and redistribute α over all vertices $j \in N \backslash Y$ with $x_j < 1$. When doing this we ensure that we do not increase the value of some x_j with more than $1 - x_j$. This is possible, since $x(N) = v(N) = \ell(n-k) + k < 2\ell(n-k) + np + n = |N|$. The resulting allocation would be an imputation with a smaller or equal number of blocking pairs. This proves Claim 1.

Claim 2. We may without loss of generality assume that $x_y = 0$ for each $y \in Y$.

We prove Claim 2 as follows. Suppose $x_y > 0$ for some $y \in Y$. Let u be the (unique) neighbor of y. We set $x_y := 0$ and $x_u := \min\{x_u + x_y, 1\}$. If necessary, we redistribute the remainder over $V \cup \bigcup_j K^j$ without violating Claim 1. This is possible, since $x(N) = \ell(n-k) + k < 2\ell(n-k) + n = |\bigcup_j K^j| + |V|$. The resulting imputation would have a smaller or equal number of blocking pairs. This proves Claim 2.

Claim 3. $x(\bigcup_j K^j) = \ell(n-k)$.

We prove Claim 3 as follows. First suppose $x(\bigcup_j K^j) > \ell(n-k)$. Then we set $x_i := \frac{1}{2}$ for each $i \in \bigcup_j K^j$ and redistribute the remainder over V without violating Claim 1. This is possible, since after setting $x_i := \frac{1}{2}$ for each $i \in \bigcup_j K^j$, we have $x(N) - x(\bigcup_j K^j) = \ell(n-k) + k - \ell(n-k) = k \leq n = |V|$. The resulting imputation would have a smaller or equal number of blocking pairs. Hence, we may assume that $x(\bigcup_j K^j) \leq \ell(n-k)$ holds.

Suppose $x(\bigcup_j K^j) < \ell(n-k)$. Then there is some K^j with $x(K^j) < \frac{\ell}{2}$. By Lemma 6, there are at least $\frac{\ell-1}{2}$ blocking pairs in K^j. We choose $\ell = 2np + 2|E| + 2$ and obtain $|B(x)| \geq \frac{\ell-1}{2} > (n-k)p + |E|$. However, let x^* be given by $x^* \equiv \frac{1}{2}$ on $\bigcup_j K^j$, $x^* \equiv 0$ on Y, $x^* \equiv 1$ on some $U \subseteq V$ of size $|U| = k$ and $x^* \equiv 0$ on $V \backslash U$. Then x^* is an imputation as $x_i^* \geq 0$ for all $i \in N$ and $x^*(N) = \ell(n-k) + k = v(N)$. We observe that $|B(x^*)| < (n-k)p + |E|$. Hence x is not an imputation with minimum number of blocking pairs. This proves Claim 3.

We now continue with the proof. Combining Claims 2 and 3 leads to

$$x(V) = x(N) - x\left(\bigcup_{j=1}^{2(n-k)} K^j\right) - x(Y) = v(N) - \ell(n-k) = \ell(n-k) + k - \ell(n-k) = k.$$

Let R be the set of vertices v in G with $x_v < 1$. We first show that $|R| \leq n-k$. Suppose $|R| \geq n-k+1$. Since $x(Y) = 0$ due to Claim 2, we find that $(n-k)p + \frac{3}{2}n - 3k \geq |B(x)| \geq |R|p \geq (n-k)p + p$. This is not possible if we choose $p = 2n$. Hence, indeed $|R| \leq n-k$ holds.

Let U consist of all vertices $u \in V$ with $x_u = 1$. Note that $U = V \backslash R$ due to Claim 1. Since $x(V) = k$ as we deduced above, we find that $|U| \leq k$, and thus $|R| \geq n-k$. As we already deduced that $|R| \leq n-k$, we find that $|R| = n-k$, and consequently, $|U| = k$. The latter equality implies that $x_v = 0$ for all $v \in R$.

Because G is 3-regular, G has $\frac{3n}{2}$ edges. Then $B(x) \geq (n-k)p+\frac{3n}{2}-3|U|$, with equality only if U is an independent set. Equality must hold since we assume that $B(x) \leq (n-k)p+\frac{3n}{2}-3k$ and $|U|=k$. Hence, indeed U is an independent set of size k. This completes the proof of Theorem 7. □

5 Future Research

Beside the notorious problem of determining the complexity of computing the nucleolus of matching games with empty core, we would like to mention one other open problem. Let x be an imputation of a matching game (N, v) defined on a graph $G = (N, E)$ with edge weighting w. We say that a pair of adjacent vertices u, v has *blocking value* $e_x(i,j)^+ = \max\{0, w_{i,j} - (x_i + x_j)\}$. We let $b(x) = \sum_{ij \in E} e_x(i,j)^+$ be the *total blocking value* of x. The complexity status of the problem that asks if a matching game (N, v) has an imputation with total blocking value at most k, where k is a positive integer part of the input, is open.

References

1. Abraham, D.J., Biró, P., Manlove, D.F.: "Almost stable" matchings in the Roommates problem. In: Erlebach, T., Persinao, G. (eds.) WAOA 2005. LNCS, vol. 3879, pp. 1–14. Springer, Heidelberg (2006)
2. Balinski, M.L.: Integer programming: Methods, uses, computation. Management Science 12, 253–313 (1965)
3. Bilbao, J.M.: Cooperative games on combinatorial structures. Kluwer Academic, Norwell (2000)
4. Biró, P.: The stable matching problem and its generalizations: an algorithmic and game theoretical approach, PhD Thesis, Budapest University of Technology and Economics, Budapest, Hungary (2007)
5. Deng, X., Ibaraki, T., Nagamochi, H.: Algorithmic aspects of the core of combinatorial optimization games. Math. Oper. Res. 24, 751–766 (1999)
6. Egerváry, J.: Matrixok kombinatorius tulajdonságairól. Matematikai és Fizikai Lapok 38, 16–28 (1931)
7. Eriksson, K., Karlander, J.: Stable outcomes of the roommate game with transferable utility. Internat. J. Game Theory 29, 555–569 (2001)
8. Faigle, U., Kern, W., Fekete, S., Hochstättler, W.: The nucleon of cooperative games and an algorithm for matching games. Math. Program. 83, 195–211 (1998)
9. Gabow, H.N.: Data structures for weighted matching and nearest common ancestors with linking. In: Proceedings of SODA 1990, pp. 434–443 (1990)
10. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some simplified NP-complete graph problems. Theoret. Comput. Sci. 1, 237–267 (1976)
11. Kern, W., Paulusma, D.: Matching games: the least core and the nucleolus. Math. Oper. Res. 28, 294–308 (2003)
12. Khachiyan, L.G.: A polynomial algorithm in linear programming. Soviet Mathematics Doklady 20, 191–194 (1979)
13. Klaus, B., Nichifor, A.: Consistency and monotonicity in one-sided assignment problems (2009) (preprint)
14. Maschler, M., Peleg, B., Shapley, L.S.: Geometric properties of the kernel, nucleolus, and related solution concepts. Math. Oper. Res. 4, 303–338 (1979)

15. Matsui, T.: A note on the nucleolus of assignment games. In: 5th International Conference on Nonlinear Analysis and Convex Analysis, pp. 253–260. World Scientific, Singapore (1998)
16. Micali, S., Vazirani, V.V.: An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In: Proceedings of FOCS 1980, pp. 17–27 (1980)
17. Nemhauser, G.L., Trotter, L.E.: Vertex packings: structural properties and algorithms. Math. Program. 8, 232–248 (1975)
18. Owen, G.: Game theory. Academic Press, San Diego (1995)
19. Paulusma, D.: Complexity aspects of cooperative games, PhD Thesis, University of Twente, Enschede, the Netherlands (2001)
20. Schmeidler, D.: The nucleolus of a characteristic function game. SIAM J. Appl. Math. 17, 1163–1170 (1969)
21. Shapley, L.S., Shubik, M.: The assignment game I: the core. Internat. J. Game Theory 1, 111–130 (1972)
22. Schrijver, A.: Combinatorial optimization. In: Polyhedra and efficiency. Algorithms and Combinatorics 24, vol. A. Springer, Berlin (2003)
23. Solymosi, T., Raghavan, T.E.S.: An algorithm for finding the nucleolus of assignment games. Internat. J. Game Theory 23, 119–143 (1994)

Binary De Bruijn Partial Words with One Hole*

Francine Blanchet-Sadri[1], Jarett Schwartz[2], Slater Stich[3], and Benjamin J. Wyatt[1]

[1] Department of Computer Science, University of North Carolina, P.O. Box 26170, Greensboro, NC 27402–6170, USA
`blanchet@uncg.edu`
[2] Department of Computer Science, Princeton University, 35 Olden Street, Princeton, NJ 08540–5233, USA
[3] Department of Mathematics, Princeton University, Fine Hall, Washington Road, Princeton, NJ 08544–1000, USA

Abstract. In this paper, we investigate partial words, or finite sequences that may have some undefined positions called holes, of maximum subword complexity. The subword complexity function of a partial word w over a given alphabet of size k assigns to each positive integer n, the number $p_w(n)$ of distinct full words over the alphabet that are *compatible* with factors of length n of w. For positive integers n, h and k, we introduce the concept of a de Bruijn partial word of order n with h holes over an alphabet A of size k, as being a partial word w with h holes over A of minimal length with the property that $p_w(n) = k^n$. We are concerned with the following three questions: (1) What is the length of k-ary de Bruijn partial words of order n with h holes? (2) What is an efficient method for generating such partial words? (3) How many such partial words are there? *Keywords*: Combinatorics on words; Partial words; Subword complexity; De Bruijn sequences; De Bruijn graphs; Eulerian paths; combinatorial problems.

1 Introduction

Let A be a k-letter alphabet and w be a finite or right infinite word over A. A *subword* or *factor* of w is a block of consecutive letters of w. The subword complexity of w is the function that assigns to each positive integer, n, the number, $p_w(n)$, of distinct subwords of length n of w. The subword complexity, also called symbolic complexity, of finite and infinite words has become an important subject in combinatorics on words. Application areas include dynamical systems, ergodic theory, and theoretical computer science. We refer the reader to Chapter 10 of [1] which surveys and discusses subword complexity of finite and infinite words. References [2] and [3] provide other surveys, [4] shows how the so-called

* This material is based upon work supported by the National Science Foundation under Grant No. DMS–0754154. The Department of Defense is also gratefully acknowledged. We thank the referees of preliminary versions of this paper for their very valuable comments and suggestions.

© Springer-Verlag Berlin Heidelberg 2010

special and bispecial factors can be used to compute the subword complexity, and [5] gives another interesting approach based on the gap function.

When we restrict our attention to finite words of maximum subword complexity, de Bruijn sequences play an important role. A k-ary de Bruijn sequence of order n is a word over an alphabet of size k where each of the k^n words of length n over the alphabet appears as a factor exactly once. It is well known that such sequences have length $k^n + n - 1$. There are $k!^{k^{n-1}}$ of them, and they can be efficiently generated by constructing Eulerian cycles in corresponding de Bruijn directed graphs. The technical report of de Bruijn provides an history on the existence of these sequences [6]. De Bruijn graphs find applications, in particular, in genome rearrangements [7], in the complexity of deciding avoidability of sets of partial words [8], etc.

In this paper, we investigate partial words of maximum subword complexity. Partial words are finite sequences that may have some undefined positions called holes (a (full) word is just a partial word without holes). Partial words can be viewed as sequences over an extended alphabet $A_\diamond = A \cup \{\diamond\}$, where $\diamond \notin A$ stands for a hole. Here $\diamond$ matches every letter in the alphabet, or is compatible with every letter in the alphabet. For example, $10\diamond01$ is a partial word with one hole over the alphabet $\{0, 1\}$. In this context, $p_w(n)$ is the number of distinct full words over the alphabet that are compatible with factors of length n of the partial word w (in our example with $w = 10\diamond01$, we have $p_w(3) = 5$ since $000, 001, 010, 100$ and 101 match factors of length 3 of w). For positive integers n, h and k, we introduce the concept of a de Bruijn partial word of order n with h holes over an alphabet A of size k, as being a partial word w with h holes over A of minimal length with the property that $p_w(n) = k^n$.

The contents of our paper is as follows: In Section 2, we review some concepts on partial words. In Section 3, we give lower and upper bounds on the length of k-ary de Bruijn partial words with h holes of order n, and show that our bounds are tight when $h = 1$. In Section 4, we provide an algorithm to construct de Bruijn binary partial words with one hole. Finally in Section 5, we show how to count such partial words by adapting the so-called BEST theorem that counts the number of Eulerian cycles in directed graphs.

2 Preliminaries

For more background on partial words, we refer the reader to [9].

Let A be a fixed non-empty finite set called an *alphabet* whose elements we call *letters*. A *word* over A is a finite sequence of elements from A. We let A^* denote the set of words over A which, under the concatenation operation of words, forms a free monoid whose identity is the empty word, which we denote by ε. Unless otherwise stated, we assume that A contains at least two letters.

A *partial word* w of length n over A can be defined as a function $w : [0..n-1] \to A_\diamond$, where $A_\diamond = A \cup \{\diamond\}$ with $\diamond \notin A$. The length of w is denoted by $|w|$, and $w(i)$, the symbol at position i, is denoted by w_i (here $[0..n-1]$ denotes the set of positions $\{0, 1, \ldots, n-1\}$). For $0 \leq i < n$, if $w(i) \in A$, then i belongs to the *domain* of w, denoted $D(w)$, and if $w(i) = \diamond$, then i belongs to the *set of holes* of w, denoted $H(w)$. Whenever $H(w)$ is empty, w is a *full word*. We refer to an occurrence of the symbol $\diamond$ as a *hole*. We let $A_\diamond^*$ denote the set of all partial words over A.

A partial word u is a *factor* of the partial word v if there exist x, y such that $v = xuy$. The factor u is called *proper* if $u \neq \varepsilon$ and $u \neq v$. The partial word u is a *prefix* (respectively, *suffix*) of v if $x = \varepsilon$ (respectively, $y = \varepsilon$).

The partial word u is *contained* in the partial word v, denoted $u \subset v$, if $|u| = |v|$ and $u(i) = v(i)$ for all $i \in D(u)$. Two partial words u and v of equal length are *compatible*, denoted $u \uparrow v$, if $u(i) = v(i)$ whenever $i \in D(u) \cap D(v)$. In other words, u and v are compatible if there exists a partial word w such that $u \subset w$ and $v \subset w$, in which case we let $u \vee v$ denote the *least upper bound* of u and v ($u \subset (u \vee v)$ and $v \subset (u \vee v)$ and $D(u \vee v) = D(u) \cup D(v)$). For example, $u = aba\diamond\diamond$ and $v = a\diamond\diamond b\diamond$ are compatible, and $(u \vee v) = abab\diamond$.

A full word u is a *subword* of w if there exists some $0 \leq i < |w| - |u|$ such that $u \uparrow w(i) \cdots w(i + |u| - 1)$. Informally, under some "filling in" of the holes in w with letters from A to form the full word w', there is some consecutive block of letters in w', $w'(i) \cdots w'(i + |u| - 1)$, such that $w'(i) = u(0), w'(i + 1) = u(1)$, and so on. Note that in this paper, subwords are always full.

A completion $\hat{w}$ of a partial word w over A is a function $\hat{w} : [0..|w| - 1] \to A$ such that $\hat{w}(i) = w(i)$ if $w(i) \neq \diamond$. A completion $\hat{w}$ is usually thought of as a "filling in" of the holes of w with letters from A. Note that two partial words u and v are compatible if there exist completions $\hat{u}$ and $\hat{v}$ such that $\hat{u} = \hat{v}$. The subword complexity of w is the function that assigns to each integer, $0 \leq n \leq |w|$, the number, $p_w(n)$, of distinct full words over A that are compatible with factors of length n of w (or the number of distinct subwords of w of length n). We let $\text{Sub}_w(n)$ denote the set of all subwords of w of length n, and we let $\text{Sub}(w) = \bigcup_{0 \leq n \leq |w|} \text{Sub}_w(n)$ the set of all subwords of w. Note that if $\hat{w}$ is a completion of w, then $p_{\hat{w}}(n) \leq p_w(n)$, since $\text{Sub}_{\hat{w}}(n) \subset \text{Sub}_w(n)$.

3 Bounds on the Length of de Bruijn Partial Words

What is the length of a shortest word w over an alphabet of size k for which $p_w(n) = k^n$, where n is a positive integer?

Theorem 1 ([1]). *For all $k, n \geq 1$ there exists a word w over an alphabet of size k, of length $k^n + n - 1$, such that $p_w(n) = k^n$.*

Such a word is often called a *k-ary de Bruijn full word of order n*, that is, a full word over a given alphabet A with size k for which every possible word of length n over A appears as a subword exactly once. De Bruijn words are often "cyclic" in the literature, meaning that subwords can wrap around from the end to the beginning of the word, but to better fit our notion of the complexity function, we unwrap them and use a non-cyclic version.

In order to prove the theorem, set $A = \{0, 1, \ldots, k-1\}$. If $k = 1$, then take 0^n, while if $n = 1$, take $01 \cdots (k - 1)$. If $k, n \geq 2$, a family of directed graphs $G_k(n)$

is defined as follows: the vertices of $G_k(n)$ are the words of length $n-1$ over A, and the edges of $G_k(n)$ are the pairs (az, zb), labelled by azb, where $a, b \in A$ and z is a word of length $n-2$ over A. It then suffices to show that $G_k(n)$ possesses an Eulerian cycle, that is, a path that traverses every edge exactly once and begins and ends at the same vertex. Indeed, $G_k(n)$ is strongly connected, that is, there is a directed path connecting any two vertices, and the indegree of each vertex equals its outdegree. A directed graph that possesses an Eulerian cycle is called an Eulerian digraph. Note that there are several linear-time algorithms, including Fleury's algorithm, for computing Eulerian cycles in digraphs [10].

We define a k-ary de Bruijn partial word with h holes of order n, to be a partial word of minimal length with h holes over an alphabet A of size k with all k^n words of length n over A being compatible with factors of it. A main question is to determine the length of k-ary de Bruijn partial words with h holes of order n. For example, 00110 is a 2-ary de Bruijn full word of order 2, which has length 5, while a 2-ary de Bruijn partial word of order 2 with one hole is $001\diamond$, which has length 4. We let $L_k(n, h)$ denote the length of a k-ary de Bruijn partial word of order n with h holes.

Definition 1. – *Let $M_z(n)$ denote the number of distinct completions of factors of length n with at least one hole of a partial word z.*
– *Let $M_k(n, h) = \max_z M_z(n)$ where the maximum is taken over all partial words z with h holes over an alphabet of size k.*

It is clear that for $n \leq h$, if z is a word with h holes, n of them being consecutive, over an alphabet of size k, then $M_z(n) = k^n$ and since k^n is the total number of words of length n over a k-letter alphabet, we have $M_k(n, h) = k^n$. We can significantly refine the upper bound of k^n on $M_k(n, h)$, when $n > h$, as stated in the next theorem.

Theorem 2. *For $k \geq 2$ and $n > h > 0$, $M_k(n, h) \leq (n-h+1)k^h + 2\frac{k^h-k}{k-1}$.*

Proof. Let z be a word with h holes over a k-letter alphabet. First, note that if the h holes of z are consecutive, or $\diamond^h$ is a factor of z, then there may be factors of length n of z that contain only the first hole (respectively, the last hole), the first two holes (respectively, the last two holes), and so on, until may be factors that contain only the first $h-1$ holes (respectively, the last $h-1$ holes), and then factors that contain all of the h holes. Note that i consecutive holes can contribute a maximum of k^i distinct completions. So, in total, z can have up to $(n-h+1)k^h + 2\sum_{i=1}^{h-1} k^i = (n-h+1)k^h + 2\frac{k^h-k}{k-1}$ distinct completions of factors of length n containing at least one hole. Now, assume that $\diamond^{h-r}$ and $\diamond^r$ are two disjoint factors of z, where $0 < r < h$. In this case, $M_z(n)$ cannot be bigger than the bound above. So if we keep splitting up the holes, we do not change our bound. □

Corollary 1. *For $k \geq 2$, $n \geq 2h+2$ and $h > 0$, we have*

$$M_k(n, h) = (n-h+1)k^h + 2\frac{k^h-k}{k-1}.$$

Proof. By Theorem 2, $M_k(n,h) \leq (n-h+1)k^h + 2\frac{k^h-k}{k-1}$. To show that $M_k(n,h) \geq (n-h+1)k^h + 2\frac{k^h-k}{k-1}$, we only need find a partial word $z_{n,h}$ with h holes over a k-letter alphabet such that $M_{z_{n,h}}(n) = (n-h+1)k^h + 2\frac{k^h-k}{k-1}$. Consider $z_{n,h} = b^n a \diamond^h a b^n$ where a, b are distinct letters of the alphabet. The factors of length n of $z_{n,h}$ with at least one hole are

- $b^{n-2}a\diamond, \ldots, b^{n-h}a\diamond^{h-1}$, as well as their reversals: the number of distinct completions of these factors is $2\frac{k^h-k}{k-1}$.
- $b^{n-h-1}a\diamond^h, \ldots, ba\diamond^h ab^{n-h-3}, a\diamond^h ab^{n-h-2}, \diamond^h ab^{n-h-1}$: the number of distinct completions of these factors is $(n-h-1+1+1)k^h = (n-h+1)k^h$.

Note that the words of length n compatible with these factors are distinct, since the factors starting at the first $n-1$ positions are distinct from each other, because they start with a different number of b's, and distinct from the rest, because they have an a at most h positions from the beginning. The factors ending at the last $n-1$ positions are also distinct because they end with different numbers of b's. □

Remark 1. Corollary 1 fails for $n < 2h+2$. In the contruction of the proof of Corollary 1, $z_{5,2} = b^5 a \diamond^2 a b^5$, and so $M_z(5) = 19 \neq 20 = (n-h+1)k^h + 2\frac{k^h-k}{k-1}$. Here, the factor $b^2a\diamond^2$ is compatible with the factor $\diamond^2ab^2$, and so the completion b^2ab^2 gets counted twice.

Theorem 3. *For $h > 0$, $L_k(n,h) \geq L_k(n,0) - M_k(n,h) + (n+h-1)$.*

Proof. A k-ary de Bruijn full word of order n contains each subword of length n exactly once. When considering partial words with h holes over an alphabet of size k, we are still limited to at most one distinct factor of length n per starting symbol, except we can get more than one distinct completion for factors with at least one hole. The number of such completions is at most $M_k(n,h)$, but this includes $(n+h-1)$ starting positions that lead to distinct subwords in a de Bruijn full word. So, in total we have $L_k(n,h) \geq L_k(n,0) - M_k(n,h) + (n+h-1)$. □

Corollary 2. *For $n \geq 2h+2$ and $h > 0$, $L_2(n,h) \geq 2^n+2n+h+2-(n-h+3)2^h$.*

Proof. We know that 2-ary de Bruijn full words of order n have length 2^n+n-1. Furthermore, from Theorem 2, Corollary 1 and Theorem 3, we get

$$\begin{aligned} L_2(n,h) &\geq 2^n + n - 1 - (2^h(n-h+3) - 4) + (n+h-1) \\ &= 2^n + 2n + h + 2 - (n-h+3)2^h \end{aligned}$$
□

In Section 4, we show that the bound of Corollary 2 is tight for $h = 1$, that is, $L_2(n,1) = 2^n + 2n + h + 2 - (n-h+3)2^h = 2^n - 1$ for $n \geq 4$.

4 Constructing de Bruijn Partial Words

We can construct k-ary de Bruijn full words of order n by finding Eulerian cycles in $G_k(n)$. In this section, we describe an algorithm to construct 2-ary de Bruijn partial words of order n with one hole by finding Eulerian paths in a trimmed version of $G_2(n)$. We also discuss the $k = 3$ case.

We first recall the conditions for a directed graph $G = (V, E)$ to have an (x, y)-Eulerian path, that is, an Eulerian path from vertex x to vertex y. Let $\mathtt{ideg}(v)$ (respectively, $\mathtt{odeg}(v)$) denote the indegree (respectively, outdegree) of vertex $v \in V$.

Lemma 1. *Let $G = (V, E)$ be a digraph, and let $x, y \in V$ be such that $\mathtt{odeg}(x) = 1 + \mathtt{ideg}(x)$ and $\mathtt{ideg}(y) = 1 + \mathtt{odeg}(y)$. Then G has an (x, y)-Eulerian path if and only if G has at most one non-trivial connected component containing x, y and for every vertex $v \in V \setminus \{x, y\}$, $\mathtt{ideg}(v) = \mathtt{odeg}(v)$.*

We now modify the Eulerian cycle approach to prove that our bounds are tight in the binary one hole case.

Theorem 4. *For $n \geq 4$, we have $L_2(n, 1) = 2^n - 1$.*

Proof. Start with the digraph $G = G_2(n)$. Let $z = x\diamond y = 1^{n-2}0\diamond 0^{n-2}1$. It can be checked that $M_z(n) = 2n = M_2(n, 1)$, that is, z has $2n$ distinct subwords of length n. Trim $G_2(n)$ by deleting all edges that are in $\mathrm{Sub}_z(n)$. Then, add a new edge from vertex x to vertex y labelled by z. Call the resulting graph, $G' = (V, E)$. First, consider any factor of length $n - 1$ with a hole in z. Then, choose a completion, v, of that factor. Thus, v is a prefix of some $v_1 \in \mathrm{Sub}_z(n)$ and a suffix of some $v_2 \in \mathrm{Sub}_z(n)$. So, both $\mathtt{ideg}(v)$ and $\mathtt{odeg}(v)$ get decreased by one, but v remains balanced. The only vertices that become isolated are 0^{n-1} and 10^{n-2}. Now, consider the factors $x = 1^{n-2}0$ and $y = 0^{n-2}1$. Here, x is a prefix of two subwords of length n, namely the two completions $1^{n-2}00$ and $1^{n-2}01$. So, two edges starting at x are deleted from G, while the edge starting at x, labelled by z, is added to G. Similarly, y is a suffix of two subwords of length n, the two completions $00^{n-2}1$ and $10^{n-2}1$. So, two edges ending at y are deleted from G, while the edge ending at y, labelled by z, is added to G.

So the graph G' satisfies the following conditions: (1) G' has a single non-trivial connected component; (2) $\mathtt{odeg}(y) = 1 + \mathtt{ideg}(y)$ and $\mathtt{ideg}(x) = 1 + \mathtt{odeg}(x)$; and (3) for every vertex $v \in V \setminus \{x, y\}$, $\mathtt{ideg}(v) = \mathtt{odeg}(v)$. By Lemma 1, G' has an Eulerian path from vertex y to vertex x. Since z has the maximum number, $M_2(n, 1)$, of distinct subwords of length n, we get $L_2(n, 0) = 2^n + n - 1$ implies $L_2(n, 1) = 2^n - M_2(n, 1) + n - 1 + (|y| + 1)$, and so $L_2(n, 1) = 2^n - 2n + n - 1 + n = 2^n - 1$ as desired. □

Example 1. Computer experiments show that there are seven z's (up to a renaming of letters) with one hole over the binary alphabet $\{0, 1\}$ such that $M_z(4) = M_2(4, 1) = 8$. They are $110\diamond110, 110\diamond011, 110\diamond001, 101\diamond001, 100\diamond101, 100\diamond100$ and $100\diamond011$. From the proof of Theorem 4, if we choose $z_1 = 110\diamond001$, then there is an Eulerian path in the resulting graph from vertex 001 to vertex 110. But, if we consider $z_2 = 110\diamond110$ instead, we note that $1110, 0111 \in \mathrm{Sub}(z_2)$ but $1111 \notin \mathrm{Sub}(z_2)$. So the vertex 111 becomes isolated with the loop labelled by 1111 (see the graph on the right in Figure 1). Therefore, the resulting graph does not have an Eulerian path. There are fourteen z's that satisfy $M_z(4) = 8$, but only four of them generate graphs that have an Eulerian path (110⋄001, its reversal, and their renamings).

What we need is to start with a word z that is *good* in the sense that if $\{10^{n-1}, 0^{n-1}1\} \subset \text{Sub}_z(n)$ then $0^n \in \text{Sub}_z(n)$, and if $\{01^{n-1}, 1^{n-1}0\} \subset \text{Sub}_z(n)$ then $1^n \in \text{Sub}_z(n)$, otherwise 0^{n-1} or 1^{n-1} would become isolated with loop 0^n or 1^n, respectively. Table 1 gives data on the number of z's over the alphabet $\{0, 1\}$ such that $M_z(n) = 2n$ versus the number of such z's that are good.

Table 1. Number of good z's over $\{0, 1\}$ for $4 \leq n \leq 8$

n	Number of z's such that $M_z(n) = 2n$	Number of good z's
4	14	4
5	98	10
6	546	40
7	2768	96
8	12832	272

After applying the algorithm described in the proof of Theorem 4 (see Algorithm 1), we get a 2-ary de Bruijn partial word of order n of length $2^n - 1$ with one hole. We let $G_2(n, z)$ denote the graph built by Algorithm 1.

Algorithm 1. Constructing a 2-ary de Bruijn word of order n with one hole, where $n \geq 4$

```
1: Build $G = G_2(n)$
2: Select a good word $z = x\diamond y$ with $|x| = |y| = n - 1$ and $M_z(n) = 2n$
3: Compute $S = \text{Sub}_z(n)$
4: Create graph $G'$ from $G$ by deleting the edges in the set $S$ along with any resulting
   isolated vertices, and add an edge from vertex $x$ to vertex $y$ labelled by $z$
5: Find an Eulerian path $p$ in $G'$ from $y$ to $x$
6: return $p$
```

Example 2. For $k = 2$ and $n = 4$, if we select $z_1 = 110\diamond001$ then Algorithm 1 produces the graph on the left in Figure 1. The 2-ary word $w = 0010110\diamond0011110$ of length $2^4 - 1 = 15$ is such that $p_w(4) = 2^4 = 16$. It can be checked that

$$001 \xrightarrow{0010} 010 \xrightarrow{0101} 101 \xrightarrow{1011} 011 \xrightarrow{0110} 110 \xrightarrow{110\diamond001} 001$$
$$\xrightarrow{0011} 011 \xrightarrow{0111} 111 \xrightarrow{1111} 111 \xrightarrow{1110} 110$$

is an Eulerian path from $y = 001$ to $x = 110$ in the trimmed graph $G_2(4, z_1)$.

The difficulty in building a de Bruijn partial word for $k = 3$, $n = 4$, and $h = 1$ for instance, is that we have to compensate for the indegree and outdegree of the nodes connected by the edge labelled by a word with one hole. When working

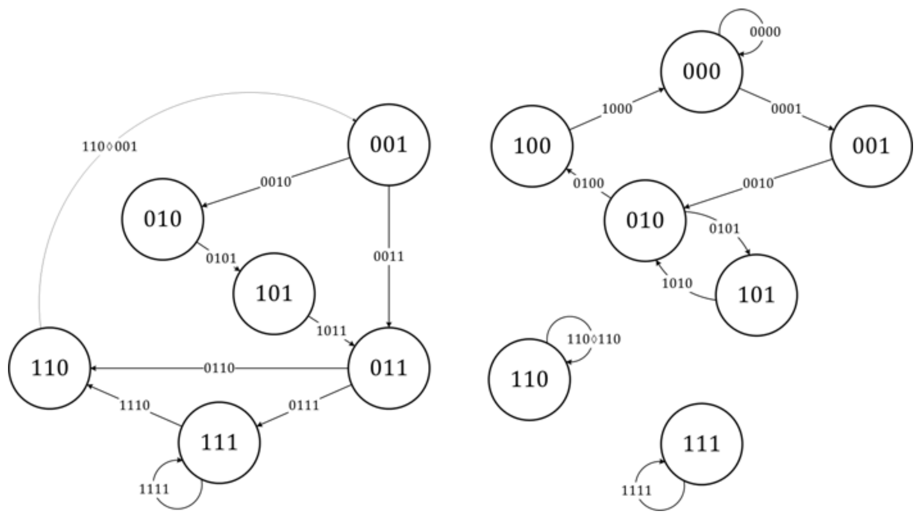


Fig. 1. Left: Non-trivial connected component of $G_2(4, 110\diamond001)$; Right: Non-trivial connected components of $G_2(4, 110\diamond110)$

with a good word $z = x\diamond y$, thus having the maximum number $kn = 3n$ of subwords of length n, a “schism” is created in which $\texttt{ideg}(x) = 3$ and $\texttt{odeg}(x) = 1$, while $\texttt{ideg}(y) = 1$ and $\texttt{odeg}(y) = 3$. For example, if we take $z = 110\diamond001$, the node 110 will now have oudegree 1 because of the edge labelled by z, and indegree 3, while 001 only has indegree 1 but will have outdegree 3. With $k = 2$, we compensate for this schism by starting the de Bruijn partial word from y (that has outdegree 2 and indegree 1), and by ending with x (that has indegree 2 and outdegree 1). Since each vertex, other than x and y are balanced, this effectively “skips” the problem entirely. When we try to compensate in a similar fashion for a 3-letter alphabet, we end up having to add an extra edge.

To produce a de Bruijn partial word in which a single subword occurs twice, use a good word of the form $x\diamond x$. For example, using $102\diamond102$ eliminates all edges to and away from the node 102. This removes the issue of compensating for an unbalanced vertex: each vertex has equal indegree and outdegree (note that $\texttt{ideg}(102) = \texttt{odeg}(102) = 1$ due to the edge $102\diamond102$ from 102 to 102). However, the vertex 102 becomes isolated. Since all edges from 102 have been deleted, an additional edge is required to connect 102 to the rest of the graph. Here, use the edge $(102, 022)$ labelled by 1022 for instance. This process can be generalized to arbitrary n, and so we get the following result.

Theorem 5. *For $n \geq 2$, we have $L_3(n, 1) = 3^n - n$.*

Proof. The equality $L_3(n, 0) = 3^n + n - 1$ implies $L_3(n, 1) = (3^n - M_3(n, 1) + |z|) + 1$ (for an extra edge), and so $L_3(n, 1) = 3^n - 3n + 2n - 1 + 1 = 3^n - n$. □

Example 3. The partial word $u = 10\diamond 100222122020121112000$ is a 3-ary de Bruijn partial word of length 24 with one hole of order $n = 3$, while

$$v = 102\diamond 1022221222022012211220021212021012 1\\ 11121002020012011200001110110010100022$$

of length 77 is one of order $n = 4$. Note that u has the subword 100 occurring twice, while v has the subword 1022 occurring twice as explained above.

5 Counting De Bruijn Partial Words

Another main question is to compute the number of k-ary de Bruijn partial words with h holes of order n, which we denote by $N_k(n, h)$. It is well known that $N_k(n, 0) = k!^{k^{n-1}}$, which can be calculated by counting the number of Eulerian cycles in $G_k(n)$. This can be done by using the so-called BEST theorem, named after de Bruijn, van Aardenne-Ehrenfest, Smith and Tutte, that counts the number of Eulerian cycles in directed graphs.

Theorem 6 ([11]). *Let $G = (V, E)$ be an Eulerian digraph, and let L_G denote the Laplacian matrix of G defined as follows: for $i = j$, $L_G(i, j) = \mathtt{odeg}(v_i) - e$, and for $i \neq j$, $L_G(i, j) = -e$, where e is the number of edges from v_i to v_j. Then the number of non-equivalent Eulerian cycles in G is*

$$C \prod_{v \in V} (\mathtt{odeg}(v) - 1)! = C \prod_{v \in V} (\mathtt{ideg}(v) - 1)! \qquad (1)$$

with C any cofactor of L_G.

To compute $N_2(n, 1)$, we need to modify Theorem 6, since we want to count the number of Eulerian paths.

Theorem 7. *Let $G = (V, E)$ be a digraph, and let $x, y \in V$ be such that $\mathtt{odeg}(x) = 1 + \mathtt{ideg}(x)$ and $\mathtt{ideg}(y) = 1 + \mathtt{odeg}(y)$. Suppose that G satisfies the conditions of Lemma 1 to have an (x, y)-Eulerian path. Let L_G denote the Laplacian matrix of G defined as above. Then the number of (x, y)-Eulerian paths in G is given by (1) with C the cofactor of L_G with the row and column corresponding to vertex y removed.*

With 2-ary de Bruijn partial words of order n with one hole, as mentioned in Section 4, we need to apply Theorem 6 to more than one graph since every word z of length $2n - 1$, with a hole in the middle and such that $M_z(n) = M_2(n, 1) = 2n$, can potentially serve as the new edge added to the graph $G_2(n)$. But after deleting the edges corresponding to subwords of length n of z, we do not necessarily have an Eulerian path, so we must only count those paths in the $G_2(n, z)$'s, where z is good. This suggests an algorithm, Algorithm 2, to count the number of 2-ary de Bruijn partial words of order n with one hole.

Algorithm 2. Computing the number $N_2(n,1)$, where $n \geq 4$

```
1: Find the set $Z$ of all good $z$'s of the form $x\diamond y$ such that $|x| = |y| = n-1$ and
   $M_z(n) = M_2(n,1) = 2n$
2: for all $z \in Z$ do
3:    Construct the Laplacian matrix $L_z = L_{G_2(n,z)}$
4:    Eliminate all rows and columns of $L_z$ that have all zero entries
5:    Calculate the determinant of the matrix $L_z$ after removing the row and column
      that correspond to $x$
6: return The sum of the determinants
```

Remark 2. Step 4 is necessary since some vertices may have become isolated. This still would allow for Eulerian paths, but would make the determinant zero if those rows and columns were left in the Laplacian matrix. We also eliminate the row and column corresponding to x to form the cofactor, since by Theorem 4, x must be the last vertex of the path because $\texttt{ideg}(x) = \texttt{odeg}(x)+1$. In step 5, the $(\texttt{ideg}(x)-1)!$ multiplicative factor is always 1 since $\texttt{ideg}(x) = 2$. Unfortunately, unlike the full case where the sum falls out easily since all cofactors of the single matrix have the same value, the cofactors of the L_z's may be different.

Example 4. Returning to Example 1 with $k = 2$ and $n = 4$, up to reversal and renaming of letters, we only need to consider $z_1 = 110\diamond 001$ to compute $N_2(n,1)$. Referring to the graph on the left in Figure 1, $L_{G_2(4,z_1)}$ is as follows:

$$\begin{array}{c|cccccc}
 & 001 & 010 & 011 & 101 & 110 & 111 \\
\hline
001 & 2 & -1 & -1 & 0 & 0 & 0 \\
010 & 0 & 1 & 0 & -1 & 0 & 0 \\
011 & 0 & 0 & 2 & 0 & -1 & -1 \\
101 & 0 & 0 & -1 & 1 & 0 & 0 \\
110 & -1 & 0 & 0 & 0 & 1 & 0 \\
111 & 0 & 0 & 0 & 0 & -1 & 1
\end{array}$$

Note that the rows and columns corresponding to the vertices 000 and 100 have been removed since all their entries are zeros. If we remove the row and column of vertex 110, we get a determinant of 4. So there are 4 Eulerian paths from 001 to 110 in $G_2(4,z_1)$: $00110\diamond 001011110, 0011110\diamond 0010110, 0010110\diamond 0011110$ and $001011110\diamond 00110$. Since the only z's that are good are $110\diamond 001$, its reversal, and their renamings, we get $N_2(4,1) = 4 \times 4 = 16$.

References

1. Allouche, J.P., Shallit, J.: Automatic Sequences: Theory, Applications, Generalizations. Cambridge University Press, Cambridge (2003)
2. Allouche, J.P.: Sur la complexité des suites infinies. Bulletin of the Belgian Mathematical Society 1, 133–143 (1994)
3. Ferenczi, S.: Complexity of sequences and dynamical systems. Discrete Mathematics 206, 145–154 (1999)

4. Cassaigne, J.: Complexité et facteurs spéciaux. Bulletin of the Belgium Mathematical Society 4, 67–88 (1997)
5. Gheorghiciuc, I.: The subword complexity of a class of infinite binary words. Advances in Applied Mathematics 39, 237–259 (2007)
6. De Bruijn, N.G.: Acknowledgement of priority to C. Flye Sainte-Marie on the counting of circular arrangements of 2n zeros and ones that show each n-letter word exactly once. Technical Report 75–WSK–06, Department of Mathematics and Computing Science, Eindhoven University of Technology, The Netherlands (1975)
7. Alekseyev, M.A., Pevzner, P.A.: Colored de Bruijn graphs and the genome halving problem. IEEE/ACM Transactions on Computational Biology and Bioinformatics 4(1), 98–107 (2007)
8. Blakeley, B., Blanchet-Sadri, F., Gunter, J., Rampersad, N.: On the complexity of deciding avoidability of sets of partial words. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 113–124. Springer, Heidelberg (2009), `www.uncg.edu/cmp/research/unavoidablesets3`
9. Blanchet-Sadri, F.: Algorithmic Combinatorics on Partial Words. Chapman & Hall/CRC Press, Boca Raton (2008)
10. Gross, J.L., Yellen, J.: Handbook of Graph Theory. CRC Press, Boca Raton (2004)
11. Stanley, R.P.: Enumerative Combinatorics, vol. 2. Cambridge University Press, Cambridge (2001)

Complexity Invariance of Real Interpretations*

Guillaume Bonfante[1] and Florian Deloup[2]

[1] Université de Nancy - LORIA, Nancy, France
[2] Université Paul Sabatier, Toulouse - IMT, France

Abstract. In the field of implicit computational complexity, we are considering in this paper the fruitful branch of interpretation methods. In this area, the synthesis problem is solved by Tarski's decision procedure, and consequently interpretations are usually chosen over the reals rather than over the integers. Doing so, one cannot use anymore the (good) properties of the natural (well-) ordering of **N** employed to bound the complexity of programs. We show that, actually, polynomials over the reals benefit from some properties that allow their safe use for complexity. We illustrate this by two characterizations, one of PTIME and one of PSPACE.

To prove the termination of a rewrite system, it is natural to interpret terms into a well-founded ordering. For instance, Lankford describes interpretations as monotone Σ-algebras with domain of interpretation being the natural numbers with their usual ordering (c.f. [16,15]).

However, in the late seventies, Dershowitz showed in a seminal paper [8] that the well-foundedness of the domain of interpretation is not necessary whenever the interpretations are chosen to be monotonic and to have the sub-term property. Thus, the domain of the Σ-algebra mentioned above can be the set of real numbers.

One of the main interesting points about choosing of real numbers rather than natural numbers is that we get (at least from a theoretical point of view) a procedure to verify the validity of an interpretation of a program by Tarski's decomposition procedure [25] and an algorithm to compute interpretations up to some fixed degree. Following Roy et al. [3], the complexity of these algorithms is exponential with respect to the size of the program.

A second good point is that the use of reals (as opposed to integers) enlarges the set of rewriting systems that are compatible with an interpretation, as shown recently by Lucas [17].

In the last years, the study of termination methods has been one of the major tools in implicit computational complexity. For instance, Moser et al. have characterized PTIME by means of POP* in [2], and context dependent interpretations in [22] after their introduction by Hofbauer [12]. One of our two characterizations, Theorem 8, use dependency pairs (c.f. [1]). In this vein, we mention here the work of Hirokawa and Moser [10], and, in the same spirit, Lucas and Peña in [19] made some investigation on the tools of rewriting to tackle the complexity of a first order functional programs.

* Work partially supported by project ANR-08-BLANC-0211-01 (COMPLICE).

© Springer-Verlag Berlin Heidelberg 2010

But, the main concern of the present paper is to show that the structure of polynomials over the reals has an important role from the point of view of complexity. Our thesis is that, in the field of complexity, due to Stengle's Positivstellensatz [24], polynomials over the reals can safely replace polynomials over the integers. It is illustrated by two theorems, Theorem 6 and Theorem 8. We show that one may recover both derivational complexity (up to a polynomial) and size bounds (also, up to a polynomial) on terms as applications of Positivstellensatz. Moreover, this can be done in a constructive way. Our thesis But, let us draw briefly the roadmap of the key technical features of this work.

Given a strict interpretation for a term rewriting system, it follows immediately that for all rewriting steps $s \to t$, we have $(\!| s |\!) > (\!| t |\!)$. If one takes the interpretation on natural numbers (as they were introduced by Lankford [16]), this can be used to give a bound on the derivation height. Thus, Hofbauer and Lautemann have shown in [11] that the derivation height is bounded by a double exponential. However, their argument uses deeply the fact that the interpretation of a term is itself a bound on the derivation height:

$$\mathtt{dh}(t) \leq (\!| t |\!). \tag{1}$$

Suppose $t_0 \to t_1 \to \cdots \to t_n$, since $(\!| t_0 |\!) > (\!| t_1 |\!) > \cdots > (\!| t_n |\!)$, on natural numbers, this means that $n \leq (\!| t_0 |\!)$. Such a proof does not hold with real numbers. The inequalities $(\!| t_i |\!) > (\!| t_{i+1} |\!)$ are due to a) for all rewrite rule $\ell \to r$, the inequality $(\!| \ell |\!) > (\!| r |\!)$ implies that

$$(\!| \ell |\!) \geq (\!| r |\!) + 1 \tag{2}$$

and b) that for all $x_i > y_i$:

$$(\!| f |\!)(x_1, \ldots, x_i, \ldots, x_n) - (\!| f |\!)(x_1, \ldots, y_i, \ldots, x_n) \geq x_i - y_i. \tag{3}$$

The two inequalities 2, 3 do not hold in general for real interpretations. To recover the good properties holding with natural numbers, people have enforced the inequalities on terms. For instance [17,21] suppose the existence of some real $\delta > 0$ such that for any rule $\ell \to r : (\!| \ell |\!) \geq (\!| r |\!) + \delta$. We prove that even without the existence of such a δ, the derivation height of a term t is bounded by $(\!| t |\!)$ up to a polynomial.

To save some space, we have omitted some proofs. The reader will find them in the extended version of the paper, see [5].

1 Preliminaries

We suppose that the reader has familiarity with first order rewriting. We briefly recall some of the main notions of the theory, essentially to fix the notations. Dershowitz and Jouannaud's survey [9] is a good entry point.

Let $\mathcal{X}$ denote a (countable) set of *variables*. Given a *signature* Σ, the set of *terms* over Σ and $\mathcal{X}$ id denoted by $\mathcal{T}(\Sigma, \mathcal{X})$ and the set of *ground terms* as $\mathcal{T}(\Sigma)$. The size $|t|$ of a term t is defined as the number of symbols in t.

Given a signature Σ, a rule is an oriented equation $\ell \to r$ with $\ell, r \in \mathcal{T}(\Sigma, \mathcal{X})$ such that variables occurring in r also occur in ℓ. A Term Rewrite System (TRS) is a finite set of such rules. A TRS induces a rewriting relation denoted by $\to$. The relation $\xrightarrow{+}$ is the transitive closure of $\to$ and $\xrightarrow{*}$ is the reflexive and transitive closure of $\to$. Finally, we say that a term t is a *normal form* if there is no term u such that $t \to u$. Given two terms t and u, $t \xrightarrow{!} u$ denotes the fact that $t \xrightarrow{*} u$ and u is a normal form. We write $t_0 \to^n t_n$ the fact that $t_0 \to t_1 \cdots \to t_n$. One defines the derivation height for a term t as the maximal length of a derivation: $\mathtt{dh}(t) = \max\{n \in \mathbf{N} \mid \exists v : t \to^n v\}$.

A *context* is a term C with a particular variable $\Diamond$. If t is a term, $C[t]$ denotes the term C where the variable $\Diamond$ has been replaced by t. A substitution is a mapping from variables to terms. A substitution σ can be extended canonically to terms and we note $t\sigma$ the application of the substitution σ to the term t.

1.1 Syntax of Programs

Definition 1. *A program is a* 5*-tuple* $f = \langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathtt{main}, \mathcal{E} \rangle$ *with:*

- $\mathcal{C}$ *is a (finite) signature of* constructor *symbols and* $\mathcal{F}$ *a (finite) signature of* function symbols*.* $\mathtt{main} \in \mathcal{F}$ *is the "main" function symbol*
- $\mathcal{E}$ *is a finite set of rules of the shape* $f(p_1, \cdots, p_n) \to r$ *where* $f \in \mathcal{F}$ *and* $p_i \in \mathcal{T}(\mathcal{C}, \mathcal{X})$*.*

Moreover, we suppose programs to be confluent. This is achieved by the following syntactic restriction due to Huet [14] (see also [23]): (i) Each rule $\mathtt{f}(p_1, \ldots, p_n) \to t$ is left-linear, that is a variable appears only once in $\mathtt{f}(p_1, \cdots, p_n)$, and (ii) there are no two left hand-sides which are overlapping.

The program $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathtt{f}, \mathcal{E} \rangle$ computes the partial function $[\![\mathtt{f}]\!] : \mathcal{T}(\mathcal{C})^n \to \mathcal{T}(\mathcal{C})$ defined as follows. For every $u_1, \cdots, u_n \in \mathcal{T}(\mathcal{C})$, $[\![\mathtt{f}]\!](u_1, \cdots, u_n) = v$ iff $\mathtt{f}(u_1, \cdots, u_n) \xrightarrow{*} v$ and $v \in \mathcal{T}(\mathcal{C})$. Otherwise, it is undefined.

Example 1. The following program computes the membership in a list. The constructors of lists are **cons**, **nil**. Elements in the list are the tally natural numbers build from **0** and $\mathbf{s}$.

$$\begin{array}{lll}
\mathtt{not}(\mathbf{tt}) \to \mathbf{ff} & \mathtt{or}(\mathbf{tt}, y) \to \mathbf{tt} & \mathbf{0} = \mathbf{0} \quad \to \mathbf{tt} \\
\mathtt{not}(\mathbf{ff}) \to \mathbf{tt} & \mathtt{or}(x, \mathbf{tt}) \to \mathbf{tt} & \mathbf{0} = \mathbf{s}(y) \to \mathbf{ff} \\
 & \mathtt{or}(\mathbf{ff}, \mathbf{ff}) \to \mathbf{ff} & \mathbf{s}(x) = \mathbf{0} \quad \to \mathbf{ff} \\
 & & \mathbf{s}(x) = \mathbf{s}(y) \to x = y
\end{array}$$

$$\mathtt{in}(x, \mathbf{nil}) \to \mathbf{ff}$$
$$\mathtt{in}(x, \mathbf{cons}(a, l)) \to \mathtt{or}(x = a, \mathtt{in}(x, l))$$

Definition 2 (Call-tree). *Suppose we are given a program* $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathcal{E} \rangle$*. Let* $\rightsquigarrow$ *be the relation*

$$(f, t_1, \ldots, t_n) \rightsquigarrow (g, u_1, \ldots, u_m) \Leftrightarrow f(t_1, \ldots, t_n) \to C[g(v_1, \ldots, v_m)] \xrightarrow{*} C[g(u_1, \ldots, u_m)]$$

where f and g are defined symbols, $t_1, \dots, t_n, u_1, \dots, u_m$ are ground constructor terms and $v_1, \dots, v_m$ are arbitrary (ground) terms. Given a term $f(t_1, \dots, t_n)$, the relation $\rightsquigarrow$ defines a tree whose root is $(f, t_1, \dots, t_n)$ and η' is a daughter of η iff $\eta \rightsquigarrow \eta'$.

1.2 Interpretations of Programs

Given a signature Σ, a Σ-algebra on the domain $\mathbf{R}^+$ is a mapping $(\!| - |\!)$ which associates to every n-ary symbol $f \in \Sigma$ an n-ary function $(\!| f |\!) : {\mathbf{R}^+}^n \to \mathbf{R}^+$. Such a Σ-algebra can be extended to terms by:

- $(\!| x |\!) = 1_{\mathbf{R}^+}$, that is the identity on $\mathbf{R}^+$, for $x \in \mathcal{X}$,
- $(\!| f(t_1, \dots, t_m) |\!) = \mathrm{comp}((\!| f |\!), (\!| t_1 |\!), \dots, (\!| t_m |\!))$ where comp is the composition of functions.

Given a term t with n variables, $(\!| t |\!)$ is a function ${\mathbf{R}^+}^n \to \mathbf{R}^+$.

Definition 3. *Given a program $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathsf{f}, \mathcal{E} \rangle$, let us consider a $(\mathcal{C} \cup \mathcal{F})$-algebra $(\!| - |\!)$ on $\mathbf{R}^+$. It is said to:*

1. *be strictly monotonic if for any symbol f, the function $(\!| f |\!)$ is a strictly monotonic function, that is if $x_i > x'_i$, then*
$$(\!| f |\!)(x_1, \dots, x_n) > (\!| f |\!)(x_1, \dots, x'_i, \dots, x_n),$$
2. *be weakly monotonic if for any symbol f, the function $(\!| f |\!)$ is a weakly monotonic function, that is if $x_i \geq x'_i$, then*
$$(\!| f |\!)(x_1, \dots, x_n) \geq (\!| f |\!)(x_1, \dots, x'_i, \dots, x_n),$$
3. *have the strict sub-term property if for any symbol f, the function $(\!| f |\!)$ verifies $(\!| f |\!)(x_1, \dots, x_n) > x_i$ with $i \in \{1, \dots, n\}$,*
4. *to be strictly compatible (with the rewriting relation) if for all rules $\ell \to r$, $(\!| \ell |\!) > (\!| r |\!)$,*
5. *to be a sup-approximation if for all constructor terms $t_1, \dots, t_n$, we have the inequality $(\!| f(t_1, \dots, t_n) |\!) \geq (\!| [\![f]\!](t_1, \dots, t_n) |\!)$.*

Definition 4. *Given a program $\langle \mathcal{X}, \mathcal{C}, \mathcal{F}, \mathsf{f}, \mathcal{E} \rangle$, a $(\mathcal{C} \cup \mathcal{F})$-algebra on $\mathbf{R}^+$ is said to be a strict interpretation whenever it verifies (1), (3), (4). It is a sup-interpretation whenever it verifies (2) and (5).*

Sup-interpretation have been introduced by Marion and Pechoux in [20]. We gave here a slight variant of their definition. In [20], the last inequality refers to the size of normal forms. We preferred to have a more uniform definition. Clearly, a strict interpretation is a sup-interpretation.When we want to speak arbitrarily of one of those concepts, we use the generic word "interpretation". We also use this terminology to speak about the function $(\!| f |\!)$ given a symbol f.

Finally, by default, we restrict the interpretations of symbols to be *Max-Poly functions*, that is functions obtained by finite compositions of the constant functions, maximum, addition and multiplication.

Definition 5. *The interpretation of a symbol f is said to be additive if it has the shape $\sum_i x_i + c$ with $c > 0$. A program with an interpretation is said to be additive when its* constructors *are additive.*

Example 2. The program given in Example 1 has both an additive strict interpretation (left side, black) and an additive sup-interpretation (right side, blue):

$$\begin{aligned}
(\!|\mathbf{tt}|\!) = (\!|\mathbf{ff}|\!) = (\!|\mathbf{0}|\!) = (\!|\mathbf{nil}|\!) &= 1 \\
(\!|\mathbf{s}|\!)(x) &= x + 1 \\
(\!|\mathbf{cons}|\!)(x, y) &= x + y + 3 \\
(\!|\mathtt{not}|\!)(x) &= x + 1 \\
(\!|\mathtt{or}|\!)(x, y) = (\!|{=}|\!)(x, y) &= x + y + 1 \\
(\!|\mathtt{in}|\!)(x, y) &= (x + 1)(y + 1)
\end{aligned}
\qquad
\begin{aligned}
(\!|\mathbf{tt}|\!) = (\!|\mathbf{ff}|\!) = (\!|\mathbf{0}|\!) = (\!|\mathbf{nil}|\!) &= 1 \\
(\!|\mathbf{s}|\!)(x) &= x + 1 \\
(\!|\mathbf{cons}|\!)(x, y) &= x + y + 1 \\
(\!|\mathtt{not}|\!)(x) &= 1 \\
(\!|\mathtt{or}|\!)(x, y) = (\!|=|\!)(x, y) &= 1 \\
(\!|\mathtt{in}|\!)(x, y) &= 1
\end{aligned}$$

Example 3. The Quantified Boolean Formula (QBF) problem is well known to be PSPACE complete. It consists in determining the validity of a boolean formula with quantifiers over propositional variables. Without loss of generality, we restrict formulae to $\neg, \vee, \exists$. Variables are represented by tally numbers. The QBF problem is solved extending the preceding program with:

$$\begin{aligned}
\mathtt{verify}(\mathbf{Var}(x), t) &\to \mathtt{in}(x, t) \\
\mathtt{verify}(\mathbf{Not}(\varphi), t) &\to \mathtt{not}(\mathtt{verify}(\varphi, t)) \\
\mathtt{verify}(\mathbf{Or}(\varphi_1, \varphi_2), t) &\to \mathtt{or}(\mathtt{verify}(\varphi_1, t), \mathtt{verify}(\varphi_2, t)) \\
\mathtt{verify}(\mathbf{Exists}(n, \varphi), t) &\to \mathtt{or}(\mathtt{verify}(\varphi, \mathbf{cons}(n, t)), \mathtt{verify}(\varphi, t)) \\
\mathtt{qbf}(\varphi) &\to \mathtt{verify}(\varphi, \varepsilon)
\end{aligned}$$

It has a sup-interpretation but not a strict interpretation:

$$\begin{aligned}
(\!|\mathbf{Not}|\!)(x) = (\!|\mathbf{Var}|\!)(x) &= x + 1 \\
(\!|\mathbf{Or}|\!)(x, y) = (\!|\mathbf{Exists}|\!)(x, y) &= x + y + 1 \\
(\!|\mathtt{verify}|\!)(x, y) = (\!|\mathtt{qbf}|\!)(x) &= 1
\end{aligned}$$

Actually, as Theorem 6 will prove it, unless PTIME = PSPACE, there is no program computing QBF with an additive strict interpretation.

2 Positivstellensatz and Applications

In this section, we introduce a deep mathematical result, the Positivstellensatz. Then we give some applications to polynomial interpretations. They will be key points of the Theorems 6 and 8 in our analysis of the role of reals in complexity (§3).

Let $n > 0$. Denote by $\mathbf{R}[x_1, \ldots, x_n]$ the $\mathbf{R}$-algebra of polynomials with real coefficients. Denote by $(\mathbf{R}^+)^n = \{x = (x_1, \ldots, x_n) \in \mathbf{R}^n \mid x_1, \ldots, x_n > 0\}$ the first quadrant. Since we need to consider only the $\mathbf{R}$-algebra of polynomial functions $(\mathbf{R}^+)^n \to \mathbf{R}$, it will be convenient to identify the two spaces. In particular throughout this section, all polynomial functions are defined on $(\mathbf{R}^+)^n$.

Theorem 3 (Positivstellensatz, Stengle [24]). *Suppose that we are given polynomials $P_1, \dots, P_m \in \mathbf{R}[x_1, \dots, x_k]$, the following two assertions are equivalent:*

1. $\{x_1, \dots, x_k : P_1(x_1, \dots, x_k) \geq 0 \wedge \dots \wedge P_m(x_1, \dots, x_k) \geq 0\} = \emptyset$
2. $\exists Q_1, \dots, Q_m : -1 = \sum_{i \leq m} Q_i P_i$ *where each Q_i is a sum of squares of polynomials (and so is positive and monotonic).*

Moreover, these polynomials $Q_1, \dots, Q_m$ can effectively computed. We refer the reader to the work of Lombardi, Coste and Roy [6]. As a consequence, all the constructions given below can be actually (at least theoretically) computed.

It will be convenient to derive from the Positivstellensatz some propositions useful for our applications.

Proposition 2. *Suppose that a TRS (Σ, R) admits an interpretation $(\!|-|\!)$ over* Max-Poly *such that for all rules $\ell \to r$, we have $(\!|\ell|\!) > (\!|r|\!)$. There is a positive, monotonic polynomial P such that for any rule $\ell \to r$, we have $(\!|\ell|\!)(x_1, \dots, x_k) - (\!|r|\!)(x_1, \dots, x_k) \geq \dfrac{1}{P(x_1, \dots, x_k)}$.*

The proof is direct consequence of Theorem 3 when ℓ and r are polynomials. By a finite case analysis, one may cope with the max function. One may notice that one cannot a priori find some constant $\delta > 0$ such that: $(\!|\ell|\!)(x_1, \dots, x_k) \geq (\!|r|\!)(x_1, \dots, x_k) + \delta$. Indeed, suppose that $(\!|\ell|\!)(x_1, \dots, x_k) - (\!|r|\!)(x_1, \dots, x_k) > 0$. Observe that $\lim_{x \to 0} P(x, 1/x) = \lim_{x \to 0} x^2 = 0$. However, taking $Q(x, y) = (1 + x + y)^2$, one has $P(x, y)Q(x, y) \geq 1$ for all $x, y \geq 0$.

Proposition 2 has an important consequence. Since, in a derivation all terms have an interpretation bounded by the interpretation of the first term, there is a minimal decay for each rule of the derivation. Then, due to the next Theorem, the result can be extended to contexts.

Theorem 4. *Given a polynomial $P \in \mathbf{R}[x_1, \dots, x_n]$ such that*

(i) $\forall x_1 \geq 0, \dots, x_n \geq 0 : P(x_1, \dots, x_n) > \max(x_1, \dots, x_n)$,

(ii) $\forall x_1 \geq 0, \dots, x_n \geq 0 : \dfrac{\partial P}{\partial x_i}(x_1, \dots, x_n) > 0$ *for all $i \leq n$,*

then, there exist $A > 0$ such that for any $\Delta > 0$, we have $P(x_1, \dots, x_i + \Delta, \dots, x_n) > P(x_1, \dots, x_n) + \Delta$ whenever $||x|| > A$.

In other words, we recovered some equivalent Equations to the Equations 2, 3 for sufficiently large terms.

Proposition 3. *Suppose that a TRS (Σ, R) admits a strict interpretation $(\!|-|\!)$ over* Max-Poly*. For all $A > 0$, the set of terms $\{t \in \mathcal{T}(\Sigma) \mid (\!|t|\!) < A\}$ is finite.*

Proposition 4. *Suppose that a TRS (Σ, R) admits a strict interpretation $(\!|-|\!)$ over Poly. There are a real $A > 0$ and a positive, monotonic polynomial P such that for all $x_1, \dots, x_n \geq 0$, if $x_{i_1}, \dots, x_{i_k} > A$, then for all symbols f, we have*

$$(\!| f |\!)(x_1,\dots,x_n) \geq x_{i_1} + \cdots + x_{i_k} + \frac{1}{P((\!| f |\!)(x_1,\dots,x_n))}.$$

This latter result gives (more or less) directly a bound on the size of terms. It is a consequence of Theorem 3 and the following Theorem.

Theorem 5. *Given a polynomial $P \in \mathbf{R}[x_1,\dots,x_n]$ such that*

(i) $\forall x_1,\dots,x_n \geq 0 : P(x_1,\dots,x_n) > \max(x_1,\dots,x_n)$,
(ii) $\forall x_i' > x_i, x_1,\dots,x_n \geq 0 : P(x_1,\dots,x_i',x_{i+1},\dots,x_n) > P(x_1,\dots,x_n)$,

then, there exist $A \geq 0$ such that $P(x_1,\dots,x_n) > x_1+\cdots+x_n$ whenever $||x|| > A$.

All the hypotheses are necessary. If $P(x,y)$ is not supposed to be greater than $\max(x,y)$, you can simply take $P(x,y) = (x+y)/2$. It is strictly monotonic, but clearly, $P(x,y) < x+y$ for all $x,y > 0$.

The second hypothesis is also necessary. A counter example is given by $P(x,y) = 16(x-y)^2 + (3/2)x + 1$.

3 The Role of Reals in Complexity

We have now all the tools to prove that reals can safely replace integers from a complexity point of view. This is illustrated by Theorem 6 and Theorem 8.

Theorem 6. *Functions computed by programs with an additive strict interpretation (over the reals) are exactly* Ptime *functions.*

The rest of the section is devoted to the proof of the Theorem. The main difficulty of the proof is that inequalities as given by the preceding section only hold for sufficiently large values. So, the main issue is to split "small" terms (and "small rewriting steps") from "large" ones. Positivstellensatz gives us the arguments for the large terms (Lemma 7), Lemmas 8,9 show that there are not too many small steps between two large steps. Lemma 11 describe how small steps and big steps alternate.

From now on, we suppose we are given a program with an additive strict interpretation over polynomials. The following Lemmas are direct applications of Proposition 2,4, they are the main steps to prove both Theorem 6 and Theorem 8. A full proof of the lemmas can be found in the technical report.

Lemma 7. *There is a polynomial P and a real $A > 0$ such that for all steps $\ell\sigma \to r\sigma$ with $(\!| r\sigma |\!) > A$, then, for all contexts C, we have $(\!| C[\ell\sigma] |\!) \geq (\!| C[r\sigma] |\!) + \dfrac{1}{P((\!| \ell\sigma |\!))}$.*

Proof. This is a consequence of Proposition 2.

Definition 7. *Given a real $A > 0$, we say that the A-size of a closed term t is the number of subterms u of t (including itself) such that $(\!| u |\!) > A$. We note $|t|_A$ the A-size of t.*

Lemma 8. *There is a constant A such that for all $C > A$, there is a polynomial Q for which $|t|_C \leq Q((\!|t|\!))$ for all closed terms t.*

Proof. This is consequence of Proposition 4.

For $A > 0$, we say that $t = C[\ell\sigma] \to C[r\sigma] = u$ is an A-step whenever $(\!|r\sigma|\!) > A$. We note such a rewriting step $t \to_{>A} u$. Otherwise, it is an $\leq A$-step, and we note it $t \to_{\leq A} u$. We use the usual $*$ notation for transitive closure. In case we restrict the relation to the call by value strategy[1], we add "cbv" as a subscript. Take care that an $\to_{\leq A}$-normal form is not necessarily a normal form for $\to$.

Lemma 9. *There is a constant A such that for all $C > A$ there is a (monotonic) polynomial P such that for all terms t, any call by value derivation $t \to^*_{\leq C,cbv} u$ has length less than $P((\!|t|\!))$.*

Proof. This is a consequence of Proposition 2.

Lemma 10. *For constructor terms, we have $(\!|t|\!) \leq \Gamma \times |t|$ for some constant Γ.*

Proof. Take $\Gamma = \max\{\dfrac{1}{\gamma_c} \mid (\!|c|\!)(x_1, \ldots, x_n) = \sum_{i=1}^n x_i + \gamma_c\}$. By induction on terms.

Lemma 11. *Let us suppose we are given a program with an additive strict interpretation. There is a strategy such that for all function symbol f, for all constructor terms $t_1, \ldots, t_n$, any derivation following the strategy starting from $f(t_1, \ldots, t_n)$ has length bounded by $Q(\max(|t_1|, \ldots, |t_n|))$ where Q is a polynomial.*

Proof. Let us consider A as defined in Lemma 9, B and P_1 as defined in Lemma 7. We define $C = \max(A, B)$. Let P_0 be the polynomial thus induced from Lemma 9. Finally, let us consider the strategy as introduced above: rewrite as long as possible the according to $\to_{\leq C,\text{cbv}}$, and then, apply an C-step. That is, we have $t_1 \to^*_{\leq C,\text{cbv}} t'_1 \to_{>C,\text{cbv}} t_2 \to^*_{\leq C,\text{cbv}} t'_2 \to^*$. In Lemma 9, we have seen that there are at most $P_0((\!|t_i|\!))$ steps in the derivation $t_i \to^*_{\leq C,\text{cbv}} t'_i$. From Lemma 7, we can state that there are at most $(\!|t_1|\!) \times P_1((\!|t_1|\!))$ such C-steps. Consequently, the derivation length is bounded by $(\!|t_1|\!) \times P_1((\!|t_1|\!)) \times P_0((\!|t_1|\!))$ since for all $i \geq 1$, $(\!|t_i|\!) \leq (\!|t_1|\!)$.

Consider now a function symbol $f \in \mathcal{F}$, from Lemma 10, $(\!|f(t_1, \ldots, t_n)|\!) = (\!|f|\!)((\!|t_1|\!), \ldots, (\!|t_n|\!)) \leq (\!|f|\!)(\Gamma \max(|t_1|, \ldots, |t_n|), \ldots, \Gamma \max(|t_1|, \ldots, |t_n|))$. Then, the conclusion is immediate.

Proof. Of Theorem 6 With the strategy defined above, we have seen that the derivation length of a term $f(t_1, \ldots, t_n)$ is polynomial wrt to $\max(|t_1|, \ldots, |t_n|)$. The computation can be done in polynomial time due to dal Lago and Martini, see [7], together with the fact that the normal form has polynomial size (Lemma 10). For the converse part, we refer the reader to [4] where a proof that Ptime programs can be computed by functional programs with strict interpretations over the integers. This proof can be safely used in the present context.

[1] Innermost in the present context.

3.1 Dependency Pairs with Polynomial Interpretation over the Reals

Termination by Dependency Pairs is a general method introduced by Arts and Giesl [1]. It puts into light recursive calls. Suppose $f(t_1, \dots, t_n) \to C[g(u_1, \dots, u_m)]$ is a rule of the program. Then, $(F(t_1, \dots, t_n), G(u_1, \dots, u_m))$ is a dependency pair where F and G are new symbols associated to f and g respectively. $S(\mathcal{C}, \mathcal{F}, R)$ denotes the program thus obtained by adding these rules. The dependency graph links dependency pairs $(u, v) \to (u', v')$ if there is a substitution σ such that $\sigma(v) \xrightarrow{*} \sigma(u)$ and termination is obtained when there is no cycles in the graph. Since the definition of the graph involves the rewriting relation, its computation is undecidable. In practice, one gives an approximation of the graph which is bigger. Since this is not the issue here, we suppose that we have a procedure to compute this supergraph which we call the dependency graph.

Theorem 7. *[Arts,Giesl [1]] A TRS $(\mathcal{C}, \mathcal{F}, R)$ is terminating iff there exists a well-founded weakly monotonic quasi-ordering $\geq$, where both $\geq$ and $>$ are closed under substitution, such that*

- *$\ell \geq r$ for all rules $\ell \to r$,*
- *$s \geq t$ for all dependency pairs (s, t) on a cycle of the dependency graph and*
- *$s > t$ for at least one dependency pair on each cycle of the graph.*

It is natural to use sup interpretations for the quasi-ordering and the ordering of terms. However, the ordering $>$ is not well-founded on $\mathbf{R}$, so that system may not terminate. Here is such an example.

Example 4. Consider the non terminating system:

$$\begin{pmatrix} \mathtt{f}(\mathbf{0}) \to \mathbf{0} \\ \mathtt{f}(x) \to \mathtt{f}(\mathbf{s}(x)) \end{pmatrix}$$

Take $(\!| 0 |\!) = 1$, $(\!| \mathbf{s} |\!)(x) = x/2$. There is a unique dependency pair $F(x) \to F(\mathbf{s}(x))$. We define $(\!| F |\!)(x) = (\!| f |\!)(x) = x + 1$.

One way to avoid these infinite descent is to force the inequalities over reals to be of the form $P(x_1, \dots, x_n) \geq Q(x_1, \dots, x_n) + \delta$ for some $\delta > 0$ (see for instance Lucas's work [18]). Doing so, one gets a well-founded ordering on reals. We propose an alternative approach to that problem, keeping the original ordering of $\mathbf{R}$.

Definition 8. *A $\mathbf{R}$-DP-interpretation for a program associates to each symbol f a monotonic function $(\!| f |\!)$such that*

1. *constructors have additive interpretations,*
2. *$(\!| \ell |\!) \geq (\!| r |\!)$ for $\ell \to r \in R$,*
3. *$(\!| s |\!) \geq (\!| r |\!)$ for $(s, r) \in DP(R)$,*
4. *for each dependency pair (s, t) in a cycle, $(\!| s |\!) > (\!| r |\!)$ holds.*

Example 5. Let us come back to Example 3. The QBF problem can be given a **R**-DP interpretation. Let us add the interpretations:

$$\begin{aligned} (\!|\mathrm{NOT}|\!)(x) &= x \\ (\!|\mathrm{OR}|\!)(x,y) = (\!|\mathrm{EQ}|\!)(x,y) &= \max(x,y) \\ (\!|\mathrm{IN}|\!)(x,y) &= x+y \\ (\!|\mathrm{VERIFY}|\!)(x,y) &= 2x+y+1 \\ (\!|\mathrm{QBF}|\!)(x) &= 2x+1 \end{aligned}$$

Theorem 8. *Functions computed by programs*

- *with additive* **R**-*DP-interpretations*
- *the interpretation of any capital symbol F has the sub-term property*

are exactly Pspace *computable functions.*

Proof. The completeness comes from the example of the QBF, plus the compositionality of such interpretation.

In the other direction, the key argument is to prove that the call tree has a polynomial depth w.r.t. the size of arguments. The proof relies again on Lemmas 7, 8, 9, 11 adapted to dependency pairs (in cycles). Indeed, since capital symbol have the sub-term property, the lemmas are actually valid in the present context.[2] The rewriting steps of dependency pairs can be reinterpreted as depth-first traversal in the call-tree. Thus, we can state that the depth of the call tree is polynomial, as we stated in an analogous way that the derivation length was polynomial.

4 Conclusion

If one goes back to the two characterization of complexity classes presented in this paper, one sees that we essentially use two arguments: a) interpretations with the subset properties provide a polynomial bound wrt the interpretation of the initial interpretation, and b) the size of terms is polynomial w.r.t. their interpretations.

As a consequence, our result can be used in other context such as proofs of termination by matrix interpretations [13] or context dependent interpretations [12]. Potentially, any system dealing with decreasing chain of (interpreted) terms could be treated.

Acknowledgement. Antoine Henrot for his helpful comments on an earlier version of the contribution.

References

1. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theoretical Computer Science 236(1-2), 133–178 (2000)
2. Avanzini, M., Moser, G.: Complexity analysis by rewriting. In: Garrigue, J., Hermenegildo, M.V. (eds.) FLOPS 2008. LNCS, vol. 4989, pp. 130–146. Springer, Heidelberg (2008)

[2] Lemma 8 is immediate here since we focus on terms of the shape $F(t_1, \ldots, t_n)$ where $t_1, \ldots, t_n$ are constructor terms.

3. Basu, S., Pollack, R., Roy, M.-F.: Algorithms in real algebraic geometry. Springer, Heidelberg (2003)
4. Bonfante, G., Cichon, A., Marion, J.-Y., Touzet, H.: Algorithms with polynomial interpretation termination proof. Journal of Functional Programming 11(1), 33–53 (2001)
5. Bonfante, G., Deloup, F.: Complexity invariance of real interpretations. Technical report, LORIA (2010), `http://www.loria.fr/~bonfante/ciri.pdf`
6. Coste, M., Lombardi, H., Roy, M.-F.: Dynamical method in algebra: Effective nullstellensätze. Annals of Pure and Applied Logic 111, 203–256 (2001)
7. dal Lago, U., Martini, S.: Derivational complexity is an invariant cost model. In: Foundational and Practical Aspects of Resource Analysis, FOPARA 2009 (2009)
8. Dershowitz, N.: A note on simplification orderings. Information Processing Letters, 212–215 (1979)
9. Dershowitz, N., Jouannaud, J.-P.: Rewrite systems. In: Handbook of Theoretical Computer Science, vol. B, pp. 243–320 (1990)
10. Hirokawa, N., Moser, G.: Automated complexity analysis based on the dependency pair method. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 364–379. Springer, Heidelberg (2008)
11. Hofbauer, D., Lautemann, C.: Termination proofs and the length of derivations. In: Dershowitz, N. (ed.) RTA 1989. LNCS, vol. 355, pp. 167–177. Springer, Heidelberg (1989)
12. Hofbauer, D.: Termination proofs by context-dependent interpretations. In: Middeldorp, A. (ed.) RTA 2001. LNCS, vol. 2051, pp. 108–121. Springer, Heidelberg (2001)
13. Hofbauer, D.: Proving termination with matrix interpretations. In: Pfenning, F. (ed.) RTA 2006. LNCS, vol. 4098, pp. 50–65. Springer, Heidelberg (2006)
14. Huet, G.: Confluent reductions: Abstract properties and applications to term rewriting systems. Journal of the ACM 27(4), 797–821 (1980)
15. Huet, G., Oppen, D.: Equations and rewrite rules: A survey. In: Book, R.V. (ed.) Formal Language Theory: Perspectives and Open Problems, pp. 349–405. AP (1980)
16. Lankford, D.S.: On proving term rewriting systems are noetherien. Technical report (1979)
17. Lucas, S.: On the relative power of polynomials with real, rational, and integer coefficients in proofs of termination of rewriting. Appl. Algebra Eng., Commun. Comput. 17(1), 49–73 (2006)
18. Lucas, S.: Practical use of polynomials over the reals in proofs of termination. In: PPDP 2007: Proceedings of the 9th ACM SIGPLAN international conference on Principles and practice of declarative programming, pp. 39–50. ACM, New York (2007)
19. Lucas, S., Peña, R.: Rewriting techniques for analysing termination and complexity bounds of safe programs. In: Hanus, M. (ed.) LOPSTR 2008. LNCS, vol. 5438, pp. 43–57. Springer, Heidelberg (2009)
20. Marion, J.-Y., Péchoux, R.: Resource analysis by sup-interpretation. In: Hagiya, M., Wadler, P. (eds.) FLOPS 2006. LNCS, vol. 3945, pp. 163–176. Springer, Heidelberg (2006)
21. Marion, J.-Y., Péchoux, R.: Characterizations of polynomial complexity classes with a better intensionality. In: Antoy, S., Albert, E. (eds.) PPDP, pp. 79–88. ACM, New York (2008)

22. Moser, G., Schnabl, A.: Proving quadratic derivational complexities using context dependent interpretations. In: Voronkov, A. (ed.) RTA 2008. LNCS, vol. 5117, pp. 276–290. Springer, Heidelberg (2008)
23. Rosen, B.: Tree-manipulating systems and church-rosser theorems. Journal of the ACM 20(1), 160–187 (1973)
24. Stengle, G.: A nullstellensatz and a positivstellensatz in semialgebraic geometry. Mathematische Annalen 207(2), 87–97 (1973)
25. Tarski, A.: A Decision Method for Elementary Algebra and Geometry, 2nd edn. University of California Press, California (1951)

Pivot and Loop Complementation on Graphs and Set Systems

Robert Brijder[⋆] and Hendrik Jan Hoogeboom

Leiden Institute of Advanced Computer Science,
Leiden University, The Netherlands
rbrijder@liacs.nl

Abstract. We study the interplay between principal pivot transform (pivot) and loop complementation for graphs. This is done by generalizing loop complementation (in addition to pivot) to set systems. We show that the operations together, when restricted to single vertices, form the permutation group S_3. This leads, e.g., to a normal form for sequences of pivots and loop complementation on graphs. The results have consequences for the operations of local complementation and edge complementation on simple graphs: an alternative proof of a classic result involving local and edge complementation is obtained, and the effect of sequences of local complementations on simple graphs is characterized.

1 Introduction

Local complementation has originally been considered in [10] as a transformation on circle graphs (or overlap graphs), modelling a natural transformation on its underlying interval segments. Subsequently, the graph transformation edge complementation has been defined in terms of local complementation in [5]. Many computational application areas have since appeared for both graph transformations. For example, local complementation on simple graphs retains the entanglement of the corresponding graph states in quantum computing [11]. Local complementation is also of main interest in relation to rank-width in the vertex-minor project initiated by Oum [15]. Moreover, edge complementation is fundamentally related to the interlace polynomial [2,3,1], the definition of which is motivated by the computation of the number of k-component circuit partitions in a graph.

Local and edge complementation as mentioned above are concerned with simple graphs, where loops are not allowed. It turns out that edge complementation on simple graphs is a special case of a general matrix operation called principal pivot transform (PPT, or simply pivot), due to Tucker [17], capable of partially (component-wise) inverting a given matrix. By [4], edge complementation can then be described in terms of set systems (more specifically, in terms of delta-matroids due to a specific exchange axiom that they fulfill).

As observed in [13], local and edge complementation can be naturally modified for graphs where loops are allowed, in such a way that they are both a special

[⋆] Corresponding author. Supported by the Netherlands Organization for Scientific Research (NWO), project "Annotated graph mining".

© Springer-Verlag Berlin Heidelberg 2010

case of PPT. From now on we refer to graphs where loops are allowed as *graphs* for short. On graphs, local and edge complementation have natural applications as well, e.g., the formal study of gene assembly in ciliates [12,8], a research area within computational biology. Surprisingly, the similarity between local and edge complementation for simple graphs on the one hand and pivots on matrices (or graphs) on the other hand has been largely unnoticed, and as a result they have been studied almost independently.

The aim of this paper is to bridge this gap between graphs and simple graphs by considering the interplay of pivot and loop complementation (flipping the existence of loops for a given set of vertices) on graphs. By generalizing loop complementation to set systems, we obtain a common viewpoint for the two operations: pivots and loop complementations are elements of order 2 (i.e., involutions) in the permutation group S_3 (by restricting to single vertices). We obtain a normal form for sequences of pivots and loop complementations on graphs. As a consequence a number of results for local and edge complementations on simple graphs are obtained including an alternative proof of a classic result [5] relating local and edge complementation ($*\{u,v\} = *\{u\} * \{v\} * \{u\}$, see Proposition 13). Finally we characterize the effect of sequences of local complementations on simple graphs.

Due to space constraints, the proofs of the results (except for Proposition 13) and some remarks and examples are omitted. Also a section on maximal pivots is omitted. We refer to [7] for a full version of this paper.

2 Notation and Terminology

In this paper matrix computations will be over $\mathbb{F}_2$, the field consisting of two elements. We will often consider this field as the Booleans, and its operations addition and multiplication are as such equal to the logical exclusive-or and logical conjunction, which are denoted by $\oplus$ and $\wedge$ respectively. These operations carry over to sets, e.g., for sets $A, B \subseteq V$ and $x \in V$, $x \in A \oplus B$ iff $(x \in A) \oplus (x \in B)$.

A *set system* (over V) is a tuple $M = (V, D)$ with V a finite set and $D \subseteq \mathcal{P}(V)$ a family of subsets of V.

For a $V \times V$-matrix A (the columns and rows of A are indexed by finite set V) and $X \subseteq V$, $A[X]$ denotes the principal submatrix of A w.r.t. X, i.e., the $X \times X$-matrix obtained from A by restricting to rows and columns in X.

We consider undirected graphs without parallel edges, however we do allow loops. For graph $G = (V, E)$ we use $V(G)$ and $E(G)$ to denote its set of vertices V and set of edges E, respectively, where for $x \in V$, $\{x\} \in E$ iff x has a loop. For $X \subseteq V$, we denote the subgraph of G induced by X as $G[X]$.

With a graph G one associates its adjacency matrix $A(G)$, which is a $V \times V$-matrix $(a_{u,v})$ over $\mathbb{F}_2$ with $a_{u,v} = 1$ iff $\{u, v\} \in E$ (we have $a_{u,u} = 1$ iff $\{u\} \in E$). In this way, the family of graphs with vertex set V corresponds precisely to the family of symmetrical $V \times V$-matrices over $\mathbb{F}_2$. Therefore we often make no distinction between a graph and its matrix, so, e.g., by the determinant of graph

G, denoted $\det G$, we will mean the determinant $\det A(G)$ of its adjacency matrix (computed over $\mathbb{F}_2$). By convention, $\det(G[\varnothing]) = 1$.

3 Pivots

In general the pivot operation is defined for matrices over arbitrary fields, e.g., as done in [16]. In this paper we restrict to symmetrical matrices over $\mathbb{F}_2$, which leads to a number of additional (equivalent) viewpoints to the same operation.

Matrices. Let A be a $V \times V$-matrix (over an arbitrary field), and let $X \subseteq V$ be such that the corresponding principal submatrix $A[X]$ is nonsingular, i.e., $\det A[X] \neq 0$. The *pivot* of A on X, denoted by $A * X$, is defined as follows. If $P = A[X]$ and $A = \left(\begin{array}{c|c} P & Q \\ \hline R & S \end{array}\right)$, then

$$A * X = \left(\begin{array}{c|c} P^{-1} & -P^{-1}Q \\ \hline RP^{-1} & S - RP^{-1}Q \end{array}\right).$$

The pivot can be considered a partial inverse, as A and $A * X$ satisfy the following characteristic relation, where the vectors x_1 and y_1 correspond to the elements of X.

$$A \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \text{ iff } A * X \begin{pmatrix} y_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ y_2 \end{pmatrix} \tag{1}$$

Equality (1)) can be used to define $A * X$ given A and X: any matrix B satisfying this equality is of the form $B = A * X$, see [16, Theorem 3.1], and therefore such B exists precisely when $\det A[X] \neq 0$. Note that if $\det A \neq 0$, then $A * V = A^{-1}$.

From now on we restrict to graphs, i.e., symmetric matrices A over $\mathbb{F}_2$. Hence, $\det A \neq 0$ is equivalent to $\det A = 1$ and moreover $A = A^T$ (the transpose of A). It is easy to verify that (over $\mathbb{F}_2$) $A * X$ is symmetric whenever A is, i.e., if A represents a graph, then $A * X$ represents a graph.

The following fundamental result on pivots, stated here for the case $\mathbb{F}_2$, is due to [17] (see also [9, Theorem 4.1.1] for an elegant proof using Equality (1)).

Proposition 1 ([17]). *Let A be a (not necessary symmetric) $V \times V$-matrix over $\mathbb{F}_2$, and let $X \subseteq V$ be such that* $\det A[X] = 1$. *Then, for $Y \subseteq V$,* $\det(A * X)[Y] = \det A[X \oplus Y]$.

Set Systems. Let M be a set system over V. We define, for $X \subseteq V$, the *pivot* (often called *twist* in the literature, see, e.g., [13]) $M * X = (V, D * X)$, where $D * X = \{Y \oplus X \mid Y \in D\}$.

For graph G, let $\mathcal{M}_G = (V(G), D_G)$ be the set system with $D_G = \{X \subseteq V(G) \mid \det G[X] = 1\}$. Graph G can be (re)constructed given $\mathcal{M}_G$: $\{u\}$ is a loop in G iff $\{u\} \in D_G$, and $\{u, v\}$ is an edge in G iff $(\{u, v\} \in D_G) \oplus ((\{u\} \in D_G) \wedge (\{v\} \in D_G))$, see [6, Property 3.1]. Hence the mapping $\mathcal{M}_{(\cdot)}$ which assigns to each graph G its set system $\mathcal{M}_G$ is injective. In this way, the family of graphs (with set V of vertices) can be considered as a subset of the family of set systems (over set V).

We now recall (from, e.g., [13]) that the pivot operation for graphs coincides with the pivot operation for set systems.

We let $G * X$ denote the graph with adjacency matrix $A(G) * X$. By Proposition 1, for $\mathcal{M}_{G*X}$ we have $Z \in D_{G*X}$ iff $\det((G*X)[Z]) = 1$ iff $\det(G[X \oplus Z]) = 1$ iff $X \oplus Z \in D_G$ iff $Z \in D_G * X$. Hence $\mathcal{M}_{G*X} = \mathcal{M}_G * X$, which shows that pivot on set systems form an alternative definition of pivot on graphs. Note that while for set system M over V, $M * X$ is defined for all $X \subseteq V$, $G * X$ is defined precisely when $\det G[X] = 1$, or equivalently, when $X \in D_G$, which in turn is equivalent to $\varnothing \in D_G * X$. Since $D_{G*X} = D_G * X$, it is easy to see that if $(G * X) * Y$ is defined, then $G * (X \oplus Y)$ is defined and they are equal.

It turns out that $\mathcal{M}_G$ has a special structure, that of a *delta-matroid* [4]. A delta-matroid is a set system that satisfies the symmetric exchange axiom: For all $X, Y \in D_G$ and all $x \in X \oplus Y$, either $X \oplus \{x\} \in D$ or there is a $y \in X \oplus Y$ with $y \neq x$ such that $X \oplus \{x, y\} \in D$[1]. In this paper we will not use this property. In fact, we will consider an operation on set systems that does not retain this property of delta-matroids, cf. Example 8.

Graphs. The pivots $G * X$ where X is a minimal element of $\mathcal{M}_G \backslash \{\varnothing\}$ w.r.t. inclusion are called *elementary*. It is noted in [13] that an elementary pivot X corresponds to either a loop, $X = \{u\} \in E(G)$, or to an edge, $X = \{u, v\} \in E(G)$, where (distinct) vertices u and v are both non-loops. Thus for $Y \in \mathcal{M}_G$, if $G[Y]$ has elementary pivot X_1, then $Y \setminus X_1 = Y \oplus X_1 \in \mathcal{M}_{G*X_1}$. By iterating this argument, each $Y \in \mathcal{M}_G$ can be partitioned $Y = X_1 \cup \cdots \cup X_n$ such that $G*Y = G*(X_1 \oplus \cdots \oplus X_n) = (\cdots(G*X_1)\cdots * X_n)$ is a composition of elementary pivots. Consequently, a direct definition of the elementary pivots on graphs G is sufficient to define the (general) pivot operation.

The elementary pivot $G * \{u\}$ on a loop $\{u\}$ is called *local complementation*. It is the graph obtained from G by complementing the edges in the neighbourhood $N_G(u) = \{v \in V \mid \{u, v\} \in E(G), u \neq v\}$ of u in G: for each $v, w \in N_G(u)$, $\{v, w\} \in E(G)$ iff $\{v, w\} \notin E(G * \{u\})$, and $\{v\} \in E(G)$ iff $\{v\} \notin E(G * \{u\})$ (the case $v = w$). The other edges are left unchanged.

We now recall *edge complementation* $G * \{u, v\}$ on an edge $\{u, v\}$ between non-loop vertices. For a vertex x consider its closed neighbourhood $N'_G(x) = N_G(x) \cup \{x\}$. The edge $\{u, v\}$ partitions the vertices of G connected to u or v into three sets $V_1 = N'_G(u) \setminus N'_G(v)$, $V_2 = N'_G(v) \setminus N'_G(u)$, $V_3 = N'_G(u) \cap N'_G(v)$. Note that $u, v \in V_3$.

The graph $G * \{u, v\}$ is constructed by “toggling” all edges between different V_i and V_j: for $\{x, y\}$ with $x \in V_i$ and $y \in V_j$ $(i \neq j)$: $\{x, y\} \in E(G)$ iff $\{x, y\} \notin E(G * \{u, v\})$, see Figure 1. The other edges remain unchanged. Note that, as a result of this operation, the neighbours of u and v are interchanged.

[1] The explicit formulation of case $X \oplus \{x\} \in D$ is often omitted in the definition of delta-matroids. It is then understood that y may be equal to x and $\{x, x\} = \{x\}$. To avoid confusion will not use this convention here.

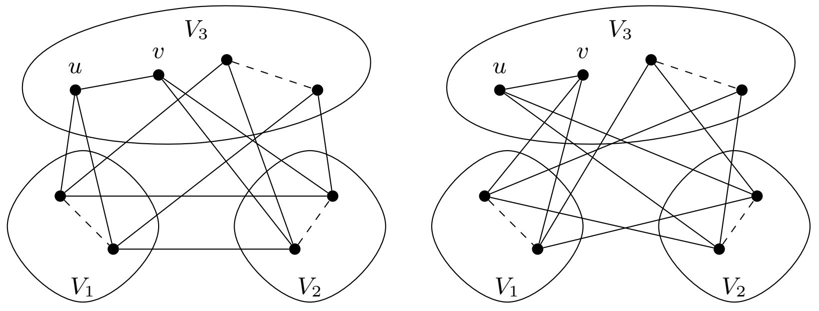


Fig. 1. Pivoting $\{u, v\}$ in a graph. Connection $\{x, y\}$ is toggled iff $x \in V_i$ and $y \in V_j$ with $i \neq j$. Note u and v are connected to all vertices in V_3, these edges are omitted in the diagram. The operation does not affect edges adjacent to vertices outside the sets V_1, V_2, V_3, nor does it change any of the loops.

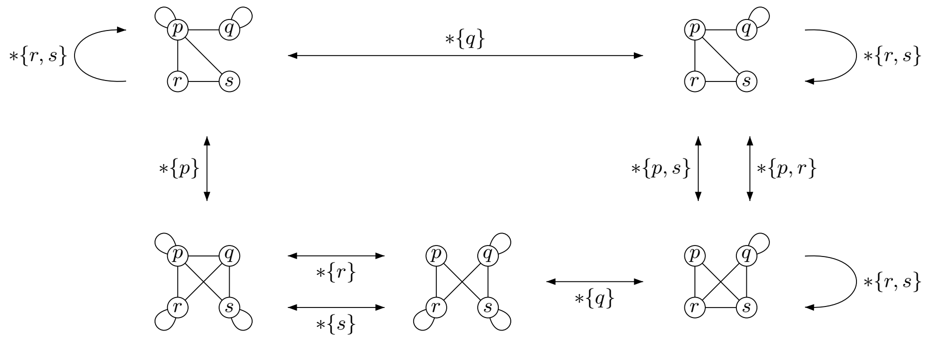


Fig. 2. The orbit of G of Example 2 under pivot. Only the elementary pivots are shown.

Example 2. Let G be the graph depicted in the upper-left corner of Figure 2. Graph G corresponds to $\mathcal{M}_G = (\{p, q, r, s\}, D_G)$, where

$$D_G = \{\varnothing, \{p\}, \{q\}, \{p, r\}, \{p, s\}, \{r, s\}, \{p, q, r\}, \{p, q, s\}, \{p, r, s\}, \{q, r, s\}\}.$$

For example, $\{p, r\} \in D_G$ since $\det(G[\{p, r\}]) = \det \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = 1$. The orbit of G under pivot as well as the applicable elementary pivots (i.e., local and edge complementation) are shown in Figure 2. For example, $G * \{p, q, r\}$ is shown on the lower-right in the same figure. Note that $D_G * \{p, q, r\} =$

$$\{\varnothing, \{q\}, \{p, r\}, \{p, s\}, \{q, r\}, \{q, s\}, \{r, s\}, \{p, q, r\}, \{p, q, s\}, \{q, r, s\}\}$$

indeed corresponds to $G * \{p, q, r\}$. □

4 Unifying Pivot and Loop Complementation

We now introduce a class of operations on set systems. It turns out that this class contains both the pivot and (a generalization of) loop-complementation. Each operation is a linear transformation, where the input and output vectors indicate the presence (or absence) of sets Z and $Z - \{j\}$ in the original and resulting set systems.

Definition 3. *Let $M = (V, D)$ be a set system, and let α be a 2×2-matrix over $\mathbb{F}_2$. We define, for $j \in V$, the* vertex flip *α of M on j, denoted by $M\alpha^j = (V, D')$, where, for all $Z \subseteq V$ with $j \in Z$, the membership of Z and $Z - \{j\}$ in D' is determined as follows:*

$$\alpha \,(Z \in D, Z - \{j\} \in D)^T = (Z \in D', Z - \{j\} \in D')^T.$$

In the above definition, we regard the elements of the vectors as Boolean values, e.g., the expression $Z \in D$ obtains either true (1) or false (0). To be more explicit, let $\alpha = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$. Then we have for all $Z \subseteq V$, $Z \in D'$ iff

$$\begin{cases} (a_{11} \wedge Z \in D) \oplus (a_{12} \wedge Z - \{j\} \in D) & \text{if } j \in Z \\ (a_{21} \wedge Z \cup \{j\} \in D) \oplus (a_{22} \wedge Z \in D) & \text{if } j \notin Z \end{cases}.$$

Note that in the above statement we may replace both $Z \cup \{j\} \in D$ and $Z - \{j\} \in D$ by $Z \oplus \{j\} \in D$ as in the former we have $j \notin Z$ and in the latter we have $j \in Z$. Thus, the operation α^j decides whether or not set Z is in the new set system, based on the fact whether or not Z and $Z \oplus \{j\}$ belong to the original system.

Note that if α is the identity matrix, then α^j is simply the identity operation. Moreover, with $\alpha_* = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ we have $M\alpha_*^j = M * \{j\}$, the pivot operation on a single element j.

By definition, a composition of vertex flips on the same element corresponds to matrix multiplication. The following lemma shows that vertex flips on different elements commute.

Lemma 4. *Let M be a set system over V, and let $j, k \in V$. We have that $(M\alpha^j)\beta^j = M(\beta\alpha)^j$, where $\beta\alpha$ denotes matrix multiplication of β and α. Moreover $(M\alpha^j)\beta^k = (M\beta^k)\alpha^j$ if $j \neq k$.*

To simplify notation, we assume left associativity of the vertex flip, and write $M\varphi_1\varphi_2 \cdots \varphi_n$ to denote $(\cdots((M\varphi_1)\varphi_2)\cdots)\varphi_n$, where $\varphi_1\varphi_2 \cdots \varphi_n$ is a sequence of vertex flip operations applied to set system M. Hence, the pivot operation as a special case of the vertex flip is also written in the simplified notation. We carry this simplified notation over to graphs G.

Due to commutative property shown in Lemma 4 we (may) define, for a set $X = \{x_1, \ldots, x_n\} \subseteq V$, $M\alpha^X = M\alpha^{x_1}\alpha^{x_2} \cdots \alpha^{x_n}$, where the result is independent on the order in which the operations are applied. Moreover, if α is of order 2 (i.e., $\alpha\alpha$ is the identity matrix), then $(M\alpha^X)\alpha^Y = M\alpha^{X \oplus Y}$.

Now consider $\alpha_+ = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$. The matrices α_+ and α_* given above generate the group $\mathrm{GL}_2(\mathbb{F}_2)$ of 2×2 matrices with non-zero determinant. In fact $\mathrm{GL}_2(\mathbb{F}_2)$ is isomorphic to the group $S_3 = \{1, a, b, c, f, g\}$ of permutations of three elements, where 1 is the identity, a, b, and c are the elements of order 2, and f and g are the elements of order 3. The matrices α_+ and α_* are both of order 2 and we may identify them with any two (distinct) elements of S_3 of order 2. The generators satisfy the relations $\alpha_+^2 = 1$, $\alpha_*^2 = 1$, and $(\alpha_*\alpha_+)^3 = 1$.

As, by Lemma 4, vertex flips on j and k with $j \neq k$ commute, we have that the vertex flips form the group $(S_3)^V$ of functions $f : V \to S_3$ where composition/multiplication is point wise: $(fg)(j) = f(j)g(j)$ for all $j \in V$. Note that by fixing a linear order of V, $(S_3)^V$ is isomorphic to $(S_3)^n$ with $n = |V|$, the direct product of n times group S_3. The vertex flips form an action of $(S_3)^V$ on the family of set systems over V.

5 Loop Complementation and Set Systems

For graphs G, the loop-complementation operation on a set of vertices $X \subseteq V(G)$ removes loops from the vertices of X when present in G and adds loops to vertices of X when not present in G. This can be formalized using adjacency matrices as follows. For a $V \times V$-matrix A and $X \subseteq V$, we obtain the $V \times V$-matrix $A + X$ by adding 1 to each diagonal element a_{xx}, $x \in X$ of A. Clearly $(A + X) + Y = A + (X \oplus Y)$ for $X, Y \subseteq V$.

In this section we focus on vertex flips of matrix α_+ (defined in the previous section). We show that this operation is a generalization to set systems of loop complementation for graphs and matrices (cf. Lemma 6). Consequently, we will call it *loop complementation* as well.

Let $M = (V, D)$ be a set system and $j \in V$. We denote $M\alpha_+^j$ by $M + \{j\}$. Hence, we have $M + \{j\} = (V, D')$ where, for all $Z \subseteq V$, $Z \in D'$ iff 1) $(Z \in D) \oplus (Z - \{j\} \in D)$ if $j \in Z$, and 2) $Z \in D$ if $j \notin Z$.

The definition of loop complementation can be reformulated as follows: $D' = D \oplus \{X \cup \{j\} \mid X \in D, j \notin X\}$.

Example 5. Let $V = \{1, 2, 3\}$. We have $(V, \{\varnothing, \{1\}, \{1, 2\}, \{3\}, \{1, 2, 3\}\}) + \{3\}$ $= (V, \{\varnothing, \{1\}, \{1, 2\}, \{3\}, \{1, 2, 3\}\} \oplus \{\{3\}, \{1, 3\}, \{1, 2, 3\}\}) = (V, \{\varnothing, \{1\}, \{1, 2\}, \{1, 3\}\})$.

We denote, for $X \subseteq V$, $M\alpha_+^X$ by $M + X$. Moreover, as α_+ is of order 2, we have, similar to the pivot operation, $(M + X) + Y = M + (X \oplus Y)$.

The next result states that indeed $M + X$ is the set system generalization of $A + X$ on matrices.

Lemma 6. *Let A be a symmetric $V \times V$-matrix over $\mathbb{F}_2$ and $X \subseteq V$. Then $\mathcal{M}_{A+X} = \mathcal{M}_A + X$.*

Surprisingly, this natural definition of loop complementation on set systems is not found in the literature.

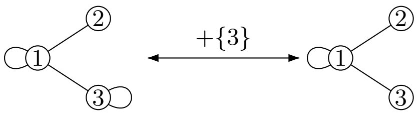


Fig. 3. Loop complementation on a graph from Example 7

Example 7. The set system $M = (\{1,2,3\}, \{\varnothing, \{1\}, \{1,2\}, \{3\}, \{1,2,3\}\})$ of Example 5 has a graph representation: $M = \mathcal{M}_G$ for graph G given on the left-hand side in Figure 3. Notice that $M + \{3\} = (\{1,2,3\}, \{\varnothing, \{1\}, \{1,2\}, \{1,3\}\})$ corresponds to graph $G + \{3\}$ given on the right-hand side in the figure.

While for a set system the property of being a delta-matroid is closed under pivot, the next example shows that it is *not* closed under loop complementation.

Example 8. Let $V = \{1,2,3\}$ and $M = (V,D)$ with $D = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,2\}, \{2,3\}, \{3,1\}\}$ be a set system. It is shown in [6, Section 3] that M is a delta-matroid without graph representation. Consider $\{1\} \subseteq V$. Then $M + \{1\} = (V, D')$ with $D' = \{\varnothing, \{2\}, \{3\}, \{2,3\}, \{1,2,3\}\}$ is not a delta-matroid: for $X = \varnothing, Y = \{1,2,3\} \in D'$, and $x = 1 \in X \oplus Y$, we have $X \oplus \{x\} = \{1\} \notin D'$ and there is no $y \in X \oplus Y$ such that $X \oplus \{x,y\} \in D'$.

6 Compositions of Loop Complementation and Pivot

In this section we study sequences of loop complementation and pivot operations. As we may consider both operations as vertex flips, we obtain in a straightforward way general equalities involving loop complementation and pivot.

Theorem 9. *Let M be a set system over V and $X \subseteq V$. Then $M + X * X + X = M * X + X * X$.*

Note also that, again by the commutative property of vertex flip, we have for $X, Y \subseteq V$ with $X \cap Y = \varnothing$, $M * X + Y = M + Y * X$.

Let us denote $\alpha_{\bar{*}} = \alpha_+ \alpha_* \alpha_+$ and denote, for $X \subseteq V$, $M\alpha_{\bar{*}}^X$ by $M \bar{*} X$. We will call the $\bar{*}$ operation the *dual pivot*. As α_+ is of order 2, we have, similar to the pivot operation and loop complementation, $(M \bar{*} X) \bar{*} Y = M \bar{*} (X \oplus Y)$. The dual pivot together with pivot and loop complementation correspond precisely to the elements of order 2 in S_3. The dual pivot defined here is readily shown to be the set system equivalent of dual pivot for graphs as considered in [8].

We now obtain a normal form for sequences of pivots and loop complementations.

Theorem 10. *Let M be a set system over V, and let φ be any sequence of pivot and loop complementation operations on elements in V. We have that $M\varphi = M + X * Y + Z$ for some $X, Y, Z \subseteq V$ with $X \subseteq Y$.*

Recall that pivot operations can be decomposed into elementary pivot operations. Hence, the normal form of Theorem 10 is equally valid for any sequence φ of local, edge, and loop complementation operations.

The central interest of this paper is to study compositions of pivot and loop complementation on graphs. As explained in Section 3, the pivot operation for set systems and graphs coincide, i.e., $\mathcal{M}_{G*X} = \mathcal{M}_G * X$, and we have taken care that the same holds for loop complementation, cf. Lemma 6. Hence results that hold for set systems in general, like Theorem 9, subsume the special case that the set system M represents a graph (i.e., $M = \mathcal{M}_G$ for some graph G) — recall that the injectivity of $\mathcal{M}_{(\cdot)}$ allows one to view the family $\mathcal{G}$ of graphs (over V) as a subset of the family of set systems (over V). We only need to make sure that we "stay" in $\mathcal{G}$, i.e., by applying a pivot or loop complementation operation to $\mathcal{M}_G$ we obtain a set system M such that $M = \mathcal{M}_{G'}$ for some graph G'. For loop complementation this will always hold, however care must be taken for pivot as $\mathcal{M}_G * X$, which is defined for all $X \subseteq V$, only represents a graph if $\det G[X] = 1$. Hence when restricting a general result (on pivot or local complementation for set systems) to graphs, we add the condition of applicability of the operations.

It is useful to explicitly state Theorem 9 restricted to graphs. This is a fundamental result for pivots on graphs (or, equivalently, symmetric matrices over $\mathbb{F}_2$) not found in the extensive literature on pivots. We will study some of its consequences in the remainder of this paper.

Corollary 11. *Let G be a graph and $X \subseteq V$. Then $G+X*X+X = G*X+X*X$ when both sides are defined.*

In the particular case of Corollary 11 it is not necessary to verify the applicability of both sides: it turns out that the applicability of the right-hand side implies the applicability of the left-hand side of the equality.

Lemma 12. *Let G be a graph and $X \subseteq V$. If $G * X + X * X$ is defined, then $G + X * X + X$ is defined.*

The reverse implication of Lemma 12 does not hold: take, e.g., G to be the connected graph of two vertices with each vertex having a loop.

Corollary 11 can also be proven directly using Equality (1), i.e., the partial inverse property of pivots. In fact one can explicitly express the result of the dual pivot applied to a (not-necessarily symmetric) matrix over $\mathbb{F}_2$: $A(x_1, y_1)^T = (x_2, y_2)^T$ iff $(A \bar{*} X)(x_1 + x_2, y_1)^T = (x_2, y_2)^T$ (if $A + X * X$ is defined), where x_1 and x_2 correspond to the X-coordinates of the vectors.

7 Consequences for Simple Graphs

In this section we consider simple graphs, i.e., undirected graphs without loops or parallel edges. Local complementation was first studied on simple graphs [10]: local complementation $*\{u\}$ on a vertex u complements the edges in the neighbourhood of u, thus it is the same operation as for graphs (loops allowed) except that applicability is not dependent on the presence of a loop on u, and neither are loops added or removed in the neighbourhood. Also edge complementation $*\{u, v\}$ on edge $\{u, v\}$ for simple graphs is defined as for graphs, inverting certain sets of edges, cf. Figure 1, but again the absence of loops is no (explicit) requirement for applicability. The "curious" identity $*\{u, v\} = *\{u\}*\{v\}*\{u\}$ for simple

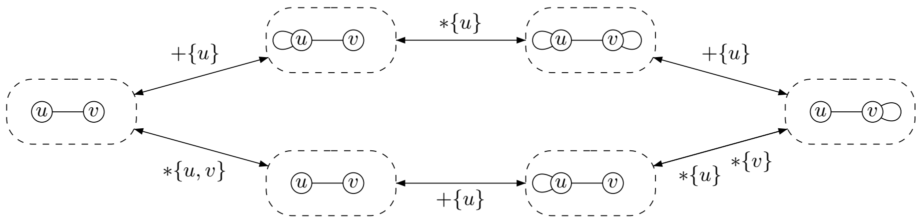


Fig. 4. Verification of applicability of $*\{u,v\}+\{u\}*\{u\}*\{v\}+\{u\}*\{u\}+\{u\}$ to any graph F having an edge $\{u,v\}$ with both u and v non-loop vertices

graphs shown in [5, Corollary 8.2] and found in standard text books, see, e.g., [14, Theorem 8.10.2], can be proven by a straightforward (but slightly tedious) case analysis involving u, v and all possible combinations of their neighbours. Here it is obtained, cf. Proposition 13, as a consequence of Theorem 9.

Proposition 13. *Let G be a simple graph having an edge $\{u,v\}$. We have $G*\{u,v\} = G*\{u\}*\{v\}*\{u\} = G*\{v\}*\{u\}*\{v\}$.*

Proof. Let M be a set system, and u and v two elements from its domain. Define $\varphi = *\{u,v\}+\{u\}*\{u\}*\{v\}+\{u\}*\{u\}+\{u\}$. Recall that for set systems we have $*\{u,v\} = *\{u\}*\{v\}$ and that the operations on different elements commute, e.g. $*\{v\}+\{u\} = +\{u\}*\{v\}$. We have therefore $\varphi = *\{u,v\}+\{u\}*\{u\}*\{v\}+\{u\}*\{u\}+\{u\} = *\{u\}*\{v\}+\{u\}*\{u\}*\{v\}+\{u\}*\{u\}+\{u\} = *\{u\}+\{u\}*\{u\}+\{u\}*\{u\}+\{u\} = \mathrm{id}$, where in the last equality we used Theorem 9. Therefore, $M\varphi = M$ for any set system M having u and v in its domain.

Hence, any graph F for which φ is applicable to F, we have $F\varphi = F$. Assume now that F is a graph (allowing loops) having an edge $\{u,v\}$ where both u and v do not have a loop. By Figure 4 we see that φ is applicable to F, and therefore $F\varphi = F$.

Now, modulo loops, i.e., considering simple graphs G, we no longer worry about the presence of loops, and we may omit the loop complementation operations from φ. Hence $*\{u,v\}*\{u\}*\{v\}*\{u\}$ is the identity on simple graphs, and therefore $*\{u,v\} = *\{u\}*\{v\}*\{u\}$. By symmetry of the $*\{u,v\}$ operation we also have that $*\{u,v\} = *\{v\}*\{u\}*\{v\}$. □

For set systems we have the decomposition $*\{u,v\} = *\{u\}*\{v\}$, whereas for for simple graphs the decomposition of an edge pivot into vertex pivots takes the form $*\{u,v\} = *\{u\}*\{v\}*\{u\}$. The rationale behind this last equality is hidden, as in fact the equality $*\{u,v\} = +\{u\}*\{u\}*\{v\}+\{u\}*\{u\}+\{u\}$ is demonstrated for graphs (loops allowed) (see the proof of Proposition 13). The fact that the equality of Proposition 13 does not hold for graphs (with loops allowed) is a consequence of the added requirement of applicability of the operations. Applicability depends on the presence of loops, and it is curious that loops are necessary to understand the operations for simple graphs (which are loopless by definition)!

The presented method allows one to discover many more curious equalities. Start with an identity for set systems, involving pivot and loop complementation. Then show applicability for (general) graphs for the sequence of operations. Finally drop the loop complementation operations to obtain an identity for simple graphs. We illustrate this by stating another equality, this time for three vertices that induce a complete graph.

Corollary 14. *Let G be a simple graph, and let $u, v, w \in V(G)$ be such that the subgraph of G induced by $\{u, v, w\}$ is a complete graph. Then $G(*\{u\} * \{v\} * \{w\})^2 = G * \{v\}$.*

In the next result, Theorem 15, we go back-and-forth between the notions of simple graph and graph. To avoid confusion, we explicitly formalize these transitions. For a simple graph G, we define $i(G)$ to be G regarded as a *graph* (i.e., symmetric matrix over $\mathbb{F}_2$) having no loops. Similarly, for graph F, we define $\pi(F)$ to be the simple graph obtained from F by removing the loops.

As a consequence of Theorem 10, the following result characterizes the effect of sequences of local complementations on simple graphs.

Theorem 15. *Let G be a simple graph, and let φ be a sequence of local complementation operations applicable to G. Then $G\varphi = \pi(i(G) + X * Y)$ for some $X, Y \subseteq V$ with $X \subseteq Y$.*

Conversely, for graph F, if $F + X * Y$ is defined for some $X, Y \subseteq V$, then there is a sequence φ of local complementation operations applicable to $\pi(F)$ such that $\pi(F)\varphi = \pi(F + X * Y)$.

8 Discussion

We have considered loop complementation $+X$, pivot $*X$, and dual pivot $\bar{*}X$ on both set systems and graphs, and have shown that they can be seen as elements of order 2 in the permutation group S_3. This group structure, in addition to the commutation property in Lemma 4, leads to identity $(+X * X)^3 = \mathrm{id}$, cf. Theorem 9, and to a normal form w.r.t. sequences of pivots and loop complementation, cf. Theorem 10.

Moreover, we obtain as a special case "modulo loops" a classic relation involving local and edge complementation on simple graphs, cf. Proposition 13. Other new relations may be easily deduced, cf. Corollary 14.

References

1. Aigner, M., van der Holst, H.: Interlace polynomials. Linear Algebra and its Applications 377, 11–30 (2004)
2. Arratia, R., Bollobás, B., Sorkin, G.B.: The interlace polynomial: a new graph polynomial. In: SODA 2000: Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 237–245. Society for Industrial and Applied Mathematics, Philadelphia (2000)

3. Arratia, R., Bollobás, B., Sorkin, G.B.: The interlace polynomial of a graph. Journal of Combinatorial Theory, Series B 92(2), 199–233 (2004)
4. Bouchet, A.: Representability of Δ-matroids. In: Proc. 6th Hungarian Colloquium of Combinatorics, Colloquia Mathematica Societatis János Bolyai, vol. 52, pp. 167–182. North-Holland, Amsterdam (1987)
5. Bouchet, A.: Graphic presentations of isotropic systems. Journal of Combinatorial Theory, Series B 45(1), 58–76 (1988)
6. Bouchet, A., Duchamp, A.: Representability of Δ-matroids over $GF(2)$. Linear Algebra and its Applications 146, 67–78 (1991)
7. Brijder, R., Hoogeboom, H.J.: The group structure of pivot and loop complementation on graphs and set systems, arXiv:0909.4004 (2009)
8. Brijder, R., Hoogeboom, H.J.: Maximal pivots on graphs with an application to gene assembly, arXiv:0909.3789 (submitted 2009)
9. Cottle, R.W., Pang, J.-S., Stone, R.E.: The Linear Complementarity Problem. Academic Press, San Diego (1992)
10. de Fraysseix, H.: Local complementation and interlacement graphs. Discrete Mathematics 33(1), 29–35 (1981)
11. Van den Nest, M., Dehaene, J., De Moor, B.: Graphical description of the action of local clifford transformations on graph states. Physical Review A 69(2), 022316 (2004)
12. Ehrenfeucht, A., Harju, T., Petre, I., Prescott, D.M., Rozenberg, G.: Computation in Living Cells – Gene Assembly in Ciliates. Springer, Heidelberg (2004)
13. Geelen, J.F.: A generalization of Tutte's characterization of totally unimodular matrices. Journal of Combinatorial Theory, Series B 70, 101–117 (1997)
14. Godsil, C., Royle, G.: Algebraic Graph Theory. Springer, Heidelberg (2001)
15. Oum, S.: Rank-width and vertex-minors. Journal of Combinatorial Theory, Series B 95(1), 79–100 (2005)
16. Tsatsomeros, M.J.: Principal pivot transforms: properties and applications. Linear Algebra and its Applications 307(1-3), 151–165 (2000)
17. Tucker, A.W.: A combinatorial equivalence of matrices. In: Combinatorial Analysis, Proceedings of Symposia in Applied Mathematics, vol. X, pp. 129–140. American Mathematical Society, Providence (1960)

Revisiting the Minimum Breakpoint Linearization Problem

Laurent Bulteau[1,2], Guillaume Fertin[2], and Irena Rusu[2]

[1] École Normale Supérieure, 45 rue d'Ulm, 75000 Paris, France
[2] Laboratoire d'Informatique de Nantes-Atlantique (LINA), UMR CNRS 6241 Université de Nantes, 2 rue de la Houssinière, 44322 Nantes Cedex 3, France
`Laurent.Bulteau@ens.fr`, {`Guillaume.Fertin`,`Irena.Rusu`}`@univ-nantes.fr`

Abstract. The gene order on a chromosome is a necessary data for most comparative genomics studies, but in many cases only partial orders can be obtained by current genetic mapping techniques. The Minimum Breakpoint Linearization Problem aims at constructing a total order from this partial knowledge, such that the breakpoint distance to a reference genome is minimized. In this paper, we first expose a flaw in two algorithms formerly known for this problem [4,2]. We then present a new modeling for this problem, and use it to design three approximation algorithms, with ratios resp. $O(\log(k)\log\log(k))$, $O(\log^2(|X|))$ and m^2+4m-4, where k is the optimal breakpoint distance we look for, $|X|$ is upper bounded by the number of pair of genes for which the partial order is in contradiction with the reference genome, and m is the number of genetic maps used to create the input partial order.

1 Introduction

In a number of comparative genomics algorithms, a full knowledge of the order of the genes on the chromosomes for the species under study is required. However, we have access to a limited number of fully sequenced genomes, and for other species, we only have genetic maps, in which there remains uncertainties in the gene order. Hence, the problem of inferring a total order, compatible with the partial knowledge on these genetic maps and optimizing some objective function, is a first step to study nonetheless all genomes. In the past few years, growing attention has been given to this problem, in which the objective function is an evolutionary distance to a reference genome (*e.g.* number of rearrangements [7], reversal [6,4], breakpoint [4,1,2], or common intervals [1] distance).

In this paper, we focus on the MBL problem, which aims at finding a linearization of a partial order while minimizing the breakpoint distance to a reference genome. In [4] and [2], the study of this problem uses the construction of a special graph, the adjacency-order graph, which leads to respectively a heuristic and an approximation algorithm (whose ratio depends on m, the number of genetic maps used to construct the studied genome). However, we have detected a flaw in this construction, which makes both above mentioned algorithms invalid on general data. Thus, in this paper, we define a new type of adjacency-order

© Springer-Verlag Berlin Heidelberg 2010

graphs, give its construction, and show that it is effective to solve the MBL problem. This renewed approach allows us to use general graph theory results [3] to obtain new approximation algorithms for MBL. Moreover, we also achieve an $O(m^2)$-approximation, in the same spirit as was done in [2].

To describe the MBL problem, let a genome be represented by a partial order Π over a given set $\Sigma = \{1, \ldots, n\}$ of markers. A *linearization* of Π is a total order (or a permutation) $\pi = \pi(1) \cdot \pi(2) \cdot \ldots \cdot \pi(n)$ on Σ, such that, for all markers i, j, if $i <_\Pi j$, then $i <_\pi j$ (alternatively, $\pi^{-1}(i) < \pi^{-1}(j)$, or i precedes j in the permutation π). In that case, π is said to be *compatible* with Π. An *interval* I of π is a list of successive values from π, that is $I = \pi(h) \cdot \pi(h+1) \cdot \ldots \cdot \pi(l)$, with $1 \leq h \leq l \leq n$, from π. For any such interval, its length L is defined as $l - h + 1$. An *adjacency* in the total order π is an interval of length $L = 2$. The *breakpoint distance* $d_B(\pi_1, \pi_2)$ between two total orders π_1 and π_2 (over the same set Σ) is defined by the total number of adjacencies in π_1 which are not adjacencies in π_2. We call Id_n the identity permutation over $\Sigma = \{1, \ldots, n\}$.

The Minimum Breakpoint Linearization Problem, in its optimization formulation, can be defined as follows:

Problem : MBL
Input : A partial order Π.
Output : A linearization π of Π that minimizes $k = d_B(\pi, Id_n)$.

Fig. 1a shows an example of partial order Π, that yields four optimal linearizations (Fig. 1c) satisfying $d_B(\pi, Id_n) = 3$ for each linearization π.

It is worth noting that the input partial order is, in practice, obtained by combining a limited number m of *genetic maps* [5,7]. A genetic map consists of an ordered list of blocks $B_1, B_2, \ldots, B_q$, each of which is an unordered list of markers, i.e. any two markers from the same block are incomparable. The blocks $B_1, B_2, \ldots, B_q$ induce a partial order Π as follows: for any $a \in B_i$ and $b \in B_j$, $a <_\Pi b$ iff $i < j$. Note that it is always assumed that combining two or more genetic maps never creates conflicts.

The Minimum Breakpoint Linearization Problem, based on the genome rearrangement problem defined by Zheng and Sankoff [7], was studied independently in [1] and [4] (we note that in the latter, the problem is denoted as PBD, and deals with two partial orders instead of one partial order and one total order). In [1], Blin et al. prove that MBL is **NP**-hard and give two types of algorithms for solving MBL: (i) a heuristic and (ii) an exact, thus exponential-time, algorithm based on dynamic programming. Moreover, this last algorithm is efficient in the specific case where input genomes are created from a bounded number m of gene maps, each with a bounded width. In [4], Fu and Jiang give an (independent) **NP**-hardness proof, and present the construction of the adjacency-order graph $\mathcal{G}_\Pi$ of a partial order Π (Π being represented as a DAG, called DAG(Π)). Their central theorem, claiming that "*All the possible common adjacencies in an acyclic adjacency-order graph $\mathcal{G}_\Pi$ could always co-exist in some linearization of DAG(Π)*", is used in a heuristic they provide for the problem, and is also used by Chen and Cui [2] to obtain an $\frac{m^2+m}{2}$-approximation algorithm. However, as shown in Theorem 2, the above mentioned theorem from [4] is false, and consequently both those algorithms are invalid for the general MBL Problem.

The paper is organized as follows. In Section 2, we point out the **APX**-hardness of MBL and give our counter-example to the central theorem in [4]. Section 3 is devoted to the definition of a new adjacency-order graph, which is used in Section 4 to show that solving MBL may be reduced to solving a variant of the well-known Feedback Vertex Set problem. Section 5 presents three approximation algorithms, two of which are based on state-of-the-art algorithms for the variant of Feedback Vertex Set we are interested in, whereas the third is specific to partial orders created from genomic maps, and has a ratio depending on the number m of those genomic maps. Section 6 is the conclusion.

Due to space constraints, most of the proofs have been omitted from this paper.

2 Revisiting Previous Works

In this section, we focus on previous works: we first show that adapting the **NP**-hardness proof for MBL from [1] leads to an **APX**-hardness proof. We then give a counterexample to Theorem 1 from [4], which implies, among others, that the approximation algorithm from [2] is invalid.

Theorem 1. *The* MBL *problem is* **APX**-*hard.*

See Theorem 1 in [1], where the reduction is in fact an L-reduction from Maximum Independent Set (MIS) to MBL, if we restrict MIS to cubic graphs.

The following theorem refers to the definition of *adjacency-order graph* (written $\mathcal{G}_\Pi$) given in [4]. The construction of this graph for any Π is not reported here, instead it is described for the directed acyclic graph given in Fig. 1a.

Theorem 2. *All the adjacencies appearing in the adjacency-order graph $\mathcal{G}_\Pi$ (as defined in [4]) of a DAG Π may not always coexist in a linearization of Π, even if $\mathcal{G}_\Pi$ is acyclic.*

Proof. Consider the directed acyclic graph $DAG(\Pi)$ obtained from the following partial order (see Fig. 1a):

$$\Pi : \quad 4 < 1 < 2 < 3; \quad 1 < 5; \quad 6 < 3$$

$$\begin{tikzcd}
 & & 5 & \\
4 \arrow[r] & 1 \arrow[r] \arrow[ur] & 2 \arrow[r] & 3 \\
 & & 6 \arrow[ur] &
\end{tikzcd}$$

(a) DAG(Π) represents the partial order Π

$$\begin{tikzcd}
 & 5\cdot 6 \arrow[dr] & \\
1\cdot 2 \arrow[ur] \arrow[rr] & & 2\cdot 3
\end{tikzcd}$$

(b) $\mathcal{G}_\Pi$, as defined in [4]

$4 \cdot 1 \cdot \underline{5 \cdot 6} \cdot \underline{2 \cdot 3}$

$4 \cdot \underline{1 \cdot 2} \cdot \underline{5 \cdot 6} \cdot 3$

$6 \cdot 4 \cdot \underline{1 \cdot 2 \cdot 3} \cdot 5$

$4 \cdot 6 \cdot \underline{1 \cdot 2 \cdot 3} \cdot 5$

(c) Optimal linearizations of Π

Fig. 1. Counterexample of the main theorem from [4]

Here, there are only three possible common adjacencies between Π and Id_n: $1\cdot 2$, $2\cdot 3$ and $5\cdot 6$. Following the definitions proposed in [4], there are three arcs in the adjacency-order graph $\mathcal{G}_\Pi$ (Fig. 1b): one from $1\cdot 2$ to $2\cdot 3$ (because of the common marker), one from $1\cdot 2$ to $5\cdot 6$ (because $1 < 5$ in Π), and one from $5\cdot 6$ to $2\cdot 3$ (because $6 < 3$ in Π). In that case, $\mathcal{G}_\Pi$ is acyclic; however, its three adjacencies cannot coexist in any linearization of Π (see Fig. 1c: there can be at most two adjacencies in the same linearization). Thus Theorem 2 is proved, which contradicts Theorem 1 in [4]. □

3 Defining a New Adjacency-Order Graph G_Π

The direct consequence of Theorem 2 is that the adjacency-order graph defined in [4] cannot be exploited. Hence, we introduce here the construction of a new type of adjacency-order graph, whose main structural property will lead us to three different approximation algorithms.

This new adjacency-order graph G_Π contains both the features of the partial order Π and of the identity permutation Id_n. Some of the cycles in this graph will express the incompatibilities between the order Π and the permutation Id_n. In order to count or to bound the breakpoint distance between a linearization π of Π and Id_n, one has to identify the vertices in the adjacency-order graph needed to break all these conflict-cycles, and to count or bound their number. The MBL problem thus becomes a graph theory problem, which allows us either to use existing algorithms or to build-up new algorithms based on graphs.

Adjacency-order graph. Let $\Pi = (\Sigma, D)$ be a directed acyclic graph (DAG) representing a partial order over $\Sigma = \{1, \dots, n\}$ (see Fig. 2a), *i.e.* we write $i <_\Pi j$ iff there is a directed path from i to j in Π. We create a set W of vertices representing the adjacencies of the identity permutation Id_n by $W = \{i\cdot(i+1) \mid 1 \le i < n\}$. Finally, let $V = \Sigma \cup W$ (Fig. 2b). Note that, in the following, we will not distinguish the vertices of Σ and their corresponding integers (this will always be clear from the context). Moreover, the natural order $<$ over the integers is also used as an order over Σ. We now construct a set of arcs F (denoted by an arrow $\rightarrow$) in the following way:

$$\begin{aligned} F = \quad & \{i\cdot(i+1) \rightarrow i \mid 1 \le i < n\} \ \cup \ \{i\cdot(i+1) \rightarrow i+1 \mid 1 \le i < n\} \\ & \cup \{i \rightarrow i\cdot(i+1) \mid 1 \le i < n\} \ \cup \ \{i+1 \rightarrow i\cdot(i+1) \mid 1 \le i < n\} \end{aligned}$$

Each arc in F has one end in W and one end in Σ. We write $E = D \cup F$ (Fig. 2c) and we define the *adjacency-order* graph G_Π of Π by $G_\Pi = (V, E)$.

In G_Π, the arcs of D that go top-down (see Fig. 2c) intuitively show incompatibilities between the order in Π and the order in Id_n. We note $X[G_\Pi]$ (or only X, if there is no ambiguity) the set containing them, that is $X[G_\Pi] = \{i \rightarrow j \in D \mid i > j\}$. Now, every cycle containing an arc of X is called a *conflict-cycle*. In Theorem 4, we prove that the adjacencies involved in conflict-cycles are incompatible, so that we need to remove at least one adjacency from each of those cycles to obtain a linearization of Π. We also define a weight map $w[G_\Pi]$ on the vertices of G_Π, which associates 1 to each $u \in W$, and ∞ to $u \in \Sigma$.

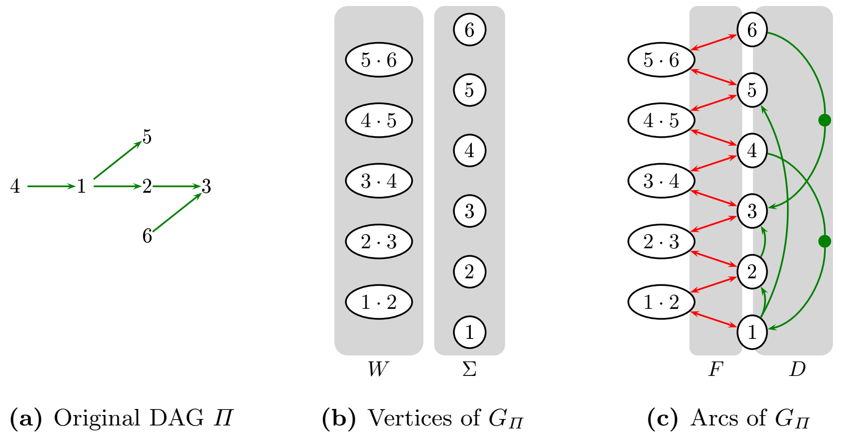


Fig. 2. Construction of an adjacency-order graph. The symmetric arcs in F are represented as double arrows. The arcs in X are marked with a large dot.

Notations. An arc between u and v is written $u \to v$, or $u \to_{E'} v$ if it belongs to some subset E'. A *path* P is a (possibly empty) sequence of arcs written $u \xrightarrow{P}{}^{*} v$, or $u \xrightarrow{P}{}^{*}_{E'} v$ if P uses only arcs from E'. A non-empty path Q is written with a $+$ sign: $u \xrightarrow{Q}{}^{+} v$. A *cycle* is a non-empty path $u \xrightarrow{\mathcal{C}}{}^{+} v$ with $v = u$.

Given in G_Π a path $P = v_0 \to v_1 \to \ldots \to v_\ell$, we use the following notations: $\ell(P) = \ell$ is the length of P, $V(P) = \{v_h \mid 0 \leq h \leq \ell\}$, $W(P) = V(P) \cap W$, $\Sigma(P) = V(P) \cap \Sigma$, $E(P) = \{v_h \to v_{h+1} \mid 0 \leq h < \ell\}$, $F(P) = E(P) \cap F$, $D(P) = E(P) \cap D$, $X(P) = E(P) \cap X$. A cycle $\mathcal{C}$ is said to be *simple* if all vertices v_h are distinct (except $v_0 = v_\ell$), which implies $\ell(\mathcal{C}) = |V(\mathcal{C})| = |E(\mathcal{C})|$.

The following property gives an insight on how conflict-cycles can appear in the adjacency-order graph. (It is not, however, used in our algorithms.)

Property 3. *Let $\mathcal{C}$ be a simple cycle with $|D(\mathcal{C})| \geq 2$. Then $\mathcal{C}$ is a conflict-cycle.*

4 Cutting All Conflict-Cycles in G_Π Is Enough

Now that we have defined how to construct G_Π starting from the input partial order Π, we turn to proving the main structural result of our paper: conflict-cycles contain all the conflicts between the partial order Π and the identity permutation Id_n (see Theorem 4). More precisely, when appropriate adjacencies in Id_n (identified as vertices in W) are given up, the remaining adjacency-order graph has no conflict-cycle and this condition is necessary and sufficient to obtain a linearization of Π that preserves all the remaining adjacencies in Id_n.

Theorem 4. *Let Π be a partial order, $G_\Pi = (V, E)$ its adjacency-order graph (with $V = \Sigma \cup W$ and $E = D \cup F$), and $W' \subseteq W$. Then there exists a total* *order π over Σ, compatible with Π, and containing every adjacency from W' iff $G_\Pi[W' \cup \Sigma]$ has no conflict-cycle.*

Proof. ($\Rightarrow$) Let π be a linearization of Π containing every adjacency of W'. The following lemma (of which the proof is omitted) will allow us to conclude by contradiction.

Lemma 5. *Let $P = v_1 \to v_2 \to \ldots \to v_\ell$ be a path with vertices in $W' \cup \Sigma$ such that (H1) the vertices v_i are pairwise distinct, (H2) $\ell \geq 2$, (H3) $v_1, v_\ell \in \Sigma$, and (H4) for any $1 \leq i < \ell$, $v_i \to v_{i+1} \in F$.*

Let $a = \min(v_1, v_\ell)$ and $b = \max(v_1, v_\ell)$. Then the sequence $a\cdot(a+1)\cdot(a+2)\cdot\ldots\cdot b$ is an interval of π. Moreover, $\Sigma(P) = \{a, \ldots, b\}$ and $W(P) = \{c\cdot(c+1) \mid a \leq c < b\}$.

We suppose, by contradiction, that there exists in $G_\Pi[W' \cup \Sigma]$ a cycle $\mathcal{C} = v_0 \to v_1 \to v_2 \to \ldots \to v_\ell = v_0$ containing an arc from X (*e.g.*, $v_0 \to_X v_1$). Wlog, we may assume that $\mathcal{C}$ is simple (otherwise, there exists a simple sub-cycle of $\mathcal{C}$ that contains an arc from X). We distinguish two cases, depending on whether $v_0 \to_X v_1$ is the only arc in $D(\mathcal{C})$ or not.

First case: $v_0 \to v_1 \in X$ and for all i, $1 \leq i < \ell$, $v_i \to v_{i+1} \in F$ holds. In that case, we can directly use Lemma 5. Indeed, the path $v_1 \to v_2 \to \ldots \to v_\ell = v_0$ satisfies hypothesis H1 (by simplicity of $\mathcal{C}$), H2 (otherwise there would be a loop $v_0 \to v_0$ in X), H3 (since $v_\ell \to v_1 \in D$) and H4 (this is assumed in this first case). We also know, due to the fact that $v_0 \to_X v_1$, that $v_1 < v_\ell$. We can conclude that $v_1\cdot v_1 + 1\cdot\ldots\cdot v_\ell$ is an interval of π, so $v_1 <_\pi v_\ell = v_0$. This contradicts the fact that π is compatible with Π, since $v_0 <_\Pi v_1$.

Second case: Let $i_0 = 0, i_1, \ldots, i_{h-1}, i_h = \ell$ be the increasing sequence of indices such that $v_{i_j} \to v_{i_j+1} \in D$ for all j such that $0 \leq j < h$. Note that $h \geq 2$ and for all j, we have $v_{i_j} \in \Sigma$. Let us prove that for all $j < h$, the relation $v_{i_j} <_\pi v_{i_{j+1}}$ holds. The case where $i_{j+1} = i_j + 1$ is easy, since the arc $v_{i_j} \to_D v_{i_j+1}$ implies $v_{i_j} <_\Pi v_{i_{j+1}}$ (by construction of G_Π) and $v_{i_j} <_\pi v_{i_{j+1}}$ (since π is compatible with Π). Now, assume there are several arcs between v_{i_j} and $v_{i_{j+1}}$, *i.e.* $i_{j+1} = i_j + m$ with $m \geq 2$. We use Lemma 5 with the path $\mathcal{P}$ in F given by $v_{i_j+1} \to v_{i_j+2} \to \ldots \to v_{i_j+m}$. Path $\mathcal{P}$ satisfies the hypotheses H1, H2, H3 and H4 of the lemma, thus one of the sequences $v_{i_j+1}\cdot(v_{i_j+1}+1)\cdot\ldots\cdot v_{i_j+m}$ and $v_{i_j+m}\cdot(v_{i_j+m}+1)\cdot\ldots\cdot v_{i_j+1}$ is an interval of π. Note that v_{i_j} is a distinct vertex from $v_{i_{j+1}}$ (since $h \geq 2$), and from other vertices in the set $\Sigma(\mathcal{P})$ as well (since each of them is the source of an arc from F in $\mathcal{C}$, whereas v_{i_j} is the source of an arc from D in $\mathcal{C}$). Consequently, v_{i_j} cannot appear in either of the intervals $v_{i_j+1}\cdot(v_{i_j+1}+1)\cdot\ldots\cdot v_{i_j+m}$ and $v_{i_j+m}\cdot(v_{i_j+m}+1)\cdot\ldots\cdot v_{i_j+1}$ of π. As v_{i_j} precedes v_{i_j+1} in Π (and thus in π), we have $v_{i_j} <_\pi v_{i'}$ for all $i' \in [i_j+1, i_j+m]$, and particularly, $v_{i_j} <_\pi v_{i_{j+1}}$.

In conclusion, we have $v_{i_j} <_\pi v_{i_{j+1}}$ for all $j < h$ and $v_{i_h} = v_{i_0}$, a contradiction since there can be no cycle in the relation $<_\pi$. Hence, the subgraph $G_\Pi[W' \cup \Sigma]$ does not contain any conflict-cycle.

($\Leftarrow$) *(constructive proof)* We use the following method to construct a linearization π of Π containing all adjacencies of W', where the subgraph $G' =$

$G_\Pi[W' \cup \Sigma]$, is assumed to contain no conflict-cycle. We denote by $V_1, \dots V_k$ the strongly connected components of G', ordered by topological order (i.e., if $u, v \in V_i$, there exists a path from u to v ; moreover, if $u \in V_i$ and $v \in V_j$ and there exists a path $u \rightarrow^* v$ in G', then $i \leq j$). We sort the elements of each set $V_i \cap \Sigma$ in ascending order of integers, and obtain a sequence μ_i. The concatenation $\mu_1 \cdot \mu_2 \cdot \dots$ gives π, a total order over Σ. We now check that π contains every adjacency in W' and is compatible with Π.

Let $a \cdot (a+1) \in W'$. Vertices a and $a+1$ are in the same strong connected component V_i, because of the arcs $a \leftrightarrow a \cdot (a+1) \leftrightarrow a+1$. Those two elements are obviously consecutive in the corresponding μ_i, and appear as an adjacency in π. By contradiction, assume now that there exist two distinct elements $a, b \in \Sigma$ such that $a <_\pi b$ and $b <_\Pi a$. We denote by i and j the indices such that $a \in V_i$ and $b \in V_j$. Since $a <_\pi b$, we have $i \leq j$, and since $b <_\Pi a$, there exists a path $b \overset{P_1}{\rightarrow}{}^+_D a$ in (Σ, D). Therefore, in G', we have $i \geq j$. We thus deduce that $i = j$, and therefore a and b share the same strong connected component. This means that there also exists a path P_2 from a to b in G'. Hence, we have a cycle $b \overset{P_1}{\rightarrow}{}^+ a \overset{P_2}{\rightarrow}{}^+ b$, which cannot be a conflict-cycle, thus those paths do not use any arc from X. The latter is in particular true along P_1, which implies $b < a$, since each arc $u \rightarrow v$ in $D - X$ is such that $u < v$. On the other hand, a appears before b in π, and therefore in μ_i, so $a < b$, a contradiction. Finally π is a feasible solution for MBL(Π), with at least $|W'|$ common adjacencies with the identity permutation Id_n. □

Since all vertices in $W - W'$ count for unconserved adjacencies (and thus define $d_B(\pi, Id_n)$), from Theorem 4 we directly get the following corollary.

Corollary 6. *The value k of an optimal solution of* MBL*(Π) is the minimum number of vertices one needs to delete in W to remove all conflict-cycles from G_Π.*

5 Three Approximation Algorithms for MBL

5.1 Two Approximation Algorithms Based on Subset-FVS

Our previous result implies that we have reduced the problem MBL to a generalization of the well studied Feedback Vertex Set (FVS) problem, where only the conflict-cycles must be cut. In order to solve MBL, we use a (more general) variant of FVS, named Subset-FVS and studied by Even et al. [3], whose definition is the following:

Problem : Subset-FVS
Input : A directed graph $G = (V, E)$, a set $Y \subseteq V \cup E$, a weight map $w : V \rightarrow \mathbb{R}$.
Output : A set $V'' \subseteq V$ of minimum weight such that, with $V' = V - V''$, no cycle in $G[V']$ uses a vertex or an arc from Y.

In our paper, we are only interested in the restriction of Subset-FVS on adjacency-order graphs, where Y is the set of top-down arcs and w is such that only vertices in W can be deleted:

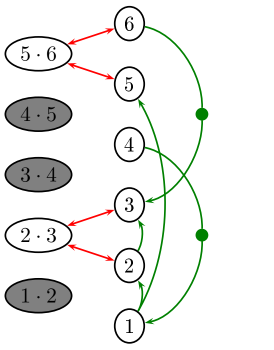
Algorithm 1. $O(\log^2(|X|))$**- and** $O(\log(k)\log\log(k))$**-approximation for MBL**

```
Input: A directed acyclic graph $\Pi = (\Sigma, D)$
1. Create $G_\Pi = (V, E)$ the adjacency-order graph of $\Pi$;
2. $W'' \leftarrow$ AOG-Subset-FVS$(G_\Pi, X[G_\Pi], w[G_\Pi])$;
3. $W' \leftarrow W - W''$;
4. $(V_1, V_2, \dots, V_h) \leftarrow$ SCC-sort$(G_\Pi[W' \cup \Sigma])$;
5. For $i \leftarrow 1$ to $h$;
6.     $\mu_i \leftarrow$ sort$(V_i \cap \Sigma)$;
7. $\pi \leftarrow \mu_1 \cdot \mu_2 \cdot \dots \cdot \mu_h$;
8. return $\pi$;
```

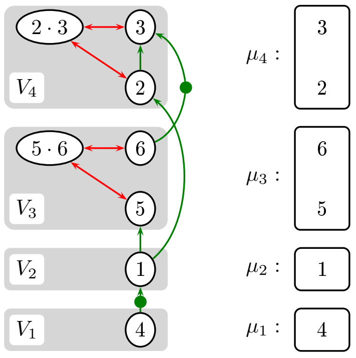
Problem : AOG-Subset-FVS
Input : An adjacency-order graph G_Π, $Y = X[G_\Pi]$, $w = w[G_\Pi]$
Output : A set W'' solution of Subset-FVS(G_Π, Y, w)

We note that any algorithm for Subset-FVS is also valid for AOG-Subset-FVS. Two approximation algorithms are given in [3] for Subset-FVS. The first one achieves an approximation ratio of $O(\log^2 |Y|)$, while the second algorithm achieves a ratio of $O(\min(\log(\tau^*)\log\log(\tau^*), \log(n)\log\log(n)))$, where τ^* is the value of the optimal *fractional* solution for the corresponding linear programming problem (thus τ^* is upper bounded by the optimal solution of Subset-FVS).

We use those approximation algorithms to solve MBL (see Algorithm 1, and Figures 3a and 3b for an example). We denote by SCC-sort() an algorithm that decomposes a graph into its strong connected components, and then topological sorts these components. Also, let sort() be an algorithm that sorts a set of integers



(a) Result of AOG-Subset-FVS: $W'' = \{1\cdot 2, 3\cdot 4, 4\cdot 5\}$

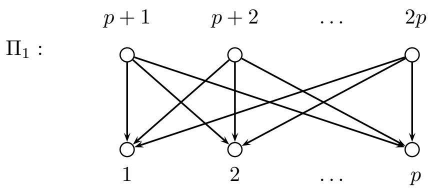


(b) SCC-sort gives four components V_1, V_2, V_3 and V_4, and Algorithm 1 returns $\pi = 4\cdot 1\cdot 5\cdot 6\cdot 2\cdot 3$

Fig. 3. Key steps of Algorithm 1 on the example given in Fig. 2a

according to the increasing order of its elements. Algorithm 1 is derived from the constructive proof of Theorem 4, and its correctness follows from Theorem 4 itself.

Depending on the algorithm used for AOG-Subset-FVS, Algorithm 1 can be either an exponential-time exact algorithm, an $O(\log^2 |X|)$-approximation or an $O(\log(k)\log\log(k))$-approximation (where $|X|$ is the number of arcs $u \to v$ in $\Pi = (\Sigma, D)$ with $u > v$, and k the optimal value of our problem). Note that the two latter ratios are incomparable, since we may have $|X| \approx nk$ or $k \approx n|X|$, as can be seen in Fig. 4a and Fig. 4b.



(a) With Π_1: $|X| = p^2 \;>>\; k = 1$

$\Pi_2 :\ 1 \to 3 \to \ldots \to (2p-1) \to 2 \to 4 \to \ldots \to (2p)$

(b) With Π_2: $|X| = 1 \;<<\; k = 2p - 1$

Fig. 4. Comparing $|X|$ (number of arcs $u \to v$ in Π such that $v < u$) to k (optimal breakpoint distance to the identity)

5.2 An $(m^2 + 4m - 4)$-Approximation Algorithm

In this section, we assume that the partial order Π is generated from m gene maps. Recall that a gene map is a totally ordered sequence of blocks, each of which is an unordered set of markers. We exploit this supplementary information to obtain an $(m^2 + 4m - 4)$-approximation algorithm for AOG-Subset-FVS, and therefore a new approximation algorithm, having the same ratio, for MBL. Before giving the algorithm, we first introduce a few definitions: a path $u \overset{R}{\to}{}^*_D v$ in (Σ, D) is said to be a *shortcut* of a conflict-cycle $\mathcal{C}$ (see Fig. 5), if:

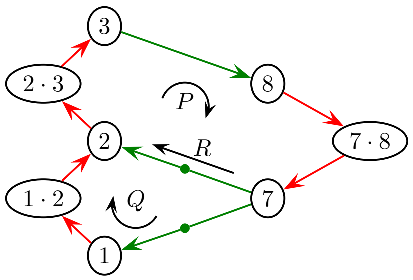


Fig. 5. Cycle $\mathcal{C} = 2 \overset{P}{\to}{}^* 7 \overset{Q}{\to}{}^* 2$ is a conflict-cycle (it contains $7 \to 1 \in X$). The length-1 path R forms a shortcut for $\mathcal{C}$ (with $\mathcal{C}' = 2 \overset{P}{\to}{}^* 7 \overset{R}{\to}{}^* 2$, $\mathcal{C}'$ is a conflict-cycle, and $W(Q) = \{1 \cdot 2\}$). So $\mathcal{C}'$ is the only minimal conflict-cycle.

Algorithm 2. $(m^2 + 4m - 4)$-approximation for AOG-Subset-FVS

```
Input: An adjacency-order graph $G_\Pi = (V, E), X[G_\Pi], w[G_\Pi]$
1. $W'' \leftarrow \emptyset$;
2. while there exists a minimal conflict-cycle $\mathcal{C}$ in $G_\Pi[V - W'']$;
3.     $L \leftarrow$ the set of low joints of $\mathcal{C}$;
4.     $W'' \leftarrow W'' \cup \{e^F \ : \ e \in L\}$;
5. return $W''$;
```

- $u, v \in \Sigma(\mathcal{C})$ (we write P and Q the paths such that $\mathcal{C} = u \xrightarrow{P}{}^{+} v \xrightarrow{Q}{}^{+} u$),
- cycle $\mathcal{C}' = u \xrightarrow{P}{}^{+} v \xrightarrow{R}{}^{*}_{D} u$ is a conflict-cycle,
- $W(Q) \neq \emptyset$ (using the shortcut removes at least one adjacency).

We say that a conflict-cycle is *minimal* if it has no shortcut. With the following property, we ensure that removing minimal conflict-cycles is enough to remove all conflict-cycles.

Property 7. *If an adjacency-order graph contains a conflict-cycle, it also contains a minimal conflict-cycle.*

Proof. Take a non-minimal conflict-cycle $\mathcal{C}$. If $\mathcal{C}$ is simple, we use the shortcut to create a conflict-cycle $\mathcal{C}$' with $|W(\mathcal{C}')| < |W(\mathcal{C})|$. Otherwise, if $\mathcal{C}$ is not simple, there exists a vertex u such that $\mathcal{C} = u \xrightarrow{P}{}^{+} u \xrightarrow{Q}{}^{+} u$, and one of P and Q, say P, uses at least one arc of X. Thus P is a conflict-cycle, with $|W(P)| \leq |W(\mathcal{C})|$, and $|\ell(P)| < |\ell(\mathcal{C})|$. In both cases, we create a conflict-cycle $\mathcal{C}'$ with either $|W(\mathcal{C}')| < |W(\mathcal{C})|$, or $|W(\mathcal{C}')| = |W(\mathcal{C})|$ and $|\ell(P)| < |\ell(\mathcal{C})|$. If $\mathcal{C}'$ is not a minimal conflict-cycle, we can replace $\mathcal{C}$ by $\mathcal{C}'$ and iterate this process: it necessarily ends ($\ell(\mathcal{C})$ and $|W(\mathcal{C})|$ must remain positive integers), and reaches a minimal conflict-cycle. □

In a cycle $\mathcal{C}$, we say that a vertex $e \in \Sigma(\mathcal{C})$ is a *low joint*, if in this cycle, e is adjacent to both an arc of D and an arc of F linking it to $e \cdot (e+1)$. Formally, there exist vertices $e^D \in \Sigma(\mathcal{C})$ and $e^F = e \cdot (e+1) \in W(\mathcal{C})$ such that one of the paths $e^D \rightarrow_D e \rightarrow_F e^F$ or $e^F \rightarrow_F e \rightarrow_D e^D$ appears in $\mathcal{C}$.

Algorithm 2 is an (m^2+4m-4)-approximation for AOG-Subset-FVS. Used as a subroutine in Algorithm 1, it gives us an (m^2+4m-4)-approximation for the MBL problem (see Corollary 10). Due to space constraints, the approximation ratio analysis for this Algorithm 2 is only briefly summarized: the first step is to bound the number of low joints that can appear in some minimal conflict-cycle by m (although two markers corresponding to different low joints can appear in the same gene map). Then, we show that for any adjacency $w \in W$, at most $2m$ minimal conflict-cycles using w can appear during step 2. of Algorithm 2. Hence we can bound the number of vertices deleted by Algorithm 2 for a given vertex $w \in W$ by $2m^2$. A tighter analysis gives us the following result.

Lemma 8. *Let $w \in W$ and let $\mathbb{C}$ be the set of all cycles considered during step 2. of Algorithm 2 going via w. Then the cardinality of the set of low joints in cycles of $\mathbb{C}$ is upper bounded by $m^2 + 4m - 4$.*

Theorem 9. *Algorithm [2] is an* (m^2+4m-4)*-approximation of* AOG-Subset-FVS*, where* m *is the number of gene maps used to create the input graph.*

Proof. Correctness of Algorithm [2] follows from Corollary [6], since Algorithm [2] removes at least one vertex from each conflict-cycle. Let $W^o = \{w_1^o, \ldots, w_k^o\}$ be an optimal solution of size k. For each w_i^o, Algorithm [2] deletes at most m^2+4m-4 adjacencies of W (by Lemma [8]). Since every cycle considered by the algorithm goes through some w_i^o, the total size of the output solution is at most $k(m^2+4m-4)$. □

Corollary 10. *Using Algorithm [2] as an approximation for* AOG-Subset-FVS *in Algorithm [1] yields an* (m^2+4m-4)*-approximation for the* MBL *problem.*

6 Conclusion

In this paper, we revisited the MBL problem with the aim of providing correct algorithms in replacement of those proposed in [4,2], which were based on an inaccurate statement. We proposed a new graph G_Π to represent the conflicts between the given partial order Π and the reference genome Id_n, we characterized the cycles containing these conflicts, we showed how the MBL problem reduces to solving the AOG-Subset-FVS problem in G_Π, and we proposed three approximation algorithms. These algorithms allow us to approach a given, practical instance of MBL from different viewpoints, by choosing the appropriate algorithm depending on the data at hand (i.e., whether the instance is created from few gene maps) and on the parameter evaluation (k and $|X|$). We also pointed out that MBL is **APX**-hard ; following this line, it would be interesting to know whether there exists a constant-ratio approximation algorithm for MBL (which would classify MBL as **APX**-complete). Another challenging question is whether MBL is Fixed-Parameter Tractable, notably when the parameter is the number m of gene maps that were used to construct the partial order Π.

References

1. Blin, G., Blais, E., Hermelin, D., Guillon, P., Blanchette, M., El-Mabrouk, N.: Gene maps linearization using genomic rearrangement distances. Journal of Computational Biology 14(4), 394–407 (2007)
2. Chen, X., Cui, Y.: An approximation algorithm for the minimum breakpoint linearization problem. IEEE/ACM Trans. Comput. Biology Bioinform. 6(3), 401–409 (2009)
3. Even, G., Naor, J., Schieber, B., Sudan, M.: Approximating minimum feedback sets and multi-cuts in directed graphs. In: Balas, E., Clausen, J. (eds.) IPCO 1995. LNCS, vol. 920, pp. 14–28. Springer, Heidelberg (1995)
4. Fu, Z., Jiang, T.: Computing the breakpoint distance between partially ordered genomes. J. Bioinformatics and Computational Biology 5(5), 1087–1101 (2007)

5. Yap, I.V., Schneider, D., Kleinberg, J., Matthews, D., Cartinhourb, S., McCouch, S.R.: A graph-theoretic approach to comparing and integrating genetic, physical and sequence-based maps. Genetics 165(4), 2235–2247 (2003)
6. Zheng, C., Lenert, A., Sankoff, D.: Reversal distance for partially ordered genomes. In: ISMB (Supplement of Bioinformatics), pp. 502–508 (2005)
7. Zheng, C., Sankoff, D.: Genome rearrangements with partially ordered chromosomes. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 52–62. Springer, Heidelberg (2005)

An $\mathcal{O}(n^2)$-time Algorithm for the Minimal Interval Completion Problem

Christophe Crespelle[1] and Ioan Todinca[2]

[1] LIP6, Université Paris 6
christophe.crespelle@lip6.fr
[2] LIFO, Université d'Orleans, BP 6759, F-45067 Orleans Cedex 2, France
ioan.todinca@univ-orleans.fr

Abstract. The minimal interval completion problem consists in adding edges to an arbitrary graph so that the resulting graph is an interval graph; the objective is to add an inclusion minimal set of edges, which means that no proper subset of the added edges can result in an interval graph when added to the original graph. We give an $\mathcal{O}(n^2)$-time algorithm to obtain a minimal interval completion of an arbitrary graph. This improves the previous $O(nm)$ time bound for the problem and lower this bound for the first time below the best known bound for minimal chordal completion.

1 Introduction

The *interval completion* of a graph $G = (V, E)$ consists in adding a set of edges F to G so that the resulting graph $H = (V, E \cup F)$ is an *interval graph*, that is, the intersection graph of some intervals of the real line. The problem of computing such a completion that realizes the minimum number of *fill edges* $|F|$ is known as the *Minimum Interval Completion* problem. If the set F is only required to be minimal for inclusion among all sets resulting in an interval completion, the problem is referred to as the *Minimal Interval Completion* problem. Applications of Minimum Interval Completion arise in various contexts such as computational biology , archeology , and clone fingerprinting . In addition, interval completion has been studied for its connection with another fundamental problem of computer science known as *chordal completion* (see [6] for a survey). A *chordal graph* is a graph that contains no chordless induced cycle on four or more vertices. The class of chordal graphs properly includes all interval graphs. The *Minimum Chordal Completion* problem (also known as *minimum fill-in* or *minimum triangulation*) received much attention, in particular because it plays a key role in *sparse matrix multiplication* [16]. Another chordal completion problem involves minimizing the maximum clique size of the completed graph. Indeed, this parameter is nothing else but *treewidth*, which is extensively studied, notably because many NP-complete problems become polynomial on graphs having this parameter bounded. Interval completion is similarly related to another famous graph parameters called *pathwidth*.

© Springer-Verlag Berlin Heidelberg 2010

Both for interval completion and chordal completion, it turns out that minimizing the number of fill edges or minimizing the maximum clique size of the completed graph are NP-complete problems (see [1] for references). But both kinds of solution are obtained on inclusion-minimal completions. Considering that computing a minimal completion is polynomial (in both cases), this significantly increases the importance of the problem and motivated many works. Minimal completion algorithms are often used as heuristics for minimum completion and for computation of pathwidth or treewidth.

Related works. The first algorithm solving the Minimal Chordal Completion problem in polynomial time is due to Rose, Tarjan and Lueker [15], with an $\mathcal{O}(nm)$ time complexity. As usual, n denotes the number of vertices and m denotes the number of edges of the input. Several authors gave different approaches with the same running time, but it took almost 30 years to improve the $\mathcal{O}(n^3)$ worst-case complexity. Using the algorithm of Heggernes, Telle and Villanger [10], one can compute a minimal chordal completion in $\mathcal{O}(n^\alpha \log n)$ time, where $\mathcal{O}(n^\alpha)$ is the time required for the multiplication of two $n \times n$ matrices.

The first polynomial-time algorithm for the Minimal Interval Completion problem was given by Ohtsuki et al. [13], running in $\mathcal{O}(nm')$ time; here m' denotes the number of edges of the resulting completed graph. A similar approach has been rediscovered in [9]. Using a completely different technique, Suchan and Todinca gave an $\mathcal{O}(nm)$-time algorithm in [17]. Note that several recent articles consider different types of minimal completion problems, e.g. into split graphs [7], comparability graphs [8] or proper interval graphs [14].

Our results. The aim of this paper is to show the following theorem.

Theorem 1. *There is an $\mathcal{O}(n^2)$-time algorithm computing a minimal interval completion of an arbitrary graph.*

Note that our approach is faster than the previous $\mathcal{O}(nm')$ algorithm of [13] and the $\mathcal{O}(nm)$ algorithm of [17]. It is also faster than the best algorithms for the Minimal Chordal Completion problem, and this is the first time that the time bound for interval completion goes below the best known bound for chordal completion. Moreover, unlike the algorithm of [10] which uses matrix multiplication techniques, ours is purely graph-theoretic. Like in [13], our algorithm is incremental in the sense that we add the vertices of G one by one, and each time a new vertex v_{i+1} arrives, the new minimal interval completion is computed from the one obtained at step i by only adding edges incident to v_{i+1}. The second common feature is that we use PQ-trees, which capture all interval representations of the interval completion computed so far. But our procedure for choosing the set F of fill edges is completely different from [13] and simpler; and we rely on the results of [3] for efficiently updating, at each step, the PQ-tree of the new completion.

After giving, in next sections, some basic definitions and preliminary results, we present in Section 4 our main combinatorial tool for the minimal interval completion, while the algorithmic details and data-structures are described in Section 5.

2 Preliminaries

We consider simple and connected input graphs. A graph is denoted by $G = (V, E)$, with $n = |V|$, and $m = |E|$. For a set $U \subseteq V$, $G[U]$ denotes the subgraph of G induced by the vertices in U. Set U of vertices is called a *clique* if $G[U]$ is complete. For a vertex $v \in V$ or a subset $U \subseteq V$, we will informally use $G - v$ and $G - U$ to denote the graphs $G[V \setminus \{v\}]$ and $G[V \setminus U]$, respectively. We also consider the operation of inserting a new vertex x together with the edges defining its neighborhood in a graph G and we denote the resulting graph by $G + x$. A path is a sequence $[v_1, v_2, \ldots, v_p]$ of pairwise distinct vertices such that v_i is adjacent to v_{i+1}, for all $1 \leq i < p - 1$. A cycle is a path such that the first and last vertices are adjacent. The *neighborhood* of a vertex v in G is $N_G(v) = \{u \mid uv \in E\}$. Similarly, for a set $U \subseteq V$, $N_G(U) = \bigcup_{v \in U} N_G(v) \setminus U$. When graph G is clear from the context, we will omit subscript G; in particular, the neighborhood of vertex x in $G + x$ will often be denoted simply $N(x)$.

A graph H is an *interval* graph if continuous intervals of the real line can be assigned to each vertex of H such that two vertices are neighbors if and only if their intervals intersect. A graph $H = (V, E \cup F)$ is called an *interval completion* of an arbitrary graph $G = (V, E)$ if H is an interval graph. If no proper subgraph of H is an interval completion of G, we say that H is a *minimal interval completion* of G. In a broader sense, we say that a graph H is *inclusion-minimal* among a set of graphs referring to the inclusion relationship on edge sets of graphs. An edge that is added to the input graph G is called a *fill edge*, and the process of adding edges between a fixed vertex x and a set U of vertices is called *filling* U.

Theorem 2 ([4]). *A graph G is interval if and only if there is a path CP_G whose vertex set is the set of all maximal cliques of G, such that the subgraph of CP_G induced by the maximal cliques of G containing vertex v forms a connected subpath, for each vertex v of G. Such a path will be called a* clique path *of G.*

Let the maximal cliques of an interval graph G be labeled $1, 2, ..., k$, according to the order in which they appear in a clique path of G. Then, as a consequence of Theorem 2, an interval representation of G can be obtained by associating with each vertex v the closed interval that consists of the labels of the maximal cliques containing v. In this way, every clique path of G defines an interval representation of G.

A vertex set $S \subseteq V$ is a *minimal separator* of G if there exist two vertices u and v such that S separates them (i.e. u and v are in different connected components of $G - S$) and S is inclusion-minimal among the sets of vertices separating u and v. The following lemma shows that minimal separators can be easily found on any clique path of the graph, and so on its PQ-tree.

Lemma 1 (see e.g. [5]). *Let G be an interval graph and let CP_G be any clique path of G. A set of vertices S is a minimal separator of G if and only if S is the intersection of two maximal cliques of G that are neighbors in CP_G. In particular, all minimal separators of G are cliques.*

It is shown in [2] that all clique paths of an interval graph G can be represented by a structure called PQ-tree. The PQ-tree of G, denoted T in the rest of the paper, is a rooted tree whose leaves are the maximal cliques of G. Its internal nodes are labeled P (*degenerate nodes*) or Q (*prime nodes*). Any Q-node q is assigned two linear orderings, denoted σ_q and $\bar{\sigma}_q$, on the set of its children, $\bar{\sigma}_q$ being the reverse order of σ_q. A *solidification* of a PQ-tree T, is an assignation, to each node u of T, of a *valid* linear ordering on its children, that is: any linear ordering if u is a P-node, σ_u or $\bar{\sigma}_u$ if u is a Q-node. Choosing a solidification of the PQ-tree, we obtain an order on the leaves by reading them from left to right in a plane drawing of the solidified rooted tree. The main property of the PQ-tree of G is that the set of orderings obtained this way is precisely the set of clique paths of G (see [2]).

In this document, the subtree of T rooted at node u will be denoted by T_u. The set of children of u will be denoted by $\mathcal{C}(u)$ and its parent by $parent(u)$.

3 The Vertex Incremental Approach

Let us observe that a minimal interval completion can be obtained incrementally. This result is due to [13].

Lemma 2 ([13]). *Let H be a minimal interval completion of an arbitrary graph G. Let G' be a graph obtained from G by adding a new vertex x, with neighborhood $N_{G'}(x)$. There is a minimal interval completion H' of G' such that $H' - x = H$.*

Hence, in computing a minimal interval completion of G, we introduce the vertices of G one by one in the order $x_1, x_2, \ldots, x_n$. Given a minimal interval completion H_i of $G_i = G[\{x_1, \ldots, x_i\}]$, we compute an interval completion H_{i+1} of G_{i+1} by adding to H_i the vertex x_{i+1} together with the edges between x_{i+1} and $N_{G_{i+1}}(x_{i+1})$, plus a well chosen set of additional edges incident to x_{i+1}. Thus, for proving Theorem 1, it is sufficient to solve the following problem in $\mathcal{O}(n)$ time.

The new problem. From now, we consider as input an *interval* graph $G = (V, E)$ on n vertices and a new vertex x to be inserted in G, together with a set of edges incident to x. We want to compute a minimal interval completion H of $G + x$, obtained by adding edges incident to x only. Moreover, the PQ-tree of graph G will be part of the input, and we will also compute the PQ-tree of H.

For the rest of this document, let G' denote the graph $G + x$. Consider any clique path CP_H of the obtained completion H of G'. By property of clique paths, the cliques containing x induce a subpath P_x of CP_H. Now, let us get back to G. Delete x from every bag (clique) in CP_H, and possibly remove the bags that do not correspond to maximal cliques of G. This yields a clique path CP_G of G, which is said to be obtained by *pruning* vertex x from CP_H. Clearly the maximal cliques that come from P_x still induce a subpath of CP_G. Our aim is to do the converse: to find a clique path CP_G of G and a subpath of CP_G in which, by adding vertex x to every bag and possibly transforming the bordering separators into new bags of H (with x contained), we obtain a minimal interval completion of G'.

Definition 1. *A clique path CP_G is called* nice *if there exists a minimal interval completion H such that CP_G is obtained by pruning x from some clique path of H. In this case, we say that H* respects *CP_G.*

For obtaining a nice clique path we have to distinguish between two cases. For lack of space, we do not detail the case where the neighborhood of x in G' is a clique. We concentrate on the more general case where the neighborhood of x is not a clique.

4 When the Neighborhood of x Is Not a Clique

This is the main and most difficult case of our algorithm. We show later how to handle it using the PQ-tree. But for now, let us first note that, as stated in [9], any clique path of G is associated with a canonical interval completion respecting it (Lemma 3 below). We need the following definition.

Definition 2. *Let CP_G be a clique path of G ordered left-to-right. If $N(x)$ is not a clique, we denote by K_L (resp. K_R) the leftmost (resp. rightmost) clique of CP_G such that x has a neighbor in $K_L \setminus K_{L+1}$ (resp. $K_R \setminus K_{R-1}$), in graph G'.*

Lemma 3 ([9]). *For any clique path CP_G of G, there is a unique interval completion H respecting CP_G which is inclusion-minimal. Moreover the neighborhood of x in H is formed exactly by $N(x)$ augmented with the vertices of the cliques strictly between K_L and K_R in CP_G, if there are any, or augmented with the vertices of the minimal separator $K_L \cap K_R$ otherwise.*

Note that if CP_G is a nice clique path, then H is necessarily a minimal interval completion of G', but it may not be otherwise. Also note that any clique path obtained from a nice one CP_G by rearranging the maximal cliques of G in an order such that the cliques of the interval $[\![K_L, K_R]\!]$ of CP_G still form an interval whose endpoints are K_L and K_R (not necessarily in this order) is a nice clique path.

Before proving our main theorem (Theorem 3) which gives a way to obtain a nice clique path using the PQ-tree, we need to introduce some definitions and notations. For any node u of the PQ-tree of G, we denote by $B[u]$ the set of vertices of G contained in the cliques of the subtree rooted in u. We call this set a *block*. The *border* of $B[u]$ is the set of vertices of the block having neighbors outside the block. The *interior* of $B[u]$ is formed by the vertices of the block that are not in the border. We say that $B[u]$ is *hit* if, in the graph G', x has a neighbor in the interior of the block. Otherwise, the block is called *clean*. If all vertices in the interior of the block are neighbors of x in G' then the block is called *full*. By extension, we also say that a node of the PQ-tree is hit, full or clean according to the state of its corresponding block. We point out that for all internal nodes u, the block $B[u]$ has a non-empty interior.

In the sequel, we denote by r the lowest node of the PQ-tree such that $N(x) \subseteq B[r]$. Since $N(x)$ is not a clique, node r is uniquely defined and is such that the interior of $B[r]$ contains at least two non-adjacent vertices both linked to x.

The children $L_\sigma(u)$ and $R_\sigma(u)$ introduced in the following definition correspond to the cliques K_L and K_R of Definition 2, the difference being that here we consider blocks instead of cliques.

Definition 3. *Consider a node u of the PQ-tree and a valid order σ of its children. We denote by $L_\sigma(u)$:*

- *either the leftmost child v of u such that the corresponding block $B[v]$ contains a neighbor of x in G', and this neighbor is not in $B[v']$, where v' is the right-hand brother of v in σ,*
- *or the last element of σ if u has no such child v.*

$R_\sigma(u)$ is defined symmetrically.

By definition of node r, and since $N(x)$ is not a clique, for all valid orders σ of the children of r, $L_\sigma(r) <_\sigma R_\sigma(r)$. Thanks to the definition above, we can identify two branches of a solidified PQ-tree delimiting the part of the tree containing nodes whose blocks are filled in the canonical completion associated to the considered solidification.

Definition 4. *Let G be an interval graph and x be a vertex to be inserted in G. Let π be a solidification of T. Denote by $\pi(u)$ the ordering defined by this solidification on the children of node u. The* left branch $LB(\pi)$ *of π is the set of nodes defined recursively as follows :*

- *$L_{\pi(r)}(r)$ is in the left branch.*
- *For any u in the left branch, $L_{\pi(u)}(u)$ is also in the left branch.*

The right branch $RB(\pi)$ of π is defined symmetrically.

The left and the right branch of π isolate a subpart of the solidified PQ-tree. Let CP_G be the clique path corresponding to this solidification π. Let H be the unique interval completion respecting CP_G that is inclusion-minimal (see Lemma 3). Observe that, by the definition of the left and right branch, the bottom of these branches correspond to K_L and K_R respectively (see Definition 2). By Lemma 3, all maximal cliques strictly in between K_L and K_R become filled in H. All cliques strictly outside this interval remain clean. An important consequence is that, for any node of the PQ-tree not belonging to one of the two branches and different from r, we can change the permutation of its children and the new solidification will yield the same interval completion.

We define a class of nodes u, that we call *forced*, which have the property that their corresponding block becomes filled in any interval completion of G (Lemma 4 below).

Definition 5. *A* forced *node is defined inductively by: a node u of the PQ-tree is forced if and only if:*

- *u is full, or*
- *u is a degenerate node and every child v of u is forced.*
- *u is a prime node and the first and the last child of σ_u are forced.*

Lemma 4 (forced blocks). *Let u be an* internal *forced node of the PQ-tree of G. The block $B[u]$ is filled in every interval completion of G'.*

Consider a minimal interval completion of G'. It respects some clique path of G, obtained from a solidification π of the PQ-tree T. The proof can be made by induction from the leaves to the root of T. Basically, it relies on the fact that a forced node u has, by definition, a full descendant and therefore is hit. Moreover, its first and last children in π are forced, and so they are hit too. It implies that all the other children of u must be filled. The fact that the first (or last) child v of u is also filled can be obtained either from the induction hypothesis, if v is an internal node, or from the base case of the induction if v is a leaf. Finally, u which has all its children filled is filled itself.

We now define a set of *nice* orderings on the children of a node u, such that, using these orderings, the corresponding solidification yields a nice clique path. The idea is to group hit nodes together as much as possible and to place non-forced nodes on the left and right branches, if possible, so that we don't need to fill sets of nodes that could avoid to be filled. As it is defined below, a nice ordering suits for nodes in $\{r\} \cup LB(\pi)$: nodes in $RB(\pi)$ will be in fact assigned the reverse order of a nice ordering.

Definition 6. *For each node u in the subtree rooted at r we define the set Π_u^{nice} of nice orders of the children of u as follows:*

1. *if u is degenerate, then Π_u^{nice} is the set of orders σ such that the hit children of u form an interval $I = [\![L_\sigma(u), R_\sigma(u)]\!]$ such that $R_\sigma(u)$ is the last element of σ and such that $L_\sigma(u)$ is forced only if all the elements of I are forced, and $R_\sigma(r)$ is forced only if the elements strictly between $L_\sigma(r)$ and $R_\sigma(r)$ are forced.*
2. *if u is prime, then Π_u^{nice} is the set of valid orders $\sigma \in \{\sigma_u, \bar{\sigma}_u\}$ such that the first element of σ is forced only if the last one is forced too, and the first element v of σ is such that $(N(x) \cap B[u]) \setminus B[v] \neq \varnothing$.*

Every node of the subtree rooted at r admits at least one nice ordering of its children. Recall that for a solidification π, and for a node u in T, we denote by $\pi(u)$ the order defined by π on the children of u.

Definition 7. *A* nice solidification *π is a solidification such that $\pi(r)$ is a nice order, for every node $u \in LB(\pi)$, $\pi(u)$ is a nice order, and for every node $v \in RB(\pi)$, $\bar{\pi}(v)$ is a nice order.*

The following theorem is our main combinatorial tool toward computing a nice clique path.

Theorem 3. *A clique path corresponding to a nice solidification of the PQ-tree is a nice clique path.*

Idea of the proof. Fix a nice solidification π of the PQ-tree and let CP_G be the corresponding clique path. Denote by H the interval completion respecting

CP_G that is minimal for this property (recall that it is unique, by Lemma 3). Assume by contradiction that there exists a minimal interval completion H' strictly contained in H. H' respects the clique path of some solidification π' of the PQ-tree of G. Choose this solidification π' as similar as possible to π, in the following sense: (1) the minimum depth d of a node u such that $\pi(u) \neq \pi'(u)$ is as large as possible (by depth we mean the distance from u to the root of the PQ-tree) and (2) subject to the first condition, the number of nodes of depth d such that the two solidifications differ on these nodes is as small as possible. Let u be a node of minimum depth, with $\pi(u) \neq \pi'(u)$. We prove that, in the solidification π', we can replace the solidification of the subtree rooted in u such that $\pi'(u)$ becomes $\pi(u)$, and the clique path defined by this new solidification gives rise to the same interval completion H'. This will give us a contradiction completing our proof. From the remarks following Definition 4, our node u is necessarily in the set $\{r\} \cup LB(\pi) \cup RB(\pi)$. Moreover, observe that, by definition, the left and the right branch of the two solidifications π and π' are the same from the root of the PQ-tree down to level d. Thus, u is also in $\{r\} \cup LB(\pi') \cup RB(\pi')$.

For lack of space we cannot consider all cases: instead, we give as an example the case where $u \neq r$ is in the left-branch of the two solidifications and is a prime node. Then $\pi'(u) = \bar{\pi}(u)$. Let v be the leftmost child of u in $\pi(u)$. We show that $B[v]$ is filled in H. By the definition of a nice ordering on the children of u, there is another child v' of u such that $N(x) \cap B[v']$ is not contained in $N(x) \cap B[v]$. Consequently, since u is on the left branch of π' and, in $\pi'(u)$, the child v' of u is to the left of v (recall that $\pi'(u) = \bar{\pi}(u)$), the block $B[v]$ is filled in H', and so is filled in H too.

This implies that $v = L_{\pi(u)}(u)$, otherwise v would be clean, and then not filled. It is possible to show that any non-forced node in $\{r\} \cup LB(\pi) \cup RB(\pi)$ is not filled in H. Then, v, which is on the left branch of π and is filled, is necessarily forced. It follows, by construction of nice orderings on prime nodes, that the rightmost child of u in $\pi(u)$ is also forced, and so is u. By Lemma 4, $B[u]$ is filled in H'. Then, in π', we can reverse $\pi'(u)$ without changing the corresponding interval completion, which is a contradiction with our choice of π'. □

5 The Algorithm

This section describes our $O(n)$ time incremental algorithm for computing a minimal interval completion of $G + x$.

5.1 Data-Structure: *PQ*-Representation

The set of leaves of the *PQ*-tree is the set of maximal cliques of the graph. Since an interval graph has at most $n - 1$ maximal cliques, it follows that the number of leaves of the *PQ*-tree is $O(n)$. And since every internal node has at least two children, the total number of nodes of a *PQ*-tree is $O(n)$. However, to get a complete representation of the graph, cliques have to be encoded. Classically, this is done by assigning to each leaf of the *PQ*-tree the list of nodes involved in the corresponding maximal clique of the graph. This makes the overall size of the structure inflate to $\Omega(n+m)$.

Here, we use a variant of the PQ-tree, called PQ-representation [3]: instead of being stored in the leaves, the vertices of G are stored in the internal nodes of the PQ-tree (thanks to the pointers defined below). This results in a complete representation of the graph which takes only $O(n)$ space and has deeper structural properties. The PQ-representation is essentially the same structure as the MPQ-tree introduced in [11]. However, we formalize it in a different way that fits better our purposes.

Recall that T is the PQ-tree of G. We denote e_x for the least common ancestor of the leaves of T corresponding to a maximal clique of G containing x.

Lemma 5 ([12]). *For any vertex x of an interval graph G, at least one of the two following conditions holds:*

1. *the maximal cliques of G containing x are exactly those corresponding to the leaves of T_{e_x}, or*
2. *e_x is a prime node and there exist two distinct children e_x^1, e_x^2 of e_x such that the maximal cliques of G containing x are exactly those corresponding to the union of the sets of leaves of T_u for any child u of e_x between e_x^1 and e_x^2 in σ_{e_x}.*

The PQ-representation of an interval graph G, denoted $PQ(G)$, is made of T and the set of vertices of G, where each vertex x stores a *primary pointer* toward e_x, and two *secondary pointers* toward resp. e_x^1 and e_x^2 when x does not satisfy Condition 1 of Lemma 5 (but Condition 2). These pointers encode which maximal cliques of G (i.e. leaves of T) contain x.

Notation 1. *For each node u of T, we define the following sets:*
$X_u = \{y \in V \mid e_y = u$ *and y has no secondary pointers*$\}$
$Y_u = \{y \in V \mid e_y = u$ *and y has secondary pointers toward the children of u*$\}$

Note that, by definition, if u is degenerate then $Y_u = \varnothing$.

In addition to the pointers from the vertices of G toward the nodes of T, in order to achieve the desired complexity, we also store for each node $u \in T$ the list of vertices in X_u, the list of vertices in Y_u, and, if $parent(u)$ is prime, the list of vertices $y \in Y_{parent(u)}$ such that $e_y^1 = u$ and the list of vertices $z \in Y_{parent(u)}$ such that $e_z^2 = u$.

Since the number of nodes in T is $O(n)$ and since each vertex of G stores at most three pointers and is stored in at most four lists associated to some nodes of T, it follows that the total size of the PQ-representation is $O(n)$.

5.2 Computing a Nice Solidification of the *PQ*-Tree

We first collect information about the nodes of T. For each node, we determine whether it is hit or clean by a bottom-up marking process of the tree in which each node forwards its type to its parent, which is then able to determine its own type. In the same way, we can determine whether the nodes are forced or not. Both routines run in $O(n)$ time.

Before computing a nice solidification, we need to check whether $N(x)$ is a clique, and in the negative, we need to identify node r. These goals are achieved by the following routine. Start with the root as current node. At each step, if the current node u has a unique child v such that $N(x) \subseteq B[v]$, then make it become the new current node, otherwise stop the process. At this point, it is easy to test whether $N(x)$ is a clique, and in the negative (which is the only case we treat in the following), the node on which the routine stopped is nothing but node r. This preliminary step takes $O(n)$ overall time. For sake of clearness, we describe the computation of a nice solidification in two steps, but they can be merged into a single top-down search from r to the leaves of T.

First step: computing nice orderings. Thanks to the information collected initially about the nodes of T, we compute, during an arbitrary traversal of T_r, a nice ordering π_u for every node $u \in T_r$. For sake of simplicity of the presentation, we compute a nice ordering for all the nodes of T_r while it is necessary only for nodes of $\{r\} \cup LB(\pi) \cup RB(\pi)$, where π is the nice solidification we intend to build. We have to distinguish two cases depending on the label of u:

1. u is degenerate; all the clean children of u are placed at the beginning of π_u, and if u has at least one non-forced hit child then we place it right after the clean nodes in π_u, and if u has another non-forced hit child, we place it at the end of π_u.
2. u is prime; if the first child u_f of u in σ_u is forced or is such that $B[u] \cap N(x) \subseteq B[u_f]$, then we set $\pi_u = \bar{\sigma}_u$, otherwise we set $\pi_u = \sigma_u$.

A degenerate node u can be treated in $O(|\mathcal{C}(u)|)$ time – recall that $\mathcal{C}(u)$ denotes the set of children of u. In the treatment of a prime node u, the difficult part is to test whether $B[u] \cap N(x) \subseteq B[u_f]$. To that purpose, we have to check whether all the children of u different from u_f are clean and whether all the vertices of $Y_u \cap N(x)$ are such that $e_y^1 = u_f$. This can be done in $O(|\mathcal{C}(u)| + |Y_u|)$ time. Thus the total running time of the first step is $O(\sum_{u \in T_r} |\mathcal{C}(u)| + |Y_u|) = O(n)$.

Second step: reversing orderings of the right branch. The only thing left to do in order to obtain a nice solidification π is to identify the nodes of the right branch $RB(\pi)$ and to reverse the nice ordering computed for them in the previous step. This is achieved by following the path defined by $RB(\pi)$, from r to the leaf corresponding to K_R, while, at the same time, we modify π along this path. Similarly, we can identify the leaf l_L corresponding to K_L in the clique path defined by the solidification π, with the difference that, doing so, we don't need to change solidification π. This step takes $O(n)$ time.

Finally, the time needed to compute a nice solidification π of the PQ-tree is $O(n)$, and we can identify K_L and K_R in the clique path defined by π within the same complexity.

5.3 Overview of the Algorithm

From Lemma 2, we can compute a minimal interval completion of graph G incrementally. We start from the empty graph, and we add the vertices of G

one by one. At each step, when a new vertex x is added, we compute a minimal interval completion of the augmented graph by adding only edges incident to x.

We proceed by computing a nice solidification π of the PQ-tree thanks to the PQ-representation, as shown in Section 5.2. This takes $O(n)$ time. Moreover, within the same complexity we can get the cliques K_L and K_R in the corresponding clique path CP_G which is, from Theorem 3, a nice clique path. Then, we compute the set F of nodes that has to be filled according to Lemma 3.

We proceed as follows. First, from the PQ-representation solidified by π, we compute the interval model of G based on the clique path corresponding to π, that is, the order σ on the maximal cliques of G corresponding to π and, for each vertex y of G, two pointers from y to the first and the last maximal clique of G containing y, in σ, denoted respectively K_y^1 and K_y^2. This can be done in $O(n)$ time by a simple search of the tree. Thanks to this interval model, we can compute the minimal interval completion H described in Lemma 3: we must fill the set of vertices $F = \{y \in V \mid K_y^1 <_\sigma K_R \text{ and } K_L <_\sigma K_y^2\}$. Set F can be easily computed thanks to a scan of the vertices of G which takes $O(n)$ time.

Thus, the only thing left to do is to update the PQ-representation in order to perform the next incremental step. This is done by inserting x in G along with the edges between x and $N(x) \cup F$. Thanks to the algorithm of [3], we obtain the updated PQ-representation of H in $O(n)$ time. Since an incremental completion step can be performed in $O(n)$ time, including the update cost of the data-structure, the total running time of our completion algorithm is $O(n^2)$.

6 Conclusions and Perspectives

We obtained an $O(n^2)$-time algorithm for the Minimal Interval Completion problem. This complexity is lower than those of the best algorithms for minimal chordal completion: $O(nm)$ in [15] or $o(n^{2.376})$ in [10]. This is somehow natural as interval graphs are "simpler" than chordal graphs. Nevertheless, this sheds a new light on the question of whether it is possible to achieve such a complexity for chordal completion. In particular, one may ask whether it is possible to mimic the approach followed here by using the intersection model of chordal graphs.

Acknowledgment. We would like to thank Karol Suchan and Christophe Paul for useful discussions on the subject.

References

1. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM J. Algebraic Discrete Methods 8(2), 277–284 (1987)
2. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. J. Comput. Syst. Sci. 13(3), 335–379 (1976)
3. Crespelle, C.: Dynamic representations of interval graphs. In: Paul, C., Habib, M. (eds.) WG 2009. LNCS, vol. 5911, pp. 77–87. Springer, Heidelberg (2010), http://www-npa.lip6.fr/~crespell/publications/DynInt.pdf

4. Gilmore, P.C., Hoffman, A.J.: A characterization of comparability graphs and of interval graphs. Canadian J. Math. 16, 539–548 (1964)
5. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs, 2nd edn. Annals of Discrete Mathematics, vol. 57. Elsevier, Amsterdam (2004)
6. Heggernes, P.: Minimal triangulations of graphs: A survey. Discrete Math. 306(3), 297–317 (2006)
7. Heggernes, P., Mancini, F.: Minimal split completions. Discrete Applied Mathematics 157(12), 2659–2669 (2009)
8. Heggernes, P., Mancini, F., Papadopoulos, C.: Minimal comparability completions of arbitrary graphs. Discrete Applied Mathematics 156(5), 705–718 (2008)
9. Heggernes, P., Suchan, K., Todinca, I., Villanger, Y.: Minimal Interval Completions. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 403–414. Springer, Heidelberg (2005)
10. Heggernes, P., Telle, J.A., Villanger, Y.: Computing minimal triangulations in time $o(n^{\alpha logn}) = o(n^{2.376})$. SIAM J. Discrete Math. 19(4), 900–913 (2005)
11. Korte, N., Möhring, R.H.: Transitive orientation of graphs with side constraints. In: Trauner, L. (ed.) 11th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 1985), pp. 143–160 (1986)
12. Korte, N., Möhring, R.H.: An incremental linear-time algorithm for recognizing interval graphs. SIAM J. Comput. 18, 68–81 (1989)
13. Ohtsuki, T., Mori, H., Kashiwabara, T., Fujisawa, T.: On minimal augmentation of a graph to obtain an interval graph. Journal of Computer and System Sciences 22(1), 60–97 (1981)
14. Rapaport, I., Suchan, K., Todinca, I.: Minimal proper interval completions. Information Processing Letters 5, 195–202 (2008)
15. Rose, D., Tarjan, R.E., Lueker, G.: Algorithmic aspects of vertex elimination on graphs. SIAM J. Comput. 5, 146–160 (1976)
16. Rose, D.J.: A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. Graph Theory and Computing, 183–217 (1972)
17. Suchan, K., Todinca, I.: Minimal interval completion through graph exploration. Theoretical Computer Science 410(1), 35–43 (2009)

Centdian Computation for Sensor Networks

Boaz Ben-Moshe[1], Amit Dvir[2], Michael Segal[2], and Arie Tamir[3]

[1] Department of Computer Science, Ariel University Center of Samaria, Israel
[2] Department of Communication Systems Engineering, Ben-Gurion University of the Negev, Israel
[3] School of Mathematical Sciences, Tel Aviv University, Israel

Abstract. This paper focuses on the centdian problem in a cactus network where a *cactus* network is a connected undirected graph, and any two simple cycles in the graph have at most one node in common. The cactus network has important applications for wireless sensor networks when a tree topology might not be applicable and for extensions to the ring architecture. The centdian criterion represents a convex combination of two QoS requirements: transport and delay. To the best of our knowledge, no efficient algorithm has yet been developed for constructing a centdian node in a cactus graph, either sequential or distributed. We first investigate the properties of the centdian node in a cycle graph, and then explore the behavior of the centdian node in a cactus graph. Finally, we present new efficient sequential and distributed algorithms for finding all centdian nodes in a cycle graph and a cactus graph.

Keywords: Distributed Algorithm, Cactus Graph, Centdian Node, Sensor Network.

1 Introduction

A wireless sensor network contains a number of sensor nodes limited in power and memory, distributed across an area using wireless communication links to deliver information between nodes. In some sensor network applications the nodes are barely changed [26]. In recent years, the wireless sensor network has attracted attention [21] since this type of network can be used in a variety of applications, such as health, military, and emergency response. One generic type of application for these networks is monitoring where all the sensors produce relevant information by sensing the area and transmitting the information to a central node called a sink node.In some applications, namely data aggregation, performing in-network fusion of data packets is a useful paradigm. However, it is not applicable in all sensing environments. For military applications, such as receiving an image of a battlefield, the data being transmitted by the nodes provide an important point of view. In such situations, it might not be feasible to aggregate the data from different sensors into a single data packet. In those applications all the information is sent to the sink node. We are assuming that the sink node has the ability to change its position [21] to improve the performance of the network and does not have energy limitations. For example, a group of soldiers

© Springer-Verlag Berlin Heidelberg 2010

(considered as a sink node) collects information from other units in a battlefield. The soldiers may move around, but have to be able to continuously receive data reports [19].

A wireless ring is a structure for applications running on wireless sensor networks that provide quality of service [11]. Wireless Token Ring Protocol (WTRP) is a medium-access-control (MAC) protocol with advantages such as robustness against single node failure, and support for flexible topologies, in which nodes may be partially connected [11]. Lee et al. [18] presented a WTRP for ad hoc networks and intelligent transportation system. Xianpu et al. [25] showed a dynamic token protocol for MANET where all nodes in the network are clustered into several subnets, whose functions are the same as the logic token rings in WTRP. Ergen [11] showed a number of topologies extended to the WTRP protocol: Hierarchical hybrid schemes – combination of star/tree and ring topologies. Token chain – combination of several rings. Data forwarding – clustering stations into multiple rings. Sensor networks – hierarchical clustering by rings, where ring leaders are connected by tree topology. Cactus graphs are motivated by models where a tree topology would be irrelevant in telecommunications. Moreover, the above extensions to the architectures form a cactus graph, which is why practical communication networks may have cactus graph topology. A cactus graph is also a planar graph, enabling us to transmit between nodes without having to consider cross-transmissions, and improving the delivery rate by using the two available paths in each cycle. Wang et al. [23] presented a robust and energy efficient routing scheme using a backup path; therefore, considering all the solutions that use a tree topology as intra topology and merging them with [23], will lead to cactus topology.

1.1 Model and Definitions

We model a network topology by an undirected connected graph $G(V, E, W)$, where V is the node set, E the set of edges between neighboring nodes, and W a function from E to $\mathbb{R}$, which takes on positive values only. For each edge $e(u, v) \in E$, $0 < w_e < \infty$ represents edge weight or length, where a length/weight value represents the amount of energy required to transmit one packet from node v to node u. Note that the edges represent logical connectivity between nodes, i.e., there is an edge between the two nodes u and v if they can hear one another.

For a given pair of nodes u and v, $P(u, v)$ denotes a simple path in G connecting u and v, and its length, $d(P(u, v))$, is defined as the sum of weights (lengths) of the edges on $P(u, v)$. Define $d(u, v)$ as the length of the shortest path between u and v, the minimum of the lengths of all paths connecting u and v. For each $v \in V$, define $dist(v) = \max_{u \in V} d(u, v)$, and $sum(v) = \sum_{u \in V} d(v, u)$. Node c is a center of the graph if $dist(c) = \min_{v \in G} dist(v)$ and node m is a median of the graph if $sum(m) = \min_{v \in G} sum(v)$.

Let T be a spanning tree of G rooted at some node v'. The *transport* of tree network T is defined as the total length of packet transmissions required to deliver packets from all nodes to node v' by a convergecast process on the tree.

The maximum *delay* of tree network T is the maximum length to be traversed by any packet when traveling from node v' to other nodes. The corresponding solution concepts for convergecast and delay constraints have been considered in the literature as median and center. Choosing the median approach often provides a solution overlooking the nodes at the end of the network (the farthest leaf). The alternative center approach may therefore be applied; that is, choosing the core to be at the center of the tree where the farthest length is the minimum among the nodes. However, locating a core at the center might entail a large increase in length. The problems with using only the center or median as a core have led to a search for a compromised solution concept called *centdian*, where a centdian function presents a kind of compromise between the center and median functions (delay and transport) [12]. The centdian function for node v (given a fixed $\lambda \in [0,1]$) in the network is defined by

$$D_v(\lambda) = \lambda \cdot dist(v) + (1-\lambda) \cdot sum(v) \ ; \ 0 \leq \lambda \leq 1 \tag{1}$$

Another possible definition for the centdian function is:

$$D_v(\alpha) = \alpha \cdot dist(v) + sum(v), \alpha = \frac{\lambda}{1-\lambda} \tag{2}$$

For each nonnegative α, the centdian value, $Cent(\alpha)$, is defined by

$$Cent(\alpha) = \min_{v \in V} D_v(\alpha) \tag{3}$$

Two neighboring nodes in a tree, a and b, have exactly two maximal connected components whose vertex sets are denoted V_a and V_b, by removing the edge $e(a,b)$ ($a \in V_a$ and $b \in V_b$). Furthermore, we define $\Delta(a,b)$ as the difference between the number of nodes in V_a and in V_b. Note that a tree network has at most two center and two median nodes. Wherever there are two median or center nodes, they are neighbors. For the sake of the next lemma provided by [12], we arbitrarily select one median and one center node. All the proofs is omitted due to lack of space.

Lemma 1. *[12] Every node in the (unique) path between the center and the median in a tree network has a λ value that minimizes the centdian function.*

Lemma 2. *[12] When the given tree is a path, the path's median node has to be the middle node if there is an odd number of nodes. If there is an even number of nodes each of the two middle nodes are the median.*

We define a cactus graph CG (sometimes called a cactus tree) as a connected graph where any two simple cycles in the graph have at most one node in common. The node set $V(CG)$ is partitioned into three subsets. A C-node is a node of degree two included in exactly one cycle. An NCG-node is a node not included in any cycle. The remaining vertices, if any, will be referred to as H-vertices, or hinges [3]. A cycle block (CB) consists of p nodes induced by a cycle, where the nodes of the block are denoted clockwise by $R = (r_0, r_1, r_2, ..., r_{p-1})$. A maximal subtree is a subtree for which the subset of NCG- and H-vertices defining it cannot be extended. A graft is a maximal subtree with no two H-vertices belonging to the same cycle [3]. A k-cactus graph is a cactus graph with each block containing at most k edges. We now define an H node radius as its *dist* value and a cycle block radius as the maximum between the *dist* values (radius) of its H nodes and the longest *dist* value in the cycle block. A cycle/graft block containing only one H node is defined as a "leaf" of the cactus graph. It is not difficult to see that a cactus graph consists of blocks, where each block is either a cycle or a graft, and is glued by H vertices [3]. One way to deal with a cactus graph is to construct a tree $T_{CG} = (V'_{CG}, E'_{CG})$, representing the CG structure, where each node in V'_{CG} represents a block or a hinge node in CG [4]. Burkard and Krarup [3] used the CG tree structure to find a median node in a cactus graph, whereas Ben-Moshe et al. [2] used it to solve center problems in a cactus graph. For our cactus network we assume that all the nodes share the same frequency band, and time is divided into equal size slots that are grouped into frames. Thus, the study is conducted in the context of TDMA (Time Division Multiple Access). In TDMA wireless sensor networks, a transmission scenario is valid if and only if it satisfies the following three conditions: First, a node is not allowed to transmit and receive simultaneously. Second, a node cannot receive from more than one neighboring node at the same time and a node receiving from a neighboring node should be spatially separated from any other transmitter by at least some distance D. However, if nodes use unique signature sequences (i.e., a joint TDMA and CDMA (Code Division Multiple Access) scheme), then the second and third conditions may be dropped, and the first condition only characterizes a valid transmission scenario. Thus, our MAC layer is based on TDMA scheduling [7,10,24], such that collisions and interferences do not occur. In the case where we are using Aloha, CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) or 802.11 as the MAC layer, we are assuming that after a finite number of tries (in case of collision, error, failure) the node will succeed in transmitting the message. We also assume the following: The network is asynchronous, so that each node can start the algorithm at any time or upon receiving a message. Messages are guaranteed to be received within some predefined delay and processes have comparable computational speeds. In a time unit a node can receive messages, perform local computation, and send some messages (broadcast for example). Each node has a unique ID in range $[0 \ldots n-1]$. The computation time of each node is negligible compared to the send/receive times. The "free" calculation is bounded in time; the nodes cannot calculate all the topology of the network free but can do some calculations such as calculate its sum and dist value. When a node is sending a message to its neighbors (using the wireless link), it will use broadcast transmission. Moreover, when a node is sending a message to one of its neighbors it will use the unicast transmission and the id of the neighbor as the MAC address.

1.2 Related Work

As defined above, a center node in a cactus graph minimizes the function $dist(v)$. Lan et al. [17] studied the center problem on general cactus graphs and showed a linear time algorithm. Recently, Ben-Moshe et al. [2] studied a more general model of general cactus graphs where nodes are associated with nonnegative weights. For this general model, they [2] presented an $O(n \log n)$ time algorithm to solve the node weighted 1-center problem, and an $O(n \log^3 n)$ time algorithm to solve the continuous node weighted 2-center problem. Zmazek and Zerovnik [27] presented a linear algorithm estimating the traffic on a cactus graph, computing the sum of all delays on cactus graphs. Das and Pal [6] found the maximum and minimum heights spanning trees on a cactus graph in linear time. Note, however, that all the algorithms above cannot be applied directly for wireless sensor networks since these algorithms are sequential and not based on local updates.

As defined above, a median problem in a cactus graph seeks a node minimizing the function *sum*. Similarly to the case of the center, there are linear time algorithms for solving the median problem on 3-cactus graphs [14]. Lan and Wang [16] showed that the median problem in 4-cactus graphs can be solved as efficiently as on trees. Burkard and Krarup [3] presented a linear time algorithm for the median problem in a cactus graph with positive and negative node weights. Recently, Hatzl [13] presented a linear time algorithm for the median problem on wheel graphs and cactus graphs, where a wheel graph is a graph consisting of a cycle of order $p-1$ and an additional vertex that is connected by an edge to each of the cycle vertices. For more information on *median* and *center* problems in cactus graphs see [2,13,17].

The centdian problem is well known in the context of the facility location problem, see [22]. Dvir and Segal [9,8] were the first to deal with the centdian function as expressed by Eq. (1) in the context of wireless ad hoc networks and wireless sensor networks.

1.3 Our Contribution

As mentioned, all existing cactus graph algorithms may not be applied directly to sensor networks since they are not based on local updates. To the best of our knowledge, no sequential or distributed algorithm presented so far has been shown to construct a centdian node in a cactus graph. We investigate the properties of a centdian node in a cycle graph and present interesting observations on the behavior of a centdian using a lower envelope [15,20] of the centdian functions of the nodes. Moreover, we show algorithms to determine centdian nodes for all λ values (α values) in a cycle graph that works in $O(n \log n)$ time (sequential solution) and $O(n)$ time with $O(n \log n)$ messages (distributed solution). We then consider the behavior of a centdian node in a cactus graph, and present new efficient algorithms for constructing all centdian nodes in cactus graphs that run in $O(n \log n)$ time (sequential solution) and $O(n)$ time with $O(n \log n)$ messages (distributed solution).

2 Centdian in a Cactus Graph

The purpose of this section is to explore the centdian nodes' behavior in a cactus graph. Dvir and Segal [9,8] presented a distributed algorithms for a centdian node/structure in a tree topology. In order to solve the problem in cactus graphs we wanted to understand first the behavior of the centdian node in a cycle graph and based on that to design an algorithm to find a centdian node in a cactus graph. Therefore, we first show algorithms to determine centdian nodes for all λ values (α values) in a cycle graph that works in $O(n \log n)$ time (sequential solution) and $O(n)$ time with $O(n \log n)$ messages (distributed solution). Finally, we consider the behavior of a centdian node in a cactus graph, and present a new efficient algorithms for constructing all centdian nodes in cactus graphs that used the cycle graph algorithm as its last step, run in $O(n \log n)$ time (sequential solution), and $O(n)$ time with $O(n \log n)$ messages (distributed solution).

2.1 Finding a Centdian Node in a Cycle Graph: Sequential Algorithm

We first compute the values of $sum(v)$ for all nodes v of the cycle graph in linear time based on the algorithm in [3]. Then, we compute the values of $dist(v)$ for all nodes of the cycle graph in linear time based on the algorithm in [17].

It is well known that in the case of a tree topology both the *dist* and *sum* functions are convex. In particular, every local minimum solution of the *dist* (*sum*) function is a global minimum. Also, the set of nodes minimizing the *dist* (*sum*) function consists of at most a pair of nodes. Moreover, these nodes are neighbors, i.e., they induce a connected subgraph (edge). In contrast, in the case of cycle topology, local solutions are not necessarily global and in the case of cycle topology the set of minimizers of *dist* (*sum*) function does not necessarily induce a connected subgraph.

Thus, contrary to the case of tree topology, where we can find the centdian node by starting from the median node and searching for the optimal node with the minimum centdian function on the path between the median and center nodes [9], we cannot apply the same technique to cactus graphs. In [12], Halpern proved that while $\lambda < 0.5$, the centdian node coincides with the median node in a tree network. This no longer holds for a cycle graph, which leads us to the conclusion that in a cycle graph a node $t \neq m, c$ can be a centdian node in some range of λ, $0 < \lambda < 1$.

Remark 1. A centdian node in a cycle graph can zigzag between upper and lower paths $P(m, c)$ for various values of λ. Meaning, for various λ values the centdian node is not in the same path between the median and the center ($\lambda = 0, \lambda = 1$).

Finally, given a cycle graph with n nodes, we give an $O(n \log n)$ algorithm to construct the centdian nodes for all values of α.

Lemma 3. *For any range of α, all the centdian nodes can be found in $O(n \log n)$ time in a cycle graph.*

2.2 Finding a Centdian Node in a Cycle Graph: Distributed Algorithm

In the following section we present a distributed algorithm to find centdian nodes in a cycle graph in $O(n)$ time with $O(n \log n)$ messages. In order to do this, we have to find a way to provide distributed solutions for each of the algorithms in Section 2.1. As the basic step of our algorithm, we find a leader and the size of the cycle graph by using the distributed algorithm given by Awerbuch [1].

To find the median and center nodes in a distributed manner, the leader sends $\underline{info(leader = id, tempDistVector = [e],\ nodes = 1)}$ messages to its neighbors, where id is the unique ID of the leader and e the length of the edge between the leader and its neighbor. When node u receives an $\underline{info}$ message from one of its neighbors, it saves the *tempDistVector* and *nodes* values, and sends a $\underline{info(leader = id, tempDistVector = tempDistVector, e], nodes++)}$ message to another neighbor. The *tempDistVector* holds the edge lengths between the nodes receiving the $\underline{info}$ message. When node z receives $\underline{info}$ messages from both its neighbors it has complete knowledge of the cycle graph. Using the vectors from both messages, node z can calculate its *dist* and *sum* values in the cycle graph. After the calculation, node z sends the $\underline{info}$ messages to its neighbors. Thus, at the end of this process each node knows its *dist* and *sum* values.

We then need to choose the nodes with the minimum *dist* values to be the centers of the cycle graph and the nodes with the minimum *sum* values to be the medians of the cycle graph. Therefore, the leader needs to send a message to the cycle graph to collect the *dist* and *sum* values and to recognize the median/center nodes. By receiving a $\underline{median/center}$ message from the leader of the cycle graph, each node knows whether it serves as a median/center node. From all the median nodes the leader chooses the median with the minimum ID as the new leader of the cycle graph.

Finally, the median leader starts the following operation of computing the lower envelope for finding the centdian nodes of the cycle graph, using the knowledge that each node has its own *dist* and *sum* value. The median leader starts a procedure to build the lower envelope centdian functions of the cycle graph by sending a $\underline{LowerEnv(LF)}$ message to one of its neighbors, where the $LF = D_{leader}(\alpha)$. When node u receives a $\underline{LowerEnv(LF)}$ message it merges its centdian function with the current lower envelope, calculates the new lower envelope function, and propagates $\underline{LowerEnv(LF)}$ message to another neighbor. At the end of this process the leader has obtained the lower envelope function and can find all the centdian nodes in the cycle graph. The running time of the different parts of the algorithm that find the centdian nodes in a cycle graph is $O(n)$ time and $O(n \log n)$ messages. First, each of the algorithms starts with finding a leader in the cycle graph and the cycle graph size using [1] in $O(n)$ time and $O(n \log n)$ messages. To find the centdian nodes in a cycle graph, each node calculates its *dist* and *sum* values, this can be done in $O(n)$ time using $O(n)$ messages. Then, to calculate the lower envelope of the centdian functions of the nodes in the cycle graph we need additional $O(n)$ time using $O(n)$ messages. From the lower envelope function we can calculate the centdian nodes for a given α range.

2.3 Finding a Centdian Node in a Cactus Graph: Sequential Algorithm

In what follows we present a number of observations and then explain the sequential algorithm for finding the block containing the centdian nodes (centdian block) in a cactus graph and finding all centdian nodes in the cactus graph in $O(n \log n)$ time.

First, from Chen et al. [4] there exists a single block of the cactus graph containing all the median nodes (median block). Similarly, from Chen et al. [5] there exists a single block of the cactus graph containing all the center nodes (center block). The effort to find the median and center block is linear [4,5,2]. Suppose, first, that the median block and the center block coincide. From the definition of centdian function it follows that for each α, all the respective centdian nodes are also in the common block. We can then apply the above results to construct the centdian function in $O(n \log n)$ time. Suppose next that the median block and the center block do not coincide.

Definition 1. *Let P be the unique path connecting the pair of nodes of T_{CG} corresponding to the median block and center block. The set of blocks of the cactus graph corresponding to the nodes of P is called the sausage (S) of the cactus graph.*

Lemma 4. *For any α value, the set of centdian nodes of the cactus graph has to be located on the sausage subgraph.*

To understand the relation between the H nodes and the centdian nodes in a cactus graph, given a range of α, we present the following Lemmata:

Lemma 5. *Let $H_1, H_2, ..., H_t$, be the sequence of H nodes on the path P of T_{CG} connecting the median block and the center block, where H_1 belongs to the median block and H_t belongs to the center block. From the discussion in [4,5], it follows that $dist(H_1) > dist(H_2) > ... > dist(H_t)$ and $sum(H_1) < sum(H_2) < ... < sum(H_t)$.*

Lemma 6. *For any $H_i \in S$, $1 \leq i \leq t$, let $[\alpha'_i, \alpha''_i]$ be the maximal interval such that for each α in the interval $Cent(\alpha) = D_{H_i}(\alpha)$. Then this interval is non-empty, and $\alpha''_i \leq \alpha'_{i+1}$.*

Lemma 7. *For each i, $1 \leq i < t$ and $\alpha'_i < \alpha < \alpha'_{i+1}$, the centdian block corresponding to α is the single block in S containing both H_i, H_{i+1}*

Lemma 8. *Let B be a block in the sausage S ($B \subseteq S$) where B is not the median or center block of the cactus graph and V_B is the node set of B. Then the set of the centdian nodes in B can be found in $O(|V_B|)$ time.*

Lemma 9. *Suppose that B is either the median block or the center block of the sausage S and V_B are the node set of B. The set of the centdian nodes in B can be found in $O(|V_B| \log |V_B|)$ time.*[1]

2.4 Finding Centdian in a Cactus Graph: Distributed Algorithm

In the following section we present a new distributed algorithm for finding centdian nodes in a cactus graph in $O(n)$ time with $O(n \log n)$ messages of sizes $O(\log n)$ bits. For this purpose, we define node x as the cactus graph leader, which can be found by using the distributed algorithm in [1]. Each node in the cactus graph having more than two neighbors needs to determine whether it is an H or an NCG node. The following distributed algorithm recognizes the H nodes.

The main idea is to apply a Depth First Search (DFS) tour to the cactus graph. Node x sends a $\underline{findH(id)}$ message to one of its neighbors, where id is the unique leader ID. When a node u, having only one neighbor, receives a $\underline{findH}$ message, it sends the message back to its neighbor. When a node v having more than one neighbor receives a $\underline{findH}$ message from one of its neighbors, it randomly chooses one of its unmarked edges (apart from the receiving edge), marks it, and sends the $\underline{findH}$ message from the chosen edge. If a node has no more unmarked edges, it sends a $\underline{findH}$ message back, through the DFS tour backtracking path. A node that sends a $\underline{findH}$ message from one of its edges and receives a message on another can conclude that both edges are on the same cycle. Thus, by performing a DFS tour on the cactus graph a $\underline{findH}$ message travels on the edges no more than $2 * |E|$ times. At the end of the algorithm, each node determines whether it is an H node, and if so, calculates the number of cycles it belongs to and identifies the neighbors in each cycle.

After a node u identifies itself as an H node, we have to compute the number of nodes in the cycle blocks with u and the number of H nodes in each cycle block containing u. Each H node sends a $\underline{countCycle(id, i = 1, h = 1)}$ message to one of its neighbors in each cycle block, where id is the unique node ID, i represents the number of nodes so far in this cycle block and h is the number of H nodes detected so far in this cycle block. When C node z receives a $\underline{countCycle}$ message, it sends a $\underline{countCycle(id,\ i{+}{+},\ h)}$ message to another neighbor. When an H node v receives a $\underline{countCycle}$ message it checks whether $id == id_v$. If so, it saves the i value as the number of nodes and the h value as the number of H nodes in this cycle block. If not, it sends a $\underline{countCycle(id,\ i{+}{+},\ h{+}{+})}$ message to another neighbor belonging to this cycle.

In order to find the median blocks in a cactus graph in a distributed manner, each H node needs to compute the number of nodes in each of its subgraphs. This can be computed by propagating information from the cactus graph leaves to a root node, and then from the root back to the leaves. At the end, each node

[1] Note that the problem of finding all centdian nodes (for all α values) in a cactus graph can be solved in $O(n \log n)$ time. Observe that the cycle block algorithm runs in $O(|V_B| \log |V_B|)$ time while the other stages of the cactus algorithm are linear. Therefore, any improvement in the runtime of the cycle block will imply improvement for the cactus case.

will determine the number of nodes in each of its subgraphs. We distinguish between the cases of NCG nodes and H nodes. An NCG node with one neighbor sends a $\underline{subGraphNodes(id,\ i = 1)}$ to its neighbor, where id is its unique node ID. Any other NCG node receiving a $\underline{subGraphNodes}$ message waits until it receives $\underline{subGraphNodes}$ messages from all but one of its neighbors. Then, it saves the number of nodes in each of its subgraphs and sends a $\underline{subGraphNodes(id,\ \sum i + 1)}$ message on its last edge, with the sum of all i values it received plus one (itself).

Each H node has to wait until receiving the information from all the H nodes in each of its cycles and then propagates the information to the rest of the cactus graph. Note that the neighbors of an H node in a graft block are the closest neighbor nodes, while the neighbors of an H node in a cycle block are the H nodes in this cycle. An H node that has a leaf cycle sends a $\underline{subGraphNodes(id, numInCycle)}$ message under the following conditions, where $numInCycle$ is the number of nodes in the subgraph of H; If the H node has only one additional graft or cycle (not leaf) block, it sends a $\underline{subGraphNodes(id,\ numInCycle = i)}$ message to its neighbor in the block, where i is the number of nodes in the leaf cycle. If the H node has more than one graft or cycle block, it waits until it receives $\underline{subGraphNodes}$ messages from all but one of its neighbors, saves the information about the number of nodes in each subgraph, and sends a $\underline{subGraphNodes(id,\ numInCycle = \sum i + 1)}$ message, with the sum of all the i values it receives plus one (itself) to its last neighbor. When an H node u (without a leaf cycle) receives a $\underline{subGraphNodes}$ message with $id \neq id_u$, u decreases its h value of this cycle by one and checks whether h equals one. If not, u saves the information about its subgraph. If yes, u sends a $\underline{subGraphNodes}$ message under the condition of an H node with leaf cycle as explained above. An H node z receiving $\underline{subGraphNodes}$ messages from all its neighbors concludes that it is the root, saves the information about the number of nodes in each subgraph, and sends its ID as the root ID and the information about the number of nodes in each subgraph to its neighbors. Each node (H or NCG) receiving $\underline{subGraphNodes}$ messages from all its neighbors with root ID, saves the information about the number of nodes in each subgraph and sends the information to its neighbors. Therefore, at the end of this process, each H node knows the number of nodes in each of its subgraphs. Then, using the algorithm from [13], we can search in a distributed manner for either a cycle block in which all its H nodes have $cut(H) \geq 0$, or for an NCG node v in which $\Delta(v, w) \geq 0$ for all w neighbors of v are the median blocks.

The distributed algorithm for finding the center blocks in a cactus graph is similar to the above algorithm. The main difference is that each H node propagates its current $dist$ value instead of the number of nodes in each block. At the end, each H node calculates its radius. Then, in each cycle block one of the H nodes is selected as the leader of the cycle block [1], and sends a $\underline{radius}$ message in its cycle block to collect the radii of the H nodes. With this information each cycle block leader can calculate the radius of its cycle block and check whether its block has more than one subgraph with the maximum radius [2]. Finally, the leader of the cactus graph can find, in a distributed manner, the cycle blocks that have more than one subgraph with the maximum radius to be the center blocks. The next stage is to find the sausage of a cactus graph. Node u, which is the H leader of the median block, sends *findSausage(id, TTL=n)* messages to its neighbors. A C or NCG node receiving a *findSausage* message, sends it to all its neighbors. An H node receiving the *findSausage* message, marks the edge receiving from the *findSausage* message, decreases the TLL by one and propagates the *findSausage* message to its neighbors. When the *findSausage* message is received by node v, which is the H leader of the center block, it sends *findSausage(id, TTL=n)* messages backward with its own ID. Each H node that receives both messages from different blocks (median and center) concludes that it belongs to the sausage. Thus, using the algorithm described above we can find the sausage S. Finally, to find the centdian function for each H node in the sausage we can use Lemma 6. Then, for a given λ value, we can use Lemma 7 to recognize the relevant H nodes by sending a message from the leader of the cactus graph to the H nodes in S. Moreover, for finding all centdian nodes in the cycle containing the relevant H nodes we can use the distributed algorithm in Section 2.2.

The running time of the different parts of the algorithm that find the centdian nodes in the cactus graph is $O(n)$ time and $O(n \log n)$ messages. First, each of the algorithms starts with finding a leader in the cycle block/cactus graph and the cycle block/cactus graph size using [1] in $O(n)$ time and $O(n \log n)$ messages. To find the centdian nodes in a cactus graph, each node determines whether it is an H node, and this can be done in $O(n)$ time using $O(n)$ messages. Then, each H node will evaluate the number of nodes in each of its subgraphs in $O(n)$ time using $O(n)$ messages and, in additional $O(n)$ time with $O(n)$ messages we can find the sausage S. Finally, to calculate the centdian nodes in the relevant block we need $O(n)$ time using $O(n)$ messages.

References

1. Awerbuch, B.: Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In: Annual ACM Symposium on Theory of Computing, New York, USA, July 1987, pp. 230–240 (1987)
2. Ben-Moshe, B., Bhattacharya, B.K., Shi, Q., Tamir, A.: Efficient algorithms for center problems in cactus networks. Theoretical Computer Science 378(3), 237–252 (2007)
3. Burkard, R.E., Krarup, J.: A linear algorithm for the pos/neg-weighted 1-median problem on cactus. Computing 60(3), 498–509 (1998)
4. Chen, M.L., Francis, R.L., Lawrence, J.F., Lowe, T.J., Tufekci, S.: Block-vertex duality and the 1-median problem. Networks 15(4), 395–412 (1986)
5. Chen, M.L., Francis, R.L., Lowe, T.J.: The 1-center problem: Exploiting block structure. Transport Science 22, 259–269 (1988)
6. Das, K., Pal, M.: An optimal algorithm to find maximum and minimum height spanning trees on cactus graphs. Advanced Modeling and Optimization 10(1), 121–134 (2008)
7. Deb, B., Nath, B.: On the node-scheduling approach to topology control in ad hoc networks. In: ACM MOBIHOC, pp. 14–26 (2005)

8. Dvir, A., Segal, M.: The (k, l)-coredian tree for ad hoc networks. Journal of Ad Hoc and Sensor Wireless Networks 6(1-2), 123–144 (2008)
9. Dvir, A., Segal, M.: Placing and maintaining a core node in wireless ad hoc sensor networks. Wireless Communications and Mobile Computing (2009) (to appear)
10. ElBatt, T.A., Ephremides, A.: Joint scheduling and power control for wireless ad-hoc networks. In: IEEE INFOCOM, pp. 976–984 (2002)
11. Ergen, M.: WTRP - wireless token ring protocol. M.sc. thesis, Electrical Engineering and Computer Science, Berkeley, California, USA (2002)
12. Halpern, J.: Finding minimal center-median convex combination (cent-dian) of a graph. Management Science 24, 534–544 (1978)
13. Hatzl, H.: Median problems on wheels and cactus graphs. Computing 80(4), 377–393 (2007)
14. Kincaid, R.K., Maimon, O.Z.: A note on locating a central vertex of a 3-cactus graph. Computers and Operations Research 17(3), 315–320 (1990)
15. Klein, R.: Algorithmische Geometrie. Addison Wesley, Reading (1996)
16. Lan, Y.F., Wang, Y.L.: An optimal algorithm for solving the 1-median problem on weighted 4-cactus graphs. European Journal of Operational Research 122(3), 602–610 (2000)
17. Lan, Y.F., Wang, Y.L., Suzuki, H.: A linear-time algorithm for solving the center problem on weighted cactus graphs. Information Processing Letters 71(5), 205–212 (1999)
18. Lee, D., Puri, A., Varaiya, P., Attias, R., Sengupta, R., Tripakis, S.: A wireless token ring protocol for ad-hoc networks. In: IEEE Aerospace Conference, Montana, USA (2002)
19. Luo, H., Ye, F., Cheng, J., Lu, S., Zhang, L.: TTDD: Two-tier data dissemination in large-scale wireless sensor networks. Wireless Networks 11(1), 161–175 (2005)
20. Sharir, M., Agarwal, P.K.: Davenport-Schinzel Sequences and their Geometric Applications. Cambridge University Press, Cambridge (1995)
21. Soochang, P., Bongsoo, K., Euisin, L., Donghun, L., Younghwan, C., Sang-Ha, K.: A novel communication architecture to support mobile users in wireless sensor fields. In: IEEE Vehicular Technology Conference, Dublin, Ireland, April 2007, pp. 66–70 (2007)
22. Tamir, A., Puerto, J., Perez-Brito, D., Rodriguez-Chia, A.M.: The pareto set for the doubly weighted center-median path problem on a tree. In: The Institute for Operations Research and the Management Sciences, September 2005, pp. 1–28 (2005)
23. Wang, H., Kong, P.-Y., Guan, W.S.K.: A robust and energy efficient routing scheme for wireless sensor networks. In: International Conference Workshops on Distributed Computing Systems, DC, USA, July 2006, pp. 83–89 (2006)
24. Wieselthier, J.E., Nguyen, G.D., Ephremides, A.: Algorithms for energy-effcient multicasting in static ad hoc wireless networks. ACM MONET 6(3), 251–263 (2001)
25. Xianpu, S., Yanling, Z., Jiandong, L.: Wireless dynamic token protocol for MANET. In: Parallel Processing Workshops, Xian, China, September 2007, pp. 1–5 (2007)
26. Yicka, J., Mukherjeea, B., Ghosal, D.: Wireless sensor network survey. Computer Networks 52(12), 2292–2330 (2008)
27. Zmazek, B., Zerovnik, J.I.: Estimating the traffic on weighted cactus networks in linear time. In: International Conference on Information Visualisation, London, UK, July 2005, pp. 536–541 (2005)

Twisted Jacobi Intersections Curves

Rongquan Feng[1,⋆], Menglong Nie[1], and Hongfeng Wu[2,⋆⋆,⋆⋆⋆]

[1] LMAM, School of Mathematical Sciences, Peking University, Beijing 100871, P.R. China
[2] Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, P.R. China
fengrq@math.pku.edu.cn, hustnml@163.com, whfmath@gmail.com

Abstract. In this paper, the twisted Jacobi intersections which contains Jacobi intersections as a special case is introduced. We show that every elliptic curve over the prime field with three points of order 2 is isomorphic to a twisted Jacobi intersections curve. Some fast explicit formulae for twisted Jacobi intersections curves in projective coordinates are presented. These explicit formulae for addition and doubling are almost as fast as the Jacobi intersections. In addition, the scalar multiplication can be more effective in twisted Jacobi intersections than in Jacobi intersections. Moreover, we propose new addition formulae which are independent of parameters of curves and more effective in reality than the previous formulae in the literature.

Keywords: elliptic curves, Jacobi intersections, twisted Jacobi intersections, scalar multiplication, cryptography.

1 Introduction

Elliptic curve cryptosystems were proposed by Miller (1986) and by Koblitz (1987) which relies on the difficulty of elliptic curve discrete logarithmic problem. One of the main operations and challenges in elliptic curve cryptosystem is the scalar multiplication. The speed of scalar multiplication plays an important role in the efficiency of the whole system. Elliptic curves can be represented in different forms. To obtain faster scalar multiplications, several elliptic curve representations have been considered in the last two decades. The detail of previous works can be find in [1,3,8].

Jacobi intersections curve is the intersection of two quadratic surfaces in three dimensional space with a point on it. The scalar multiplication on Jacobi intersections show competitive efficiency in scalar multiplication, such as faster doubling and tripling operations. Chudnovsky and Chudnovsky [5] proposed fast doubling and addition formulae for Jacobi intersections in projective coordinates. After that, Liardet and Smart [9], and Bernstein and Lange [1] presented slightly

⋆ Supported by NSF of China (No. 10990011).
⋆⋆ Corresponding author
⋆⋆⋆ China Postdoctoral Science Foundation funded project.

© Springer-Verlag Berlin Heidelberg 2010

faster formulae. Hisil etc. [7] presented faster tripling formulae. Some slightly faster formulae with a trick can also be found in [8].

In this paper, the Jacobi intersections is generalized to "twisted Jacobi intersections" which contains Jacobi intersections as a special case. It is shown that every elliptic curve over the prime field with three points of order 2 is isomorphic to a twisted Jacobi intersections curve. Some fast explicit formulae for twisted Jacobi intersections curves in projective coordinates are presented. These explicit formulae for addition and doubling are almost as fast in the general case as they are for the Jacobi intersections. In addition, the scalar multiplication can be more effective in twisted Jacobi intersections than in Jacobi intersections. Moreover, we propose new addition formulae which are independent of parameters of curves and more effective in reality than the previous formulae in the literature.

This paper is organized as follows. In Section 2, the Jacobi intersections is reviewed, the twisted Jacobi intersections is introduced, and each twisted Jacobi intersections is a twist of a Jacobi intersections is proved. It is shown that every elliptic curve over the prime field with three points of order 2 is isomorphic to a twisted Jacobi intersections curve. In Section 3, the Jacobi intersections addition law is generalized to that for the twisted Jacobi intersections curves, and the explicit addition formulae and formulae independent of parameters of curves are proposed. The Jacobi versus twisted Jacobi is given in Section 4, and the conclusion is in Section 5.

2 Jacobi Intersections and Twisted Jacobi Intersections

In this section we briefly review Jacobi intersections curves and the Jacobi intersections addition law. We then introduce twisted Jacobi intersections curves and discuss their relations to Jacobi intersections curves.

Jacobi intersections

Throughout the paper we consider elliptic curves over a non-binary field K, i.e., a field K whose characteristic is not 2.

A Jacobi intersection form elliptic curve over K is defined by

$$\begin{cases} u^2 + v^2 = 1 \\ bu^2 + w^2 = 1, \end{cases}$$

where $b \in K$ with $b(1-b) \neq 0$. A point (u, v, w) on a Jacobi intersections curve is represented as $(U : V : W : Z)$ satisfying

$$U^2 + V^2 = Z^2, \;\; bU^2 + W^2 = Z^2$$

and $(u, v, w) = (U/Z, V/Z, W/Z)$. Here $(U : V : W : Z) = (\lambda U : \lambda V : \lambda W : \lambda Z)$ for any nonzero $\lambda \in K$. The negative of $(U : V : W : Z)$ is $(-U : V : W : Z)$. The neutral element $(0, 1, 1)$ is represented as $(0 : 1 : 1 : 1)$. The reader is refereed to [5] for more details on Jacobi intersections curves.

The affine version of the unified addition formulae, i.e., that can handle generic doubling, simplifying protection against side-channel attacks, are given by

$$(u_3, v_3, w_3) = (u_1, v_1, w_1) + (u_2, v_2, w_2),$$

where

$$u_3 = \frac{u_1v_2w_2 + u_2v_1w_1}{v_2^2 + u_2^2w_1^2},\ v_3 = \frac{v_1v_2 - u_1w_1u_2w_2}{v_2^2 + u_2^2w_1^2},\ w_3 = \frac{w_1w_2 - bu_1v_1u_2v_2}{v_2^2 + u_2^2w_1^2}.$$

The point addition formulae in projective homogenous coordinates are given by

$$(U_3 : V_3 : W_3 : Z_3) = (U_1 : V_1 : W_1 : Z_1) + (U_2 : V_2 : W_2 : Z_2),$$

where

$$U_3 = U_1Z_1V_2W_2 + V_1W_1U_2Z_2,\quad V_3 = V_1Z_1V_2Z_2 - U_1W_1U_2W_2$$

$$W_3 = W_1Z_1W_2Z_2 - bU_1V_1U_2V_2,\ Z_3 = Z_1^2V_2^2 + U_2^2W_1^2.$$

Twisted Jacobi intersections

Definition 1. *A twisted Jacobi intersection form elliptic curve over K is defined by*

$$\begin{cases} au^2 + v^2 = 1 \\ bu^2 + w^2 = 1, \end{cases}$$

where $a, b \in K$ with $ab(a-b) \neq 0$. A Jacobi intersection elliptic curve is a twisted Jacobi intersection curve with $a = 1$.

The twisted Jacobi intersection curve $E_{a,b}:\ au^2 + v^2 = 1,\ bu^2 + w^2 = 1$ is a quadratic twist of the Jacobi intersection curve $E_{1,b/a}:\ \bar{u}^2 + \bar{v}^2 = 1,\ (b/a)\bar{u}^2 + \bar{w}^2 = 1$. The map $(u, v, w) \mapsto (\bar{u}/\sqrt{a}, \bar{v}, \bar{w})$ is an isomorphism from $E_{a,b}$ to $E_{1,b/a}$ over $K(\sqrt{a})$. Thus if a is a square in K then $E_{a,b}$ is isomorphic to $E_{1,b/a}$ over K. More generally, $E_{a,b}$ is a quadratic twist of $E_{\bar{a},\bar{b}}$ for any $\bar{a}, \bar{b}$ satisfying $\bar{b}/\bar{a} = b/a$. Conversely, every quadratic twist of a twisted Jacobi intersection curve is isomorphic to a twisted Jacobi intersection curve, i.e., the set of isomorphism classes of twisted Jacobi intersection curves is invariant under quadratic twists.

Theorem 1. *Let K be a field with char$(K) \neq 2$ and $E_{a,b}:\ au^2 + v^2 = 1,\ bu^2 + w^2 = 1$ be a twisted Jacobi intersection form curve define over K with $ab(a-b) \neq 0$. Then $E_{a,b}$ is a smooth curve and isomorphic to an elliptic curve of the form $E : y^2 = x(x-a)(x-b)$ over K.*

The proof of Theorem 1 appears in the full version of the paper [6].

Theorem 2. *Let K be a field with char$(K) \neq 2$. Then every elliptic curve over K having three K-rational points of order 2 is isomorphic to a twisted Jacobi intersections curve.*

Proof. Let E be an elliptic curve over K having three K-rational points of order 2. Let $(\theta_1, 0), (\theta_2, 0)$ and $(\theta_3, 0)$ be these three distinct points of order 2 on the Weierstrass curve E, i.e., $y^2 = x^3 + a_2x^2 + a_4x + a_6 = (x - \theta_1)(x - \theta_2)(x - \theta_3)$. Replacing (x, y) by $(x+\theta_1, y)$ yields the equation of the form $y^2 = x(x-a)(x-b)$, where $a = \theta_2 - \theta_1, b = \theta_3 - \theta_1$. Therefore every elliptic curve over K having three $K-$rational points of order 2 is isomorphic to a twisted Jacobi intersections curve by Theorem 1. □

3 Arithmetic on Twisted Jacobi Intersections

Let K be a non-binary field. In this section we present fast explicit formulae for addition and doubling on twisted Jacobi intersections curves over K.

Theorem 3. *Let $P = (u_1, v_1, w_1)$, $Q = (u_2, v_2, w_2)$ be two points on a twisted Jacobi intersections elliptic curve $E_{a,b}$: $au^2 + v^2 = 1$, $bu^2 + w^2 = 1$, and let $R = P + Q := (u_3, v_3, w_3)$. Then the affine version of the unified addition formulae are given by*

$$u_3 = \frac{u_1v_2w_2 + u_2v_1w_1}{v_2^2 + au_2^2w_1^2},\ v_3 = \frac{v_1v_2 - au_1w_1u_2w_2}{v_2^2 + au_2^2w_1^2},\ w_3 = \frac{w_1w_2 - bu_1v_1u_2v_2}{v_2^2 + au_2^2w_1^2}.$$

Especially, if $P = Q$ and $R = 2P := (u_3, v_3, w_3)$, then

$$u_3 = \frac{2u_1v_1w_1}{v_1^2 + au_1^2w_1^2},\ v_3 = \frac{v_1^2 - au_1^2w_1^2}{v_1^2 + au_1^2w_1^2},\ w_3 = \frac{w_1^2 - bu_1^2v_1^2}{v_1^2 + au_1^2w_1^2}.$$

The identity element is $(0, 1, 1)$. The negative of the point (u, v, w) is $(-u, v, w)$.

Proof. For the correctness of the addition law, observe that it coincides with the Jacobi intersections addition law on

$$\bar{u}^2 + v^2 = 1,\ \frac{b}{a}\bar{u}^2 + w^2 = 1,$$

with $\bar{u} = \sqrt{a}u$. These formulae also work for doubling. □

Theorem 4. *Let K be a field of odd characteristic. Let $E_{a,b}$: $au^2+v^2 = 1, bu^2+w^2 = 1$ be a twisted Jacobi intersections curve over K. Let $P = (u_1, v_1, w_1)$ and $Q = (u_2, v_2, w_2)$ be points on $E_{a,b}$. If ab is not a square in K, or if -1 is a square in K and neither a nor b is a square in K, then $v_2^2 + au_2^2w_1^2 \neq 0$.*

Proof. If $v = w = 0$, then $au^2 = bu^2$ and $a = b$, therefore ab is a square in K, contradict to ab is not a square in K. Therefore at most one in $\{u, v, w\}$ is equal to 0 for a point (u, v, w) on $E_{a,b}$. Thus if $u_2 = 0$, then $v_2^2+au_2^2w_1^2 = v_2^2 \neq 0$. If $u_1 = 0$, then $w_1^2 = 1$, and $v_2^2+au_2^2w_1^2 = v_2^2+au_2^2 = 1$. Let $u_1u_2 \neq 0$, assume that ab is not a square in K. If $v_2^2+au_2^2w_1^2 = 0$, then $au_2^2+v_2^2-(v_2^2+au_2^2w_1^2) = au_2^2(1-w_1^2) = 1$. Thus $1-w_1^2 = 1/au_2^2 = bu_1^2$. therefore $ab = (1/u_1u_2)^2$ is a square in K, contradict to the assumption. Now assume that neither a nor b is a square in K, then $w_1v_2 \neq 0$. If $v_2^2 + au_2^2w_1^2 = 0$, then $a = -(v_2/u_2w_1)^2$ is square in K since -1 is a square in K, which is a contradiction. □

Note that Theorem 4 shows that if ab is not a square in K, then the twisted addition formulae is complete. But generally, both a and b are non-squares in K. Therefore ab is not a square in K is not a reasonable assumption when $a \neq 1$. But in this case, if -1 is a square in K, then the above twisted addition formulae is also complete.

When using projective homogenous coordinates to eliminate field inversions, each point is represented by the quadruplet $(U : V : W : Z)$ which satisfies the equations

$$aU^2 + V^2 = Z^2,\ bU^2 + W^2 = Z^2,$$

and corresponds to the affine point $(U/Z,\ V/Z,\ W/Z)$ with $Z \neq 0$.

Theorem 5. *Let $P = (U_1 : V_1 : W_1 : Z_1)$, $Q = (U_2 : V_2 : W_2 : Z_2)$ be two points on the twisted Jacobi intersections elliptic curve $E_{a,b}$: $aU^2 + V^2 = Z^2$, $bU^2 + W^2 = Z^2$, and let $R = P + Q := (U_3 : V_3 : W_3 : Z_3)$. Then the projective version of the unified addition formulae are given by*

$$U_3 = U_1Z_1V_2W_2 + V_1W_1U_2Z_2, \quad V_3 = V_1Z_1V_2Z_2 - aU_1W_1U_2W_2,$$

$$W_3 = W_1Z_1W_2Z_2 - bU_1V_1U_2V_2,\ Z_3 = Z_1^2V_2^2 + aU_2^2W_1^2.$$

The identity element is $(0 :\ 1 :\ 1 :\ 1)$. The negative of the point $(U : V : W : Z)$ is $(-U : V : W : Z)$. □

Note that $Z_1^2(Z_2^2 - V_2^2) = aZ_1^2U_2^2$ and $aU_2^2(bU_1^2 + W_1^2) = aU_2^2Z_1^2$. We have $Z_1^2V_2^2 + aU_2^2W_1^2 = Z_1^2Z_2^2 - abU_1^2U_2^2$ which can be used to simplify the formulae.

Especially, the above theorem gives the following doubling formulae.

$$U_3 = 2U_1V_1W_1Z_1, \qquad V_3 = V_1^2Z_1^2 - aU_1^2W_1^2,$$

$$W_3 = W_1^2Z_1^2 - bU_1^2V_1^2, \quad Z_3 = V_1^2Z_1^2 + aU_1^2W_1^2.$$

Note that $bU_1^2 = Z_1^2 - W_1^2$, and $V_1^2W_1^2 = W_1^2(Z_1^2 - aU_1^2) = W_1^2Z_1^2 - aU_1^2W_1^2$. We have the second doubling formulae

$$\begin{aligned} U_3 &= 2U_1V_1W_1Z_1,\ V_3 = V_1^2Z_1^2 - aU_1^2W_1^2, \\ W_3 &= 2W_1^2Z_1^2 - V_1^2Z_1^2 - aU_1^2W_1^2, \qquad (1) \\ Z_3 &= V_1^2Z_1^2 + aU_1^2W_1^2. \end{aligned}$$

Moreover, from

$$\begin{aligned} W_1^2Z_1^2 - bU_1^2V_1^2 &= W_1^2(aU_1^2 + V_1^2) - (Z_1^2 - W_1^2)V_1^2 = aU_1^2W_1^2 + 2V_1^2W_1^2 - V_1^2Z_1^2 \\ &= aU_1^2W_1^2 - V_1^2Z_1^2 + 2W_1^2(bU_1^2 + W_1^2 - aU_1^2) \\ &= aU_1^2W_1^2 - V_1^2Z_1^2 + 2bW_1^2U_1^2 + 2W_1^4 - 2aU_1^2W_1^2 \\ &= -aU_1^2W_1^2 - V_1^2Z_1^2 + 2(bU_1^2W_1^2 + W_1^4), \end{aligned}$$

we have the third doubling formulae

$$\begin{aligned} U_3 &= 2U_1V_1W_1Z_1,\ V_3 = V_1^2Z_1^2 - aU_1^2W_1^2, \\ W_3 &= -aU_1^2W_1^2 - V_1^2Z_1^2 + 2(bU_1^2W_1^2 + W_1^4), \\ Z_3 &= V_1^2Z_1^2 + aU_1^2W_1^2. \end{aligned} \tag{2}$$

Addition in projective coordinates. By Theorem 5, the following formulae compute $(U_3 : V_3 : W_3 : Z_3) = (U_1 : V_1 : W_1 : Z_1) + (U_2 : V_2 : W_2 : Z_2)$ in $13M + 2S + 5D$ costs, i.e., 13 field multiplications, 2 squarings and 5 multiplications by the curve constant a and b, or in $14M + S + 4D$ costs. We denote the two algorithms by "AProjective.1" and "AProjective.2".

$$\begin{aligned} A &= U_1V_1;\ B = W_1Z_1;\ C = U_2V_2;\ D = W_2Z_2;\ E = U_1W_2; \\ F &= V_1Z_2;\ G = W_1U_2;\ H = Z_1V_2;\ J = AD;\ K = BC; \\ U_3 &= (H + F)(E + G) - J - K; \\ V_3 &= (H + E)(F - aG) - J + aK; \\ W_3 &= (B - bA)(C + D) + bJ - K; \\ Z_3 &= H^2 + a \cdot G^2 = H^2 + aG \cdot G. \end{aligned}$$

If the points represented by the sextuplet (U, V, W, Z, UV, WZ), then the addition formula can by modified by: $(U_3 : V_3 : W_3 : Z_3 : A_3 : B_3) = (U_1 : V_1 : W_1 : Z_1 : A_1 : B_1) + (U_2 : V_2 : W_2 : Z_2 : A_2 : B_2)$, where $A_1 = U_1V_1, B_1 = W_1Z_1, A_2 = U_2V_2, B_2 = W_2Z_2$. The cost are $11M + 2S + 5D$ or $12M + S + 4D$. We denote the two algorithms bye "MProjective.1" and "MProjective.2".

$$\begin{aligned} C &= U_1W_2; D = V_1Z_2; E = W_1U_2; F = Z_1V_2; G = A_1B_2; H = B_1A_2; \\ U_3 &= (D + F)(C + E) - G - H; \\ V_3 &= (C + F)(D - aE) - G + aH; \\ W_3 &= (B_1 - bA_1)(A_2 + B_2) + bG - H; \\ Z_3 &= F^2 + a \cdot E^2 = F^2 + aE \cdot E; \\ A_3 &= U_3V_3; B_3 = W_3Z_3. \end{aligned}$$

Note that, if $a = \varepsilon^2$ is a square element in the field, then $Z_3 = (F + \varepsilon E)^2 - 2\varepsilon H$, the cost is $11M + 1S + 6D$.

Doubling 1 in projective coordinates. The following formulae compute $(U_3 : V_3 : W_3 : Z_3) = 2(U_1 : V_1 : W_1 : Z_1)$ in $3M + 4S + 1D$ by using formulae (1), where the $1D$ is a multiplication by a:

$$\begin{aligned} A &= V_1Z_1;\ B = A^2;\ C = U_1W_1;\ D = C^2;\ E = 2(W_1Z_1)^2; \\ U_3 &= (A + C)^2 - B - D;\ V_3 = B - aD; \\ W_3 &= E - B - aD;\ Z_3 = B + aD. \end{aligned}$$

Doubling 2 in projective coordinates. The following formulae compute $(U_3 : V_3 : W_3 : Z_3) = 2(U_1 : V_1 : W_1 : Z_1)$ in $2M + 5S + 2D$ by using formulae (2), where the $2D$ are multiplications by a and by b:

$$\begin{aligned} A &= V_1Z_1;\ B = A^2;\ C = U_1W_1;\ D = C^2;\ E = W_1^4; \\ U_3 &= (A + C)^2 - B - D;\ V_3 = B - aD; \\ W_3 &= 2(bD + E) - B - aD;\ Z_3 = B + aD. \end{aligned}$$

Doubling 1 in projective coordinates with $Z_1 = 1$. The following formulae compute $(U_3 : V_3 : W_3 : Z_3) = 2(U_1 : V_1 : W_1 : 1)$ in $1M + 4S + 1D$ by using formulae (1), where the $1D$ is a multiplication by a:

$$\begin{aligned}A &= V_1;\ B = A^2;\ C = U_1W_1;\ D = C^2;\ E = 2W_1^2;\\ U_3 &= (A + C)^2 - B - D;\ V_3 = B - aD;\\ W_3 &= E - B - aD;\ Z_3 = B + aD.\end{aligned}$$

Doubling 2 in projective coordinates with $Z_1 = 1$. The following formulae compute $(U_3 : V_3 : W_3 : Z_3) = 2(U_1 : V_1 : W_1 : 1)$ in $1M + 5S + 2D$ by using formulae (2), where the $2D$ are multiplications by a and by b:

$$\begin{aligned}A &= V_1;\ B = A^2;\ C = U_1W_1;\ D = C^2;\ E = W_1^4;\\ U_3 &= (A + C)^2 - B - D;\ V_3 = B - aD;\\ W_3 &= 2(bD + E) - B - aD;\ Z_3 = B + aD.\end{aligned}$$

Note that $V_1^2Z_1^2 = Z_1^2(Z_1^2 - aU_1^2) = Z_1^4 - aU_1^2Z_1^2$, $U_1^2W_1^2 = U_1^2(Z_1^2 - bU_1^2) = U_1^2Z_1^2 - bU_1^4$ and $W_1^4 = (Z_1^2 - bU_1^2)^2 = Z_1^4 + b^2U_1^4 - 2bU_1^2Z_1^2$. We have the following doubling formulae:

$$\begin{aligned}U_3 &= 2U_1Z_1V_1W_1,\ V_3 = abU_1^4 - 2aU_1^2Z_1^2 + Z_1^4,\\ W_3 &= abU_1^4 - 2bU_1^2Z_1^2 + Z_1^4,\ Z_3 = Z_1^4 - abU_1^4.\end{aligned} \tag{3}$$

Doubling 3 in projective coordinates with $Z_1 = 1$. The following formulae compute $(U_3 : V_3 : W_3 : Z_3) = 2(U_1 : V_1 : W_1 : 1)$ in $2M + 2S + 3D$ by using formulae (3), where the $3D$ are multiplications by a, b, ab:

$$\begin{aligned}A &= U_1^2;\ B = A^2;\ C = Z_1^2;\ D = C^2;\ E = (U_1 + Z_1)^2 - A - C;\\ F &= (A + C)^2 - B - D;\ G = abB;\\ U_3 &= V_1W_1E;\ V_3 = G - aF + D;\\ W_3 &= G - bF + D;\ Z_3 = D - G.\end{aligned}$$

The comparison of the costs of above doubling formulae in this paper to those in previous works is listed in Table 1.

Table 1. Algorithm comparison with other algorithms in Doubling

Coordinates	Source of algorithms	Doubling	Doubling($Z_1 = 1$)
Jacobi Intersections	[9]	4M+3S	-
Jacobi Intersections	[2]	3M+4S	2M+4S
Jacobi Intersections	[7]	2M+5S+1D	-
Twisted Jacobi Intersections	Doubling 1	3M+4S+1D	1M+4S+1D
Twisted Jacobi Intersections	Doubling 2	2M+5S+2D	1M+5S+2D
Twisted Jacobi Intersections	Doubling 3	2M+6S+3D	2M+2S+3D

Doubling formulae independent of a and b. From $aU_1^2 = Z_1^2 - V_1^2$ and $bU_1^2 = Z_1^2 - W_1^2$, we have the following doubling formulae which are independent of the parameters a and b:

$$U_3 = 2U_1V_1W_1Z_1,\ V_3 = V_1^2Z_1^2 - Z_1^2W_1^2 + V_1^2W_1^2,$$

$$W_3 = W_1^2Z_1^2 - V_1^2Z_1^2 + V_1^2W_1^2,\ Z_3 = V_1^2Z_1^2 + Z_1^2W_1^2 - V_1^2W_1^2.$$

Addition formulae independent of a and b

Theorem 6. *Let $P = (u_1, v_1, w_1)$, $Q = (u_2, v_2, w_2)$ be two different points on the twisted Jacobi intersections elliptic curve $E_{a,b}:\ au^2 + v^2 = 1,\ bu^2 + w^2 = 1$, and let $R = P + Q = (u_3, v_3, w_3)$. Then the addition formulae can be given by*

$$u_3 = \frac{u_1^2 - u_2^2}{u_1v_2w_2 - v_1w_1u_2},\ v_3 = \frac{u_1v_1w_2 - w_1u_2v_2}{u_1v_2w_2 - v_1w_1u_2},\ w_3 = \frac{u_1w_1v_2 - v_1u_2w_2}{u_1v_2w_2 - v_1w_1u_2}.$$

Proof. From

$$\begin{aligned}
&(u_1^2 - u_2^2)(v_2^2 + au_2^2w_1^2) = u_1^2v_2^2 + au_1^2w_1^2u_2^2 - u_2^2v_2^2 - au_2^2u_2^2w_1^2 \\
&= u_1^2v_2^2 + (1 - v_1^2)w_1^2u_2^2 - u_2^2v_2^2 - (1 - v_2^2)u_2^2w_1^2 \\
&= u_1^2v_2^2 - u_2^2v_2^2 + u_2^2v_2^2w_1^2 - v_1^2w_1^2u_2^2 \\
&= u_1^2v_2^2 - u_2^2v_2^2(1 - w_1^2) - v_1^2w_1^2u_2^2 \\
&= u_1^2v_2^2 - bu_1^2u_2^2v_2^2 - v_1^2w_1^2u_2^2 \\
&= u_1^2v_2^2(1 - bu_2^2) - v_1^2w_1^2u_2^2 \\
&= u_1^2v_2^2w_2^2 - v_1^2w_1^2u_2^2 \\
&= (u_1v_2w_2 + v_1w_1u_2)(u_1v_2w_2 - v_1w_1u_2),
\end{aligned}$$

we have
$$\frac{u_1^2 - u_2^2}{u_1v_2w_2 - v_1w_1u_2} = \frac{u_1v_2w_2 + v_1w_1u_2}{v_2^2 + au_2^2w_1^2}.$$

From

$$\begin{aligned}
&(u_1v_1w_2 - w_1u_2v_2)(u_1v_2w_2 + v_1w_1u_2) \\
&= u_1^2v_1v_2w_2^2 + u_1u_2v_1^2w_1w_2 - u_1u_2v_2^2w_1w_2 - u_2^2v_1v_2w_1^2 \\
&= u_1^2v_1v_2(1-bu_2^2) + u_1u_2w_1w_2(1-au_1^2) - u_1u_2(1-au_2^2)w_1w_2 - u_2^2v_1v_2(1 - bu_1^2) \\
&= u_1^2v_1v_2 - au_1^2u_1u_2w_1w_2 - u_2^2v_1v_2 + au_2^2u_1u_2w_1w_2 \\
&= (u_1^2 - u_2^2)(v_1v_2 - au_1w_1u_2w_2),
\end{aligned}$$

we have

$$\begin{aligned}
\frac{v_1v_2 - au_1w_1u_2w_2}{v_2^2 + au_2^2w_1^2} &= \frac{(u_1v_1w_2 - w_1u_2v_2)(u_1v_2w_2 + v_1w_1u_2)}{(u_1^2 - u_2^2)(v_2^2 + au_2^2w_1^2)} \\
&= \frac{u_1v_1w_2 - w_1u_2v_2}{\frac{(u_1^2 - u_2^2)(v_2^2 + au_2^2w_1^2)}{u_1v_2w_2 + v_1w_1u_2}} = \frac{u_1v_1w_2 - w_1u_2v_2}{u_1v_2w_2 - v_1w_1u_2}.
\end{aligned}$$

Again, from

$$\begin{aligned}
&(u_1w_1v_2 - v_1u_2w_2)(u_1v_2w_2 + v_1w_1u_2)\\
&= u_1^2w_1w_2v_2^2 + u_1u_2v_1v_2w_1^2 - u_1u_2v_1v_2w_2^2 - u_2^2v_1^2w_1w_2\\
&= u_1^2w_1w_2(1-au_2^2) + u_1u_2v_1v_2(1-bu_1^2) - u_1u_2v_1v_2(1-bu_2^2) - u_2^2w_1w_2(1-au_1^2)\\
&= u_1^2w_1w_2 - bu_1^2u_1v_1u_2v_2 - u_2^2w_1w_2 + bu_2^2u_1v_1u_2v_2\\
&= (u_1^2 - u_2^2)(w_1w_2 - bu_1v_1u_2v_2),
\end{aligned}$$

we have

$$\frac{w_1w_2 - bu_1v_1u_2v_2}{v_2^2 + au_2^2w_1^2} = \frac{(u_1w_1v_2 - v_1u_2w_2)(u_1v_2w_2 + v_1w_1u_2)}{(u_1^2 - u_2^2)(v_2^2 + au_2^2w_1^2)}$$

$$= \frac{u_1w_1v_2 - v_1u_2w_2}{\dfrac{(u_1^2 - u_2^2)(v_2^2 + au_2^2w_1^2)}{u_1v_2w_2 + v_1w_1u_2}} = \frac{u_1w_1v_2 - v_1u_2w_2}{u_1v_2w_2 - v_1w_1u_2}.$$

The theorem follows from Theorem 3. □

The formulae fail for point doubling. In addition, there are exceptional cases. For example, when $2P = 2Q$, then the formulae cannot work. The above formulae in projective homogenous coordinates are given by the following theorem.

Theorem 7. *Let $P = (U_1 : V_1 : W_1 : Z_1)$, $Q = (U_2 : V_2 : W_2 : Z_2)$ be two different points on the twisted Jacobi intersections elliptic curve $E_{a,b} :\ aU^2 + V^2 = Z^2,\ bU^2 + W^2 = Z^2$, and let $R = P + Q = (U_3 : V_3 : W_3 : Z_3)$. Then the addition formulae can be given by*

$$U_3 = U_1^2Z_2^2 - Z_1^2U_2^2,\ \ V_3 = U_1V_1W_2Z_2 - W_1Z_1U_2V_2,$$

$$W_3 = U_1W_1V_2Z_2 - V_1Z_1U_2W_2,\ \ Z_3 = U_1Z_1V_2W_2 - V_1W_1U_2Z_2.$$

The projective addition formulae in Theorems 5 and 7 have exceptional points in each case. But the following theorem tells us that the formulae together in Theorems 5 and 7 cover all points.

Theorem 8. *Let $P = (U_1 : V_1 : W_1 : Z_1)$, $Q = (U_2 : V_2 : W_2 : Z_2)$ be two points on the twisted Jacobi intersections elliptic curve $E_{a,b} :\ aU^2 + V^2 = Z^2,\ bU^2 + W^2 = Z^2$ defined over K with $ab(a-b) \neq 0$, let $R = (U_3 : V_3 : W_3 : Z_3)$ and $S = (U_3' : V_3' : W_3' : Z_3')$, where*

$$U_3 = U_1Z_1V_2W_2 + V_1W_1U_2Z_2,\ \ \ V_3 = V_1Z_1V_2Z_2 - aU_1W_1U_2W_2,$$

$$W_3 = W_1Z_1W_2Z_2 - bU_1V_1U_2V_2,\ \ Z_3 = Z_1^2V_2^2 + aU_2^2W_1^2,$$

and

$$U_3' = U_1^2Z_2^2 - Z_1^2U_2^2, \qquad V_3' = U_1V_1W_2Z_2 - W_1Z_1U_2V_2,$$

$$W_3' = U_1W_1V_2Z_2 - V_1Z_1U_2W_2,\ \ Z_3' = U_1Z_1V_2W_2 - V_1W_1U_2Z_2.$$

Then $P + Q = R = S$ if $R = S$, and $P + Q = R$ (or S) if $S = 0$ (or $R = 0$).

Proof. If $R \neq (0,0,0,0)$, then $R \in E_{a,b}$ and $P+Q=R$. Similarly, if $S \neq (0,0,0,0)$, then $S \in E_{a,b}$ and $P+Q=S$. Now assume $R=S=(0,0,0,0)$. Then $U_1Z_1V_2W_2 + V_1W_1U_2Z_2 = 0$ and $U_1Z_1V_2W_2 - V_1W_1U_2Z_2 = 0$. Thus $U_1Z_1V_2W_2 = V_1W_1U_2Z_2 = 0$.

If $U_1 = 0$, then $Z_1^2U_2^2 = 0$ since $U_1^2Z_2^2 - Z_1^2U_2^2 = 0$. Thus $Z_1 = 0$ or $U_2 = 0$. When $Z_1 = 0$, then $V_1 = W_1 = 0$ from $aU^2 + V^2 = Z^2$ and $bU^2 + W^2 = Z^2$. Therefore $P = (0,0,0,0)$, which is contradict to $P \in E_{a,b}$. When $U_2 = 0$, then $V_1Z_1V_2Z_2 = 0$ from $V_3 = V_1Z_1V_2Z_2 - aU_1W_1U_2W_2 = 0$. We can get $Q = (0,0,0,0)$ by the similar argument as above. Contradict to $Q \in E_{a,b}$.

The similar argument works for the cases when $U_2 = 0$, $Z_1 = 0$, $Z_2 = 0$, $V_2 = 0$, $W_2 = 0$, $V_1 = 0$ or $W_1 = 0$.

If $P \neq Q$, From Theorem 7 we know that $P+Q=R=S$ if $R \neq (0,0,0,0)$ and $S \neq (0,0,0,0)$. $\square$

New addition algorithm use Theorem 7. The following formulae compute $(U_3 : V_3 : W_3 : Z_3) = (U_1 : V_1 : W_1 : Z_1) + (U_2 : V_2 : W_2 : Z_2)$ in $15M$, We denote the algorithm by "Independent.1".

$$\begin{aligned}
&A = U_1Z_2;\; B = U_2Z_1;\; C = V_1W_2;\; D = V_2W_1;\\
&E = U_1Z_1;\; F = V_1W_1;\; G = U_2Z_2;\; H = V_2W_2;\\
&U_3 = (A+B)(A-B);\\
&V_3 = AC - BD;\; W_3 = AD - BC;\; Z_3 = EH - FG.
\end{aligned}$$

Note that $U_3 = U_1^2(bU_2^2 + W_2^2) - U_2^2(bU_1^2 + W_1^2) = U_1^2W_2^2 - U_2^2W_1^2$. If the points represented by the sextuplet (U,V,W,Z,UW,VZ), then the addition formula can by modified by: $(U_3 : V_3 : W_3 : Z_3 : M_3 : N_3) = (U_1 : V_1 : W_1 : Z_1 : M_1 : N_1) + (U_2 : V_2 : W_2 : Z_2 : M_2 : N_2)$, where $M_1 = U_1W_1, N_1 = V_1Z_1, M_2 = U_2W_2, N_2 = V_2Z_2$, the cost are $13M$. We denote the algorithm be "MIndependent.2".

$$\begin{aligned}
&A = U_1W_2;\; B = U_2W_1;\; C = V_1Z_2;\; D = V_2Z_1;\\
&U_3 = (A+B)(A-B);\\
&V_3 = AC - BD;\; W_3 = M_1N_2 - M_2N_1;\; Z_3 = AD - BC;\\
&M_3 = U_3W_3, N_3 = V_3Z_3.
\end{aligned}$$

The comparison of the costs of above addition formulae in this paper to those in previous works is listed in Table 2.

Note that, Table 2 show that the addition in twisted Jacobi intersections are almost as fast as that in the Jacobi intersections. The new algorithm based on the formula independent of parameters of curves is more effectively than the best result in literature for Jacobi intersection curves when $D > 0.6M$.

4 Jacobi versus Twisted Jacobi

The twisted Jacobi intersection curve is a generalization of Jacobi intersections, and twisted Jacobi intersection curve cover more elliptic curves than Jacobi intersections curves do. An example in [2] shows that for prime $p = 2^{255} - 19$,

Table 2. Algorithm comparison with other algorithms in addition

Coordinates	Source	Addition	$D = 0M$	$S = D = 1M$
Jacobi Intersections	[9]	13M+2S+1D	14.6M	16M
Jacobi Intersections	[7](projective)	13M+1S+2D	13.8M	16M
Jacobi Intersections	[7](modified)	11M+1S+2D	11.8M	14M
Twisted Jacobi	AProjective.1	13M+2S+5D	14.6M	20M
Twisted Jacobi	AProjective.2	14M+1S+4D	14.8M	19M
Twisted Jacobi	MProjective.1	11M+2S+5D	12.6M	18M
Twisted Jacobi	MProjective.2	12M+1S+4D	12.8M	17M
Twisted Jacobi(a square)	MProjective.2	11M+1S+6D	11.8M	18M
Twisted Jacobi	Independent.1	15M	15M	15M
Twisted Jacobi	MIndependent.2	13M	13M	13M

one multiplication by 121665 and one multiplication by 121666, which together are faster than a multiplication by 208003386839886583686474089955893887370928784529770630033400064708706245363 94 $\equiv$ 121665/121666 (mod p). That is, for a large parameter b of Jacobi intersections curves $U^2 + V^2 = Z^2$, $bU^2 + W^2 = Z^2$, we can choose smaller a' and b' such that the twisted Jacobi intersections $a'U^2 + V^2 = Z^2$, $b'U^2 + W^2 = Z^2$ is quadratic twisted to it, but can save computation costs. For example, in algorithms MProjective.1, if a, b are smaller and $a = \varepsilon^2$ is a square element in the field, then we can omit the multiplications by the small constants. Thus $Z_3 = F^2 + a \cdot E^2 = (F + \varepsilon E)^2 - 2\varepsilon H$, and the algorithm cost $11M + 1S$, which is more efficient than the algorithm in [7](modified).

5 Conclusion

In this paper, the twisted Jacobi intersections which contains Jacobi intersections as a special case is introduced. We show that every elliptic curve over the prime field with three points of order 2 is isomorphic to a twisted Jacobi intersections curve. Some fast explicit formulae for twisted Jacobi intersections curve in projective coordinates are presented. These explicit formulae for addition and doubling are almost as fast as the Jacobi intersections. In addition, the scalar multiplication can be more effective in twisted Jacobi intersections than in Jacobi intersections. Finally, new addition formulae which are independent of parameters of curves are proposed and it can be more effective than the previous results in literature when $D > 0.6M$. At last, we hope the faster point operation formulae on twist Jacobi intersection can be proposed.

References

1. Bernstein, D.J., Lange, T.: Explicit-formulae database, `http://www.hyperelliptic.org/EFD`
2. Bernstein, D.J., Birkner, P., Joye, M., Lange, T., Peters, C.: Twisted Edwards curves. In: Vaudenay, S. (ed.) AFRICACRYPT 2008. LNCS, vol. 5023, pp. 389–405. Springer, Heidelberg (2008)

3. Bernstein, D.J., Lange, T.: Analysis and optimization of elliptic-curve single-scalar multiplication, Cryptology ePrint Archive, Report 2007/455
4. Billet, O., Joye, M.: The Jacobi model of an elliptic curve and side-channel analysis. In: Fossorier, M.P.C., Høholdt, T., Poli, A. (eds.) AAECC 2003. LNCS, vol. 2643, pp. 34–42. Springer, Heidelberg (2003)
5. Chudnovsky, D.V., Chudnovsky, G.V.: Sequences of numbers generated by addition in formal groups and new primality and factorization tests. Advances in Applied Mathematics 7, 385–434 (1986)
6. Feng, R., Nie, M., Wu, H.: Twisted Jacobi Intersections Curves, Full version available at Cryptology ePrint Archive: Report 2009/597, http://eprint.iacr.org/2009/597
7. Hisil, H., Carter, G., Dawson, E.: New formulae for efficient elliptic curve arithmetic. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 138–151. Springer, Heidelberg (2007)
8. Hisil, H., Koon-Ho Wong, K., Carter, G., Dawson, E.: Faster group operations on elliptic curves. In: Brankovic, L., Susilo, W. (eds.) Proc. Seventh Australasian Information Security Conference (AISC 2009), Wellington, New Zealand. CRPIT, vol. 98, pp. 7–19. ACS (2009)
9. Liardet, P.-Y., Smart, N.P.: Preventing SPA/DPA in ECC systems using the Jacobi form. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 391–401. Springer, Heidelberg (2001)
10. Washington, L.C.: Elliptic Curves: Number Theory and Cryptography. CRC Press, Boca Raton (2003)
11. Silverman, J.H.: The Arithmetic of Elliptic Curves. Springer, Heidelberg (1986)

$L(2,1,1)$-Labeling Is NP-Complete for Trees*

Petr A. Golovach[1], Bernard Lidický[2], and Daniël Paulusma[1]

[1] Department of Computer Science, University of Durham,
Science Laboratories, South Road, Durham DH1 3LE, England
{petr.golovach,daniel.paulusma}@durham.ac.uk
[2] Charles University, Faculty of Mathematics and Physics,
DIMATIA and Institute for Theoretical Computer Science (ITI),
Malostranské nám. 2/25, 118 00, Prague, Czech Republic
bernard@kam.mff.cuni.cz

Abstract. An $L(p_1, p_2, p_3)$-labeling of a graph G with span λ is a mapping f that assigns each vertex u of G an integer label $0 \leq f(u) \leq \lambda$ such that $|f(u) - f(v)| \geq p_i$ whenever vertices u and v are of distance i for $i \in \{1, 2, 3\}$. We show that testing whether a given graph has an $L(2, 1, 1)$-labeling with some given span λ is NP-complete even for the class of trees.

1 Introduction

Classical graph coloring involves the labeling of the vertices of some given graph by integers usually called colors such that no two adjacent vertices receive the same color. In many applications the objective is to minimize the number of colors. Graph coloring has been a popular research topic since its introduction as a map coloring problem more than 150 years ago. Some reasons for this are its appealingly simple definition, its large variety of open problems, and its many application areas. Whenever conflicting situations between pairs of objects can be modeled by graphs, and one is looking for a partition of the set of objects in subsets of mutually non-conflicting objects, this can be viewed as a graph coloring problem. This holds for classical settings such as neighboring countries (map coloring) or interfering jobs on machines (job scheduling), as well as for more recent settings like colliding data streams in optical networks (wavelength assignment), or colliding traffic streams (time slot allocation), to name just a few.

The graph coloring problem studied in this paper has its applications in wireless communication. In a wireless network, each transmitter has been assigned a frequency channel for its transmissions. However, two transmissions can interfere if their channels are too close. Whether this happens depends on the physical structure of the network; even if two transmitters use different channels, there still may be interference if the two transmitters are located close to each other. As radio spectrum gets more and more scarce because the number of wireless networks is rapidly increasing, the task is *to minimize the total number of different frequencies while avoiding interference.*

* Supported by EPSRC (EP/GO43434/1) and the Royal Society ((JP090172).

© Springer-Verlag Berlin Heidelberg 2010

A wireless network can be modeled by an undirected graph $G = (V, E)$ with no loops and no multiple edges. The transmitters are represented by vertices and the *distance* $\text{dist}_G(u, v)$ between two transmitters u, v is the number of edges on a shortest path from u to v. A *labeling* of G is a mapping $f : V \to \{0, 1, \ldots\}$ that assigns each vertex of V a *label* $f(v)$ representing a frequency channel (in this setting, the convention is to use "label" instead of "color").

How far channels of two transmitters must be away from each other depends on the distance of these transmitters in the network. We model this by posing extra restrictions on a labeling. This approach is called *distance constrained labeling* and it is done via a *metric graph* H, the vertices of which represent the available channels and are denoted $0, \ldots, |V(H)| - 1$. For positive integers $p_1, p_2, \ldots, p_k$, a labeling f of G with $\{f(u) \mid u \in V(G)\} \subseteq V(H)$ is called an $H(p_1, \ldots, p_k)$-*labeling* if for $i = 1, \ldots, k$

$$\text{dist}_H(f(u), f(v)) \geq p_i \text{ for all } u, v \in V_G \text{ with } \text{dist}_G(u, v) = i.$$

It is natural to assume that frequencies must be farther apart if transmitters are closer; so we restrict ourselves to the case in which $p_1 \geq p_2 \geq \cdots \geq p_k$. We can now translate the earlier mentioned task into the problem:

$H(p_1, \ldots, p_k)$-Labeling
Parameters: $p_1, \ldots, p_k$.
Instance: graphs G and H.
Question: does G have an $H(p_1, \ldots, p_k)$-labeling?

Not only for its practical applications but also because of its many interesting theoretical properties, distance constrained labeling has received much attention in recent literature, in particular the cases in which H is a path or a cycle, respectively. Below we discuss computational complexity issues for the case when H is a path; for a survey on known algorithmic results for other metric graphs we refer to Fiala, Golovach and Kratochvíl [8].

Path (Linear) Metric. Let H be a path $P_{\lambda+1}$ on vertices $0, \ldots, \lambda$ with an edge between vertices i and $i + 1$ for $i = 0, \ldots, \lambda - 1$. Then an $H(p_1, \ldots, p_k)$-labeling is called an $L(p_1, \ldots, p_k)$-*labeling* with *span* λ, and $H(p_1, \ldots, p_k)$-Labeling is formulated as the problem:

$L(p_1, \ldots, p_k)$-Labeling
Parameters: $p_1, \ldots, p_k$.
Instance: a graph G and integer λ.
Question: does G have an $L(p_1, \ldots, p_k)$-labeling with span λ?

The minimum λ such that G has an $L(p_1, \ldots, p_k)$-labeling is denoted $\lambda_{p_1,\ldots,p_k}(G)$. We note that an $L(1)$-labeling of a graph G is a proper coloring of G, and hence $\lambda_1(G) = \chi(G) - 1$, where $\chi(G)$ is the *chromatic number* of G.

Especially, $L(p_1, p_2)$-labelings are well studied, cf. the surveys of of Calamoneri [2] and Yeh [17]. Below we only mention a number of algorithmic and complexity results. Fiala, Kloks and Kratochvíl [10] showed that $L(2, 1)$-Labeling is NP-complete, already for fixed $\lambda \geq 4$. Chang and Kuo [4] presented a nontrivial dynamic programming algorithm to show that $L(2,1)$-Labeling can be solved in polynomial time for trees. Later on, Chang et al. [3] showed that $L(p_1,1)$-Labeling is even polynomially solvable for trees when p_1 is not fixed but part of the input (see also Fiala, Kratochvíl and Proskurowski [12]). However, for any fixed p_1, p_2, the $L(p_1,p_2)$-Labeling problem is NP-complete, even for the class of trees, if $p_2 \geq 2$ and p_2 does not divide p_1, and polynomially solvable otherwise [8]. It is also known that, for fixed $p_1 \geq 2$, $L(p_1,1)$-Labeling is already NP-complete for the class of graphs of treewidth two [7]. This is in contrast to the polynomial time result of Zhou, Kanari and Nishizeki [18] on $L(1,1)$-Labeling for graphs of bounded treewidth (but $L(1,1)$-Labeling is W[1]-hard when parameterized by the treewidth of an input graph [9]).

What about $L(p_1,\ldots,p_k)$-labelings for $k \geq 3$? Zhou, Kanari and Nishizeki [18] showed that $L(1,\ldots,1)$-Labeling can be solved in polynomial time on graphs of bounded treewidth. Bertossi, Pinotti and Rizzi [1] showed the same for the class of interval graphs. For general graphs $L(2,1,1)$-Labeling is NP-complete for any fixed $\lambda \geq 5$ and polynomially solvable otherwise [6]. Up to now, the computational complexity of $L(p_1,1,\ldots,1)$-Labeling for fixed $p_1 \geq 2$ is still *open* for trees, even for $p_1 = 2$ and $k = 3$. However, it is known [14] that the prelabeling extension of $L(2,1,1)$-Labeling is NP-complete for trees (in this variant of the problem some vertices have preassigned labels). It is also known [15] that $L(p_1,1,1)$-Labeling is NP-complete for trees if p_1 is part of the input. Fiala, Golovach and Kratochvíl [6] showed that $\lambda_{2,1,1}(T)$ can be approximated almost tightly for trees. To be more precise, they prove the following result. Here, $\omega(G)$ denotes the size of a maximum clique in a graph G, and G^k denotes the *k-th distance power* of G, i.e., the graph with vertex set $V(G^k) = V(G)$, and edges between any two distinct vertices that are at distance at most k in G.

Proposition 1 ([6]). *For any tree T, $\omega(T^3) - 1 \leq \lambda_{2,1,1}(T) \leq \omega(T^3)$.*

Our result. Despite Proposition 1, we show that $L(2,1,1)$-Labeling is still NP-complete for trees. This is the *first* hardness result on $L(p_1,\ldots,p_k)$-labelings on trees for fixed $p_1,\ldots,p_k$ and $k \geq 3$. We prove it in Section 2 by a reduction from 3-Satisfiability. Notice that contrary to the NP-completeness result for $L(2,1,1)$-Labeling on general graphs [6], which holds for fixed span $\lambda \geq 5$, in our proof the span λ will depend on the size of the 3-SAT instance; for fixed λ, the problem is polynomially solvable on graphs of bounded treewidth [6]. This follows from the observation that, for fixed λ, we can describe an $L(p_1,\ldots,p_k)$-Labeling problem in Monadic Second-Order Logic, and then we can apply the well-known theorem of Courcelle [5] for graph classes of bounded treewidth.

2 L(2,1,1)-Labelings for Trees

We prove the following theorem.

Theorem 1. *The $L(2,1,1)$-Labeling problem is* NP-*complete for the class of trees.*

Before we prove Theorem 1 we first introduce some additional terminology. Let G be a graph. Then $N_G(v)$ denotes the (open) neighborhood of G and $\deg_G(v)$ denotes the degree of vertex $v \in V(G)$. For integers i and j, we define the *interval* $[i,j]$ as the set $\{i, i+1, i+2, \ldots, j\}$. By $[i,j]_{\equiv 2}$ we denote the set of all even integers in the interval $[i,j]$.

2.1 Auxiliary Constructions

We first construct gadgets where some vertices are forced predetermined labels in an arbitrary $L(2,1,1)$-labeling. A set of integers $S \subseteq [0,\lambda]$ is called *symmetric* if for each $i \in S$, $\lambda - i \in S$. Notice that for any $L(p_1,\ldots,p_k)$-labeling l of a graph G of span λ, the mapping $\overline{l}\colon V(G) \to [0,\lambda]$, such that $\overline{l}(v) = \lambda - l(v)$ for $v \in V(G)$, is an $L(p_1,\ldots,p_k)$-labeling of G of span λ too. Hence our gadgets force symmetric sets of labels.

From here we assume that λ is an even positive integer and $\lambda \geq 14$.

We consider a star $K_{1,\lambda-1}$ with the central vertex u. Then a new vertex w is added and joined by an edge with a leaf v of the star. Denote the obtained tree by T_1. We say that w is the *root* of T_1. An example of T_1 is shown in Fig. 4 We need the following properties of T_1.

Lemma 1. *For any $\lambda - L(2,1,1)$-labeling of T_1 with span λ,*

- *the vertex u is labeled by an integer from the set $\{0,\lambda\}$;*
- *if u is labeled by 0 then the root w is labeled by 1 and if u is labeled by λ then w is labeled by $\lambda - 1$.*

For any $i \in \{1, \lambda-1\}$ and any integer $j \in [3, \lambda-3]$, there is a $\lambda - L(2,1,1)$-labeling l of T_1 such that $l(w) = i, l(v) = j$.

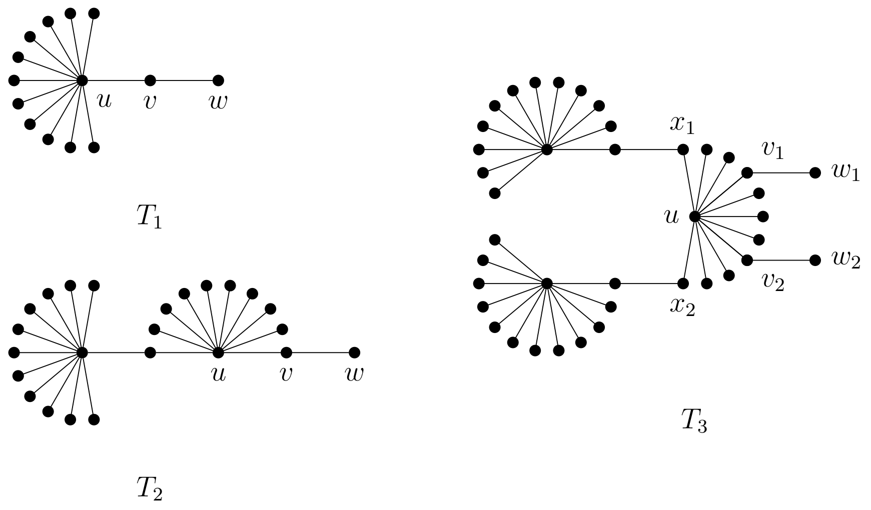


Fig. 1. Gadgets T_1, T_2 and T_3 for $\lambda = 13$

Proof. Since all vertices of $N_{T_1}(u)$ should be labeled by different labels which are 2-distant from the label of u and since $\deg_{T_1}(u) = \lambda - 1$, for any $\lambda - L(2,1,1)$-labeling of T_1, the vertex u can be labeled only either by 0 or λ. Assume that u is labeled by 0. Then vertices of $N_{T_1}(u)$ are labeled by all integers from $[2, \lambda]$. Hence, w should be labeled by 1. Symmetrically, if u is labeled by λ, then w is labeled by $\lambda - 1$.

The second claim of the lemma is proved by the direct check. □

The next gadget is denoted by T_2 and is constructed as follows (see Fig. 1). We introduce a star $K_{1,\lambda-3}$ with the central vertex u and add a copy of T_1 rooted in u. Then a new vertex w is added and joined by an edge with a leaf v of the tree adjacent to u. The vertex w is the *root* of T_2. The properties of T_2 are given in the following lemma.

Lemma 2. *For any $\lambda - L(2,1,1)$-labeling of T_2 with span λ,*

– *the vertex u is labeled by an integer from the set $\{1, \lambda - 1\}$;*
– *if u is labeled by 1 then the root w is labeled by an integer from $\{0, 2\}$ and if u is labeled by $\lambda - 1$ then w is labeled by a label from $\{\lambda - 2, \lambda\}$.*

For any $i \in \{0, 2, \lambda - 2, \lambda\}$ and any integer $j \in [5, \lambda - 5]$, there is a $\lambda - L(2,1,1)$-labeling l of T_2 such that $l(w) = i, l(v) = j$.

Proof. By Lemma 1 the vertex u is labeled either by 1 or $\lambda - 1$. Assume that u is labeled by 1. Since $\deg_{T_1}(u) = \lambda - 2$, for any $\lambda - L(2,1,1)$-labeling of T_2, the vertices $N_{T_2}(u)$ are labeled by all integers from $[3, \lambda]$. Therefore, w should be labeled by 0 or 2. Symmetrically, if u is labeled by $\lambda - 1$, then w is labeled by $\lambda - 2$ or λ.

The second claim of the lemma is proved by the direct check. □

Now we construct the gadget T_3 (see Fig. 1). We consider a star $K_{1,\lambda-2}$ with the central vertex u. Then two copies of T_1 rooted in two different leaves x_1, x_2 of the star are added. Finally we add two vertices w_1, w_2 and join them by edges with two different leaves (v_1 and v_2 respectively) of the constructed tree adjacent to u. We call w_1 and w_2 the *roots* of T_3. The properties of T_3 are summarized in the next lemma.

Lemma 3. *For any $\lambda - L(2,1,1)$-labeling of T_3 with span λ,*

– *the vertex u is labeled by an integer from $[3, \lambda - 3]$;*
– *if u is labeled by i, then roots w_1, w_2 are labeled by labels from $\{i - 1, i + 1\}$.*

For any integer $i \in [3, \lambda - 3]$, any pair of integers $j_1, j_2 \in \{i - 1, i + 1\}$ and any pair of different integers $r_1, r_2 \in [i+3, \lambda-(i+3)]$, there is a $\lambda-L(2,1,1)$-labeling l of T_3 such that $l(u) = i$, $l(w_1) = j_1$, $l(w_2) = j_2$, $l(v_1) = r_1$ and $l(v_2) = r_2$.

Proof. By Lemma 1 the vertices x_1 and x_2 can be labeled either 1 or $\lambda - 1$. Since they must have different labels, one of them is labeled by 1 and the second is labeled by $\lambda - 1$. Hence, u can be labeled only by an integer from $i \in [3, \lambda - 3]$.

Assume that u is labeled by i. For any $\lambda - L(2,1,1)$-labeling of T_3, the vertices $N_{T_3}(u)$ are labeled by all integers from $[0,\lambda] \setminus [i-1, i+1]$. Therefore, w_1 and w_2 can be labeled only by integers from $\{i-1, i+1\}$.

As before, the second claim of the lemma is verified directly. Notice that neighbors of x_1 and x_2 different from u can always be labeled by $i-1$ and $i+1$. □

For further constructions we assume that k is a positive integer and $2 \le k \le \lambda/4 - 2$.

We construct a rooted tree $T(k)$ so that the root can be labeled only by integers from $[2, 2k]_{\equiv 2} \cup [\lambda - 2k, \lambda - 2]_{\equiv 2}$. First we introduce $k-1$ copies of trees T_3. For $i \in \{1, \ldots, k-1\}$, denote by $u^{(i)}, v_1^{(i)}, v_2^{(i)}, w_1^{(i)}, w_2^{(i)}$ the vertices u, v_1, v_2, w_1, w_2 of i-th copy of T_3. Then vertices $w_2^{(i-1)}$ and $w_1^{(i)}$ are identified for $i \in \{2, \ldots, k-1\}$. Finally, a copy of T_2 rooted in $w_1^{(1)}$ is added. Let $u^{(0)}$ and $v^{(0)}$ be the vertices u and v of T_2 respectively. We call $w_2^{(k-1)}$ the *root* of $T(k)$. Construction of $T(k)$ is shown in Fig. 2.

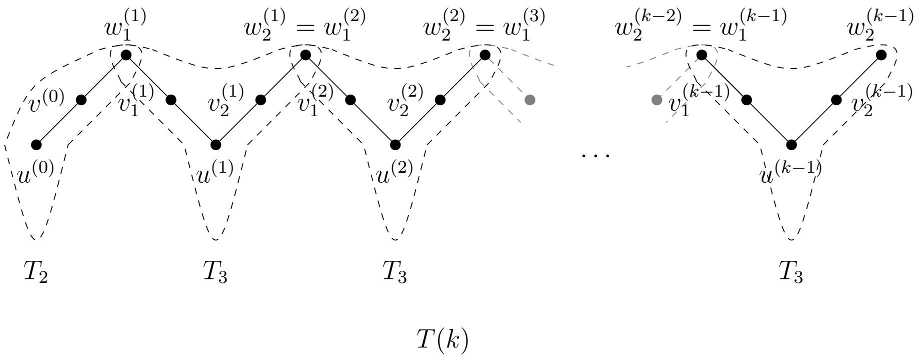


Fig. 2. Gadget $T(k)$

Lemma 4. *For any $\lambda - L(2,1,1)$-labeling of $T(k)$ with span λ,*

- *the root $w_2^{(k-1)}$ is labeled by an integer from $[2, 2k]_{\equiv 2} \cup [\lambda - 2k, \lambda - 2]_{\equiv 2}$;*
- *if $w_2^{(k-1)}$ is labeled by i, then $u^{(k-1)}$ is labeled either $i-1$ or $i+1$ if $i/2 < k$ and $u^{(k-1)}$ is labeled by $i-1$ if $i = k$.*

For any integer $i \in [2, 2k]_{\equiv 2} \cup [\lambda - 2k, \lambda - 2]_{\equiv 2}$ and any integer $r \in [2k+2, \lambda - (2k+2)]$ there is a $\lambda - L(2,1,1)$-labeling l of $T(k)$ such that $l(w_2^{(k-1)}) = i$ and $l(v_2^{(k-1)}) = r$.

Proof. Notice that by Lemma 2 the vertex $w_1^{(1)}$ is labeled by an integer from the set $\{0, 2, \lambda - 2, \lambda\}$. Since by Lemma 3 it cannot be labeled by 0 or λ, this vertex is labeled either by 2 or $\lambda - 2$. Then the first claim of the lemma is proved by the inductive applications of Lemma 3. We use the fact that if $w_1^{(j)}$ is labeled by i then $u^{(i)}$ is labeled by $i-1$ or $i+1$ and $w_2^{(i)}$ is labeled by an integer from $\{i-2, i, i+2\}$.

The second claim immediately follows from Lemmata 2 and 3. It is sufficient to notice that all vertices $v_1^{(j)}$ can be labeled by $r+1$ or $r-1$, the vertices $v_1^{(j)}$ and $v^{(0)}$ can be labeled by r for $j \in \{1, \dots, k-1\}$. □

Using gadgets $T(k)$ it is possible to construct a rooted tree $F(k)$ (see Fig. 3) such that the root can be labeled only by an integer $2k$ or $\lambda - 2k$. We construct a star $K_{1,2k+1}$ with the central vertex v and leaves $w_0, \dots, w_{2k}$. Then four copies of T_2 rooted in w_1, w_2, w_3 and w_4 respectively are introduced, and for each $i \in \{2, \dots, k-1\}$, two copies of $T(i)$ rooted in w_{2i+1} and w_{2i+2} are added. Finally, a copy of $T(k)$ rooted in w_0 is constructed. The vertex w_0 is declared the *root* of $F(k)$.

Lemma 5. *For any $\lambda - L(2,1,1)$-labeling of $F(k)$ with span λ,*

- *the root w_0 is labeled either by $2k$ or $\lambda - 2k$;*
- *the vertices at distance two from the root are labeled by all integers from $[0, 2k-2]_{\equiv 2} \cup [\lambda - (2k-2), \lambda]_{\equiv 2}$ and one vertex is labeled by $2k-1$ or $\lambda - (2k-1)$.*

For any pair of different integers $r_1, r_2 \in [2k+2, \lambda - (2k+2)]$ there is a $\lambda - L(2,1,1)$-labeling l of $F(k)$ such that the vertices adjacent to the root are labeled by r_1 and r_2.

Proof. By Lemma 2 vertices w_1, w_2, w_3, w_4 have to be colored by $0, 2, \lambda-2, \lambda$. By inductive application of Lemma 4 and the fact that all labels of $w_5, \dots, w_{2k}$

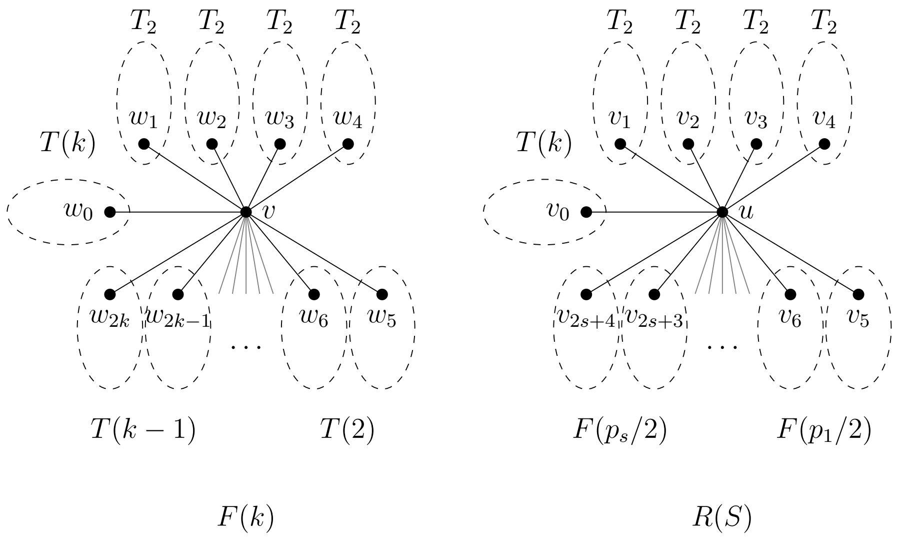


Fig. 3. Gadgets $F(k)$ and $R(S)$

have to be different we conclude that $w_5, \ldots, w_{2k}$ are labeled by all even integers from $[4, 2k-2]_{\equiv 2} \cup [\lambda - (2k-2), \lambda - 4]_{\equiv 2}$. Then again by Lemma 4 the vertex w_0 is labeled either by $2k$ or $\lambda - 2k$ and the vertex at distance two from w_0 in the copy of $T(2k)$ is labeled either by $2k-1$ or $\lambda - (2k-1)$.

The second claim follows from Lemmata 2 and 4, since v can be labeled by r_1 and the other vertices adjacent to $w_0, \ldots, w_{2k}$ can be labeled by r_2. □

We proceed by constructing a rooted tree $R(S)$ such that the root can be labeled only by integers from the set of labels S (see Fig. 3). Let $S \subset [4, 2k]_{\equiv 2} \cup [\lambda - 2k, \lambda - 4]_{\equiv 2}$ be a symmetric set of even integers. Denote by X the set of all integers from $[4, 2k]_{\equiv 2} \setminus S$ and let $X = \{p_1, \ldots, p_s\}$. We construct a star $K_{1,2s+5}$ with the central vertex u and leaves $v_0, \ldots, v_{2s+4}$. Then four copies of T_2 rooted in v_1, v_2, v_3, v_4 respectively are introduced, and for each $i \in \{1, \ldots, s\}$, two copies of $F(p_i/2)$ rooted in v_{2i+3} and v_{2i+4} are added. Finally, a copy of $T(k)$ rooted in v_0 is constructed. The vertex v_0 is declared the *root* of $R(S)$.

Lemma 6. *For any $\lambda - L(2,1,1)$-labeling of $R(S)$ with span λ,*

- *the root v_0 is labeled by an integer from S;*
- *the vertices at distance two from the root are labeled by integers from $[0, 2k] \cup [\lambda - 2k, \lambda]$.*

For any integer $t \in S$ and any pair of different integers $r_1, r_2 \in [2k+2, \lambda - (2k+2)]$ there is a $\lambda - L(2,1,1)$-labeling l of $R(S)$ such $l(v_0) = t$ and the vertices adjacent to the root are labeled by r_1 and r_2.

Proof. By Lemma 2 vertices v_1, v_2, v_3, v_4 have to be colored by $0, 2, \lambda - 2, \lambda$. By Lemma 5 the vertices $v_5, \ldots, v_{2p+4}$ are labeled by all integers from $[4, 2k]_{\equiv 2} \cup [\lambda - 2k, \lambda - 4]_{\equiv 2} \setminus S$. By Lemma 4 the vertex v_0 is labeled by an even integer from $[4, 2k]_{\equiv 2} \cup [\lambda - 2k, \lambda - 4]$ and this vertex is 2-distant from the vertices $v_1, \ldots, v_{2p+4}$. Therefore it can be labeled only by integers from S. The fact that the vertices at distance two from the root are labeled by integers from $[0, 2k] \cup [\lambda - 2k, \lambda]$ immediately follows from Lemmata 2, 3 and 5.

To prove the second claim, let us note that by Lemmata 2 and 5 there are labelings of all copies of T_2 and $F(p_i/2)$ such that the vertices adjacent to the roots of these trees are labeled by r_1. Using Lemma 4 we can notice that there is a labeling of $T(k)$ such that the root is labeled by t and the vertex adjacent to the root is labeled by r_1. It remains to label u by r_2 to receive the $L(2,1,1)$-labeling of $R(S)$ from these labelings of auxiliary gadgets. □

We conclude this part of the proof by the following easy observation.

Lemma 7. *The tree $R(S)$ has $O(\lambda^4)$ vertices.*

2.2 Polynomial Reduction

We proceed with reduction of the well known NP-complete 3-SATISFIABILITY problem [13, problem L02, page 259] to our $L(2,1,1)$-LABELING problem for trees.

Let Φ be a boolean formula in conjunctive normal form that has variables $x_1, x_2, \ldots, x_n$ and clauses $C_1, C_2, \ldots, C_m$. Each clause consists of three literals. We choose $\lambda = 8n + m + 9$ is m is odd and $\lambda = 8n + m + 10$ otherwise.

For each variable x we define the set of integers $X_i = \{4i, 4i+2, \lambda-(4i+2), \lambda-4i\}$ and construct three copies of trees $R(X_i)$ with roots $x_i^{(1)}$, $x_i^{(2)}$ and $x_i^{(3)}$. For each clause C_j we define the set of six integers Y_j as follows. For each literal z in C_j, integers $4i, \lambda-4i$ are included in Y_j if $z = x_i$ and integers $4i+2, \lambda-(4i+2)$ are included in Y_j if z is a negation of the variable x_i for some $i \in \{1, \ldots, n\}$. Then a copy of $R(Y_j)$ with a root y_j is constructed. Finally, we add a vertex u and join it with all vertices $x_i^{(1)}, x_i^{(2)}, x_i^{(3)}$ by edges and with all vertices y_j by paths of length two with middle vertices $v_1, \ldots, v_m$. Denote the obtained tree by T (see Fig. 4).

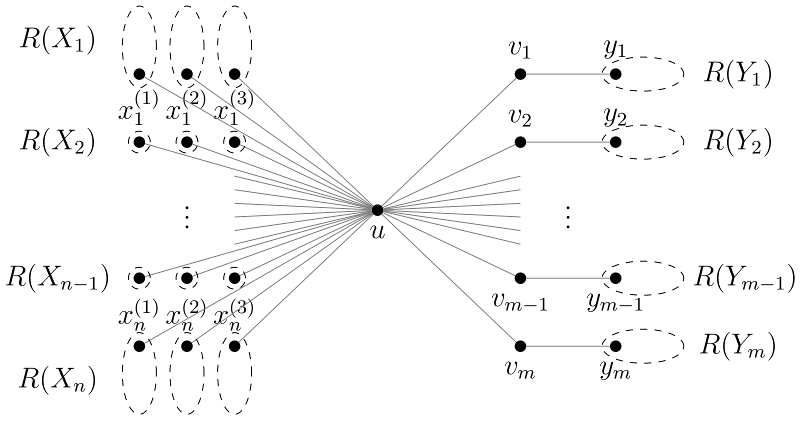


Fig. 4. A tree T

Lemma 8. *The tree T has an $L(2,1,1)$-labeling of span λ if and only if the formula Φ can be satisfied.*

Proof. Suppose that there is a $\lambda - L(2,1,1)$-labeling of T. By Lemma 6 for each $i \in \{1, \ldots, n\}$, vertices $x_i^{(1)}, x_i^{(2)}, x_i^{(3)}$ are labeled by integers from X_i. Since these vertices are 2-distant in T, the labels have to be different. Hence exactly one label from X_i is not used for the labeling of $x_i^{(1)}, x_i^{(2)}, x_i^{(3)}$. Denote this label by p_i. If $p_i = 4i$ or $p_i = \lambda - 4i$ then we assume that $x_i = true$ and $x_i = false$ otherwise. We prove that these values give a truth assignment which satisfies Φ. By Lemma 6 the vertex y_j is labeled by an integer from the Y_j. Assume that y_j is labeled by $4i$ or $\lambda - 4i$ for some $i \in \{1, \ldots, n\}$. This label should be different from the labels of vertices $x_i^{(1)}, x_i^{(2)}, x_i^{(3)}$. Therefore C_j contains the literal x_i and $x_i = true$. Similarly, if y_j is labeled by $4i + 2$ or $\lambda - (4i + 2)$ for some $i \in \{1, \ldots, n\}$, then this label is not used for the labeling of $x_i^{(1)}, x_i^{(2)}, x_i^{(3)}$, i.e. C_j contains the literal $\overline{x}_i$ and $x_i = false$.

Assume now that the formula Φ has a satisfying truth assignment and variables $x_1, \ldots, x_n$ have corresponding values. Notice that sets $X_1, \ldots, X_n$ do not intersect. We label $x_i^{(2)}$ by $\lambda - (4i + 2)$ and $x_i^{(3)}$ by $\lambda - 4i$ for $i \in \{1, \ldots, n\}$. The vertex $x_i^{(1)}$ is labeled by $4i + 2$ if $x_i = true$, and $x_i^{(1)}$ is labeled by $4i$ if $x_i = false$. Each clause C_j contains a literal $z = true$. If $z = x_i$ for some $i \in \{1, \ldots, n\}$ then Y_j contains the integer $4i$ and this label was not used for the labeling of $x_i^{(1)}, x_i^{(2)}, x_i^{(3)}$. We use $4i$ to label y_j. Similarly, if $z = \overline{x}_i$ for some $i \in \{1, \ldots, n\}$ then Y_j contains the integer $4i + 2$ and since this label was not used for the labeling of $x_i^{(1)}, x_i^{(2)}, x_i^{(3)}$, we label y_j by $4i + 2$. By lemma 6 these labeling of roots of trees $R(S)$ can be extended to the labelings of all vertices of these trees such that the vertices at distance two from the root are labeled by integers from $[0, 4n + 2] \cup [\lambda - (4n + 2), \lambda]$ and the vertices adjacent to the roots are labeled by $4n+4$ and $4n+6$. We extend this labeling to the $L(2, 1, 1)$-labeling of T by labeling u by $4n + 5$ and $v_1, \ldots, v_m$ by $4n + 7, \ldots, 4n + m + 6$. $\square$

To conclude the proof of Theorem 1 it remains to note that it follows from Lemma 7 that T has $O((n + m)^5)$ vertices.

3 Conclusions

We showed that the $L(2, 1, 1)$-Labeling problem is NP-complete for trees (while $L(2, 1)$-Labeling can be solved in polynomial time for this graph class [4]). We expect that $L(p_1, p_2, p_3)$-Labeling remains NP-complete for all p_1, p_2, p_3 such that $p_1 > p_3$, but this does not follow directly from our results. Determining the computational complexity of the corresponding problem on trees for the *cyclic metric*, which we explain below, is still open.

Cyclic Metric. Let H be a cycle $C_{\lambda+1}$ on vertices $0, \ldots, \lambda$ with an edge between vertices i and $i + 1$ for $i = 0, \ldots, \lambda$ (modulo $\lambda + 1$). Then an $H(p_1, \ldots, p_k)$-labeling is called a $C(p_1, \ldots, p_k)$-*labeling* with *span* λ, and the corresponding decision problem is denoted $C(p_1, \ldots, p_k)$-Labeling.

Fiala and Kratochvíl [11] showed that $C(2, 1)$-Labeling is NP-complete, already for fixed $\lambda \geq 6$. Just as for the result for the path metric, Fiala, Golovach and Kratochvíl [7] showed that as a matter of fact $C(2, 1)$-Labeling is already NP-complete for the class of graphs with treewidth 2. On the positive side, Liu and Zhu [16] presented a closed formula for the minimum λ such that a tree has a $C(p_1, p_2)$-labeling. This immediately implies that $C(p_1, p_2)$-Labeling can be solved in polynomial time for trees, even if p_1 and p_2 are both part of the input. Similarly to the case of the path metric, Fiala, Golovach and Kratochvíl [6] showed that for any tree T, $\omega(T^3) - 1 \leq c_{2,1,1}(T) \leq \omega(T^3)$, where $c_{2,1,1}(G)$ is the minimum λ such that G has a $C(2, 1, 1)$-labeling. This leads us to the following question: what is the computational complexity of $C(2, 1, 1)$-Labeling on trees?

References

1. Bertossi, A.A., Pinotti, M.C., Rizzi, R.: Channel assignment on strongly-simplicial graphs. In: 17th International Symposium on Parallel and Distributed Processing, p. 222. IEEE Computer Society, Washington (2003)
2. Calamoneri, T.: The $L(h, k)$-labelling problem: a survey and annotated bibliography. Comput. J. 49, 585–608 (2006)
3. Chang, G.J., Ke, W.T., Kuo, D., Liu, D.F., Yeh, R.K.: On $L(d, 1)$-labelings of graphs. Discrete Math. 220, 57–66 (2000)
4. Chang, G.J., Kuo, D.: The $L(2, 1)$-labeling problem on graphs. SIAM J. Discrete Math. 9, 309–316 (1996)
5. Courcelle, B.: The monadic second-order logic of graphs. I: recognizable sets of finite graphs. Inform. and Comput. 85, 12–75 (1990)
6. Fiala, J., Golovach, P.A., Kratochvíl, J.: Elegant distance constrained labelings of trees. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 58–67. Springer, Heidelberg (2004)
7. Fiala, J., Golovach, P.A., Kratochvíl, J.: Distance constrained labelings of graphs of bounded treewidth. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 360–372. Springer, Heidelberg (2005)
8. Fiala, J., Golovach, P.A., Kratochvíl, J.: Computational complexity of the distance constrained labeling problem for trees. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 294–305. Springer, Heidelberg (2008)
9. Fiala, J., Golovach, P.A., Kratochvíl, J.: Parameterized complexity of coloring problems: treewidth versus vertex cover. In: TAMC 2009. LNCS, vol. 5532, pp. 221–230. Springer, Heidelberg (2009)
10. Fiala, J., Kloks, T., Kratochvíl, J.: Fixed-parameter complexity of lambda-labelings. Discrete Appl. Math. 113, 59–72 (2001)
11. Fiala, J., Kratochvíl, J.: Partial covers of graphs. Discuss. Math. Graph Theory 22, 89–99 (2002)
12. Fiala, J., Kratochvíl, J., Proskurowski, A.: Distance constrained labeling of precolored trees. In: Restivo, A., Ronchi Della Rocca, S., Roversi, L. (eds.) ICTCS 2001. LNCS, vol. 2202, pp. 285–292. Springer, Heidelberg (2001)
13. Garey, M.R., Johnson, D.R.: Computers and Intractability. Freeman, New York (1979)
14. Golovach, P.A.: Systems of pairs of q-distant representatives, and graph colorings. Zap. Nauchn. Sem. S.-Peterburg. Otdel. Mat. Inst. Steklov (POMI) 293, 5–25 (2002)
15. Golovach, P.A.: Distance-constrained labelings of trees. Vestn. Syktyvkar. Univ. Ser. 1 Mat. Mekh. Inform. 6, 67–78 (2006)
16. Liu, D., Zhu, Z.: Circular distance two labellings and circular chromatic numbers. Ars Combin. 69, 177–183 (2003)
17. Yeh, R.K.: A survey on labeling graphs with a condition at distance two. Discrete Math. 306, 1217–1231 (2006)
18. Zhou, X., Kanari, Y., Nishizeki, T.: Generalized vertex-coloring of partial k-trees. IEICE Trans. Fundamentals of Electronics, Communication and Computer Sciences E83-A, 671–678 (2000)

Complexity of Paths, Trails and Circuits in Arc-Colored Digraphs

Laurent Gourvès[1,2], Adria Lyra[3,4], Carlos Martinhon[3,⋆], and Jérôme Monnot[1,2]

[1] CNRS, FRE 3234, F-75775 Paris, France
[2] Université de Paris-Dauphine, LAMSADE, F-75775 Paris, France
[3] Fluminense Federal University, Inst. of Comp., Niterói, RJ, 24210-240, Brazil
[4] Fed. Center of Techn. Educ. Celso S. Fonseca, CEFET/RJ, 26041-271, Brazil
{laurent.gourves,monnot}@lamsade.dauphine.fr, alyra@ic.uff.br, mart@dcc.ic.uff.br

Abstract. We deal with different algorithmic questions regarding properly arc-colored s-t paths, trails and circuits in arc-colored digraphs. Given an arc-colored digraph D^c with $c \geq 2$ colors, we show that the problem of maximizing the number of arc disjoint properly arc-colored s-t trails can be solved in polynomial time. Surprisingly, we prove that the determination of one properly arc-colored s-t path is **NP**-complete even for planar digraphs containing no properly arc-colored circuits and $c = \Omega(n)$, where n denotes the number of vertices in D^c. If the digraph is an arc-colored tournament, we show that deciding whether it contains a properly arc-colored circuit passing through a given vertex x (resp., properly arc-colored Hamiltonian s-t path) is **NP**-complete, even if $c = 2$. As a consequence, we solve a weak version of an open problem posed in Gutin *et. al.* [17].

Keywords: Arc-colored digraphs, Properly arc-colored paths/trails and circuits, Hamiltonian directed path, arc-colored tournaments, Polynomial algorithms, **NP**-completeness.

1 Introduction, Notation and Terminology

In the last few years a great number of applications has been modelled as problems in edge-colored graphs [3,5]. For instance, problems in molecular biology correspond to extracting Hamiltonian or Eulerian paths or cycles colored in specified pattern [21,22,10], transportation and connectivity problems where reload costs are associated to pair of colors at adjacent edges [13,15], social sciences [9], VLSI optimization [19] among others. In this paper, we are specially concerned (from an algorithmic perspective) with different questions regarding properly arc-colored s-t paths, trails and circuits on arc-colored digraphs.

Given a (not necessarily edge-colored) graph $G = (V, E)$, a trail between s and t in G (called s-t *trail*) is a sequence $\rho = (v_0, e_0, v_1, e_1, \dots, e_k, v_{k+1})$ where $v_0 = s$,

⋆ Sponsored by FAPERJ and CNPq.

© Springer-Verlag Berlin Heidelberg 2010

$v_{k+1} = t$ and $e_i = v_i v_{i+1}$ for $i = 0, \ldots, k$ and $e_i \neq e_j$ for $i \neq j$. A path between s and t in G (called *s-t path*) is a trail $\rho = (v_0, e_0, v_1, e_1, \ldots, e_k, v_{k+1})$ between s and t where $v_i \neq v_j$ for $i \neq j$. To extend the definitions above for digraphs we just change edges $e_i = v_i v_{i+1}$ by arcs (or oriented edges) $e_i = \boldsymbol{v_i v_{i+1}}$. In this case, s-t trails (resp., s-t paths) are called *directed s-t trails* (resp., *directed s-t paths*).

Let $I_c = \{1, \ldots, c\}$ be a given set of colors ($c \geq 2$). In this work, D^c denotes a digraph whose arcs have a color in I_c, with no loops and parallel arcs linking the same pair of vertices. The vertex and arc sets of D^c are denoted by $V(D^c)$ and $A(D^c)$, respectively. For a given color i, $A^i(D^c)$ denotes the set of arcs of D^c colored by i. Given D^c and two vertices $u, v \in V(D^c)$, we denote by $\boldsymbol{uv}$ an arc of $A(D^c)$ and its color by $c(\boldsymbol{uv})$. In addition, we define $N^+_{D^c}(x) = \{y \in V(D^c) : \boldsymbol{xy} \in A(D^c)\}$ the *out-neighborhood* of x in D^c ($d^+_{D^c}(x) = |N^+_{D^c}(x)|$ is the *out-degree* of x in D^c), $N^-_{D^c}(x) = \{y \in V(D^c) : \boldsymbol{yx} \in A(D^c)\}$ the *in-neighborhood* of x in D^c ($d^-_{D^c}(x) = |N^-_{D^c}(x)|$ is the *in-degree* of x in D^c) and $N_{D^c}(x) = N^+_{D^c}(x) \cup N^-_{D^c}(x)$ the *neighborhood* of $x \in V(D^c)$. We say that, T^c defines an *arc-colored tournament* if it is obtained from a non-oriented complete edge-colored graph K^c by choosing an arbitrary direction for each colored edge of K^c.

From now on, we write PAC instead of *properly arc-colored.* A PAC *path* (resp., PAC *trail*) is a directed path (resp., trail) such that any two successive arcs have different colors. A PAC path or trail in D^c is *closed* if its end-vertices coincide and its first and last arcs differ in color. They are also refereed, respectively, as PAC *circuits* and *directed* PAC *closed trails.* The *length* of a directed trail, path, closed trail or circuit is the number of its arcs. Here, we only deal with PAC paths of length greater or equal than 2.

1.1 Some Related Work

Problems regarding properly edge-colored paths, trails and cycles (or PEC paths, trails and cycles, for short) in c-edge-colored (undirected) graphs have been widely studied from a graph theory and algorithmic point of views (see [3,1,24], the book [5] and the recent survey [18]). For instance, in [23], the author gives polynomial algorithms for several problems, including the determination of a PEC s-t path (if one exists). More recently, the authors in [1] introduced the notion of *trail-path* graph. Using this concept, they extend Szeider's Algorithm to deal with PEC s-t trails and they propose a polynomial algorithm for the determination of a PEC s-t trail. A polynomial time characterization of c-edge-colored graphs containing PEC cycles was first presented by Yeo [24] and generalized in [1] for PEC closed trails.

When dealing with PEC paths or trails with additional constraints in c-edge-colored graphs, the results are less optimistic. For example, it is well known that deciding whether a general 2-edge-colored graph (colored in *blue* and *red*) contains a PEC Hamiltonian cycle, a PEC Hamiltonian s-t path, or a PEC cycle passing through a prescribed pair of vertices are **NP**-complete problems [5]. Basically, the idea is to start from the proof of **NP**-completeness of these problems in uncolored digraphs and to use Häggkvist's transformation which consists in replacing each arc $e = \boldsymbol{xy}$ by an undirected path of length 2, xv_e and $v_e y$

(where v_e is a new vertex) with colors *blue* and *red*, respectively. Moreover, it is proved in [9] that deciding whether a 2-edge-colored graph contains a PEC s-t path passing through a vertex z is **NP**-complete. On the other side, many problems of this kind become polynomial in 2-edge-colored complete graphs. For instance, in [9] the authors proved that finding a PEC s-t path passing through a vertex z (if any) can be solved in polynomial time. The authors of [11] produced a nice characterization of c-edge-colored complete graphs which admit a PEC Hamiltonian path (with a non specified *source* and *destination*), and then they deduce a new polynomial algorithm for finding it (if one exists). In [16], the authors show that generalizations of these last 2 problems are polynomial if we are restricted to c-edge-colored graphs with no PEC closed trails. Finally, in [20] a characterization of c-edge-colored multigraphs which contain a PEC Eulerian trail is given. A $O(n^2 logn)$ algorithm for finding a PEC Eulerian trail in c-edge-colored multigraphs (if one exists) is described in [7].

In [15], the authors consider edge-colored s-t paths, trails and walks with minimum reload costs. In this case, we are given a c-edge-colored graph and a $c \times c$ matrix $R = [r_{i,j}]$ (for $i, j \in I_c$) whose entries define the *reload cost* when changing color i for color j. Given a trail (path) $\rho = (v_1, e_1, v_2, e_2, \ldots, e_k, v_{k+1})$ between vertices s and t, we define the reload cost of ρ as $r(\rho) = \sum_{j=1}^{k-1} r_{c(e_j),c(e_{j+1})}$. In [2], the authors deal with the determination of minimum directed s-t trails with reload costs in edge-colored digraphs whose total cost is given by the sum of reload costs (between successive colors in a trail) and positive costs associated to each arc. As discussed in [15], reload s-t trails (or paths) in edge-colored graphs may be converted into PEC trails (or paths) by conveniently choosing reload costs between each pair of colors (for instance, by setting $r_{i,i} = 1$ and $r_{i,j} = r_{j,i} = 0, \forall i, j \in I_c, i \neq j$. In this case, we seek a s-t trail (or s-t path) with reload cost 0).

Finally, if we deal with c-arc-colored digraphs, the existing results concerning the complexity of finding PAC s-t paths and circuits are rather rare and less optimistic. To our best knowledge, there is only one result which says that deciding whether a PAC circuit exists in 2-arc-colored digraphs, is **NP**-complete [17].

1.2 Contributions

In Section 2, we deal with the problem of finding PAC s-t trails in c-arc-colored digraphs D^c with $c \geq 2$. We show that the problem of maximizing the number of arc disjoint PAC s-t trails can be solved in polynomial time. As a consequence, we prove that is polynomial to decide whether D^c contains a directed PAC closed trail. In Section 3, we restrict our attention to path problems over c-arc-colored digraphs with no PAC circuits. We show that the determination of one PAC s-t path is **NP**-complete even if D^c is a planar c-arc-colored digraph containing no PAC circuits and $c = \Omega(|V(D^c)|)$. Finally, in Section 4 we focus on c-arc-colored tournaments. We prove that deciding whether a c-arc-colored tournament T^c (for $c = \Omega(|V(T^c)|^2)$) contains a PAC circuit passing through a given vertex x is **NP**-complete. This solves a weak version of an open problem initially posed by Gutin, Sudako and Yeo [17], whose objective is to determine whether T^c (for $c = 2$)

contains a PAC circuit. In addition, we prove that deciding whether T^c has a PAC s-t path or a PAC Hamiltonian s-t path is **NP**-complete. Notice that there is no evident link between deciding whether a c-arc-colored tournament possesses a PAC s-t path and a PAC Hamiltonian s-t path, although finding a PAC s-t path seems to be an easier problem than finding a PAC Hamiltonian s-t path.

2 PAC Trails and Closed Trails in Arc-Colored Digraphs

Here, we are interested in the complexity of finding PAC s-t trails and closed trails in arc-colored digraphs. These problems turn out to be polynomial using minimum cost flow computation. The details of proofs are omitted due to space limitation.

Theorem 1. *Given an arbitrary c-arc-colored digraph D^c, finding a* PAC *s-t trail in D^c (if any) can be done within polynomial time.*

As a consequence, we can prove the following results:

Corollary 1. *Let D^c be a c-arc-colored digraph with $c \geq 2$. The problem of finding a directed* PAC *closed trail in D^c (if any) can be solved in polynomial time.*

Corollary 2. *The problem of maximizing the number of arc disjoint* PAC *s-t trails in D^c can be solved in polynomial time.*

3 PAC Paths in Arc-Colored Digraphs with No PAC Circuits

Finding PEC paths, PEC trails or PEC cycles in edge-colored graphs is polynomial [23,1]. However finding PAC paths or PAC circuits in arc-colored digraphs seems harder. For example, the authors of [17] proved that deciding whether a 2-arc-colored digraph contains a PAC circuit is **NP**-complete. However, the PAC s-t path problem is polynomial in the following simple case:

Theorem 2. *If D^c is a c-arc-colored digraph containing no circuits at all (*PAC *or not) and s, t are two vertices of D^c then deciding the existence of a* PAC *path from s to t is polynomial time solvable.*

Unfortunately, this result does not hold in 2-arc-colored digraphs with no PAC circuits (note that non PAC circuits are allowed in this case).

Theorem 3. *Deciding whether a 2-arc-colored digraph with no* PAC *circuits contains a* PAC *path from s to t is **NP**-complete.*

Proof. We use a reduction from the Path with Forbidden Pairs Problem (PFPP, in short). In PFPP, we are given a (non-colored) digraph $D = (V, A)$, two vertices $v, w \in V$ and a collection $C = \{\{a_1, b_1\}, \ldots, \{a_q, b_q\}\}$ of pairs of vertices ($a_i \neq b_i$) from $V \setminus \{v, w\}$. The objective is to determine whether there exists a directed path connecting v to w and passing through at most one vertex from each pair. This problem was shown **NP**-complete [12] even if D is acyclic and all pairs of C are required to be disjoint, i.e., $\{a_i, b_i\} \cap \{a_j, b_j\} = \emptyset$ for $i \neq j$ (see problem [GT54] page 203 in [14]).

Let $D = (V, A)$ be an acyclic digraph containing $v, w \in V$ and a subset C of disjoint pairs of vertices. W.l.o.g., assume that $d^-_D(v) = d^+_D(w) = 0$. The construction of D^c is done in two steps. We build a digraph D' at first and then we build D^c from D'. The digraph $D' = (V', A')$ is such that $V' = V \cup \{s\}$, $A' = A \cup A'_1 \cup A'_2$ and vertices v and w are replaced by u and t respectively. Let $A'_1 := \{\boldsymbol{sa_1}, \boldsymbol{sb_1}, \boldsymbol{a_q u}, \boldsymbol{b_q u}\}$ and $A'_2 := \{\boldsymbol{a_i a_{i+1}}, \boldsymbol{a_i b_{i+1}}, \boldsymbol{b_i a_{i+1}}, \boldsymbol{b_i b_{i+1}} : i = 1, \ldots, q-1\}$. For the moment, notice that two arcs connecting the same pair of vertices may exist.

We build D^c as follows: for arcs in A'_1, $\boldsymbol{sa_1}$ and $\boldsymbol{sb_1}$ are colored in *blue* (color 2), while arcs $\boldsymbol{a_q u}$ and $\boldsymbol{b_q u}$ are colored in *red* (color 1). Next, we apply a directed version of Häggkvist's transformation (see Subsection 1.1): each arc $e = \boldsymbol{xy}$ of $A \cup A'_2$ is replaced by a directed path of length two, that is $\boldsymbol{xv_e}, \boldsymbol{v_e y}$, except for arcs incident to t. If $e = \boldsymbol{xy} \in A$, then $\boldsymbol{xv_e}$ is colored in *blue* and $\boldsymbol{v_e y}$ is colored in *red*. If $e = \boldsymbol{xt}$, then e is colored in *blue*. By extension, arcs $\boldsymbol{xv_e}, \boldsymbol{v_e y}$ are in A. If $e = \boldsymbol{xy} \in A'_2$ then $\boldsymbol{xv_e}$ is colored in *red* and $\boldsymbol{v_e y}$ is colored in *blue*. By extension, arcs $\boldsymbol{xv_e}, \boldsymbol{v_e y}$ are in this case in A'_2. The construction is completed (an example is given in Figure 1).

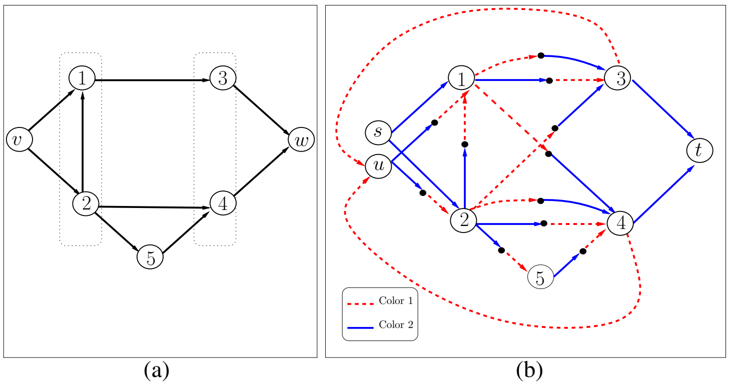


Fig. 1. Reduction from the PFPP with $C = \{\{1, 2\}, \{3, 4\}\}$ to the PAC s-t *path* problem. Color 1 (resp., 2) corresponds to *red* (resp., *blue*).

This construction is clearly done within polynomial time and D^c is a 2-arc-colored digraph. We now give an intermediate property.

Property 1. Any PAC path of D^c cannot use two consecutive arcs $\boldsymbol{xy}$ and $\boldsymbol{yz}$ such that $\boldsymbol{xy} \in A$ (resp., $\boldsymbol{xy} \in A'_1 \cup A'_2$) and $\boldsymbol{yz} \in A'_1 \cup A'_2$ (resp., $\boldsymbol{xy} \in A$) except if $y = u$.

Proof. By inspection. If $\boldsymbol{xy} \in A$ (resp., $\boldsymbol{xy} \in A'_2$) and $\boldsymbol{yz} \in A'_2$ (resp., $\boldsymbol{yz} \in A$) then $\boldsymbol{xy} = \boldsymbol{v_{e_1}y}$ is *red* (resp., *blue*) and $\boldsymbol{yz} = \boldsymbol{yv_{e_2}}$ is *red* (resp., *blue*). If $y \neq u$, then either $\boldsymbol{xy} = \boldsymbol{sa_1}$ (resp., $\boldsymbol{xy} = \boldsymbol{sb_1}$) (recall that $d^-_D(s) = 0$) and $\boldsymbol{yz} \in A$ or $\boldsymbol{xy} = \boldsymbol{v_{e_1}a_q}$ (resp., $\boldsymbol{xy} = \boldsymbol{v_{e_2}b_q}$) and $\boldsymbol{yz} = \boldsymbol{a_qu}$ (resp., $\boldsymbol{yz} = \boldsymbol{b_qu}$). In the first case, these two arcs $\boldsymbol{xy}$ and $\boldsymbol{yz}$ are *blue*, while in the second case, these two arcs $\boldsymbol{xy}$ and $\boldsymbol{yz}$ are *red.* ●

From Property 1, we deduce that any PAC path of D^c from s to t uses some arcs in $A'_1 \cup A'_2$ at first and after it uses some arcs in A (after passing through u).

Let us show that D^c does not contain any PAC circuit. Since (V', A) has no circuits (by hypothesis) and $(V', A'_1 \cup A'_2)$ has no circuits (by construction), any circuit of D^c must contain two consecutive arcs such that the first arc is in A (resp., $A'_1 \cup A'_2$) and the second arc is in $A'_1 \cup A'_2$ (resp., A). Using Property 1, the circuit is not PAC.

Finally, using Property 1, we claim that we have a directed path from v to w in D and visiting at most one vertex from each pair of C, if and only if, we have a PAC path from s to t in D^c. To see that, let Γ with $|\Gamma| \leq q$ be a subset of vertices of C belonging to a directed path from v to w in D. As a consequence, we can construct a directed PAC sub-path from u to t in D^c, say α, visiting the same set Γ of vertices and only containing arcs of A. Therefore, from Property 1, we can determine a PAC path from s to t in D^c by concatenating a PAC sub-path from s to u and containing no vertices of Γ (only with arcs of $A'_1 \cup A'_2$), which always exist in this case, with the PAC sub-path α from u to t. Conversely, consider a PAC path from s to t in D^c (note that all PAC s-t paths in D^c contain vertex u). Thus, by Property 1, it follows that the associated PAC sub-path from s to u only contains arcs of $A'_1 \cup A'_2$ and the PAC sub-path from u to t only contains arcs of A (each of them containing at most one vertex of $\{a_i, b_i\}$ for $i = 1, .., q$). After deleting all arcs of $A'_1 \cup A'_2$ in D^c, the resulting path from u to t will be directely associated to a path from v to w in D passing through at most one vertex from each pair of C. □

Now, we show that our previous theorem can be extended to include any number of colors. Formally, we have the following result:

Corollary 3. *Deciding whether a c-arc-colored digraph D^c with no* PAC *circuits contains a* PAC *s-t path is* ***NP****-complete, even if $c = \Omega(|V(D^c)|^2)$.*

Theorem 3 can also be extended to planar c-arc-colored digraphs.

Corollary 4. *Deciding whether a planar c-arc-colored digraph D^c_P with no* PAC *circuits contains a* PAC *s-t path is* ***NP****-complete even for $c = \Omega(|V(D^c_P)|)$.*

4 PAC Circuits, Paths and Hamiltonian Paths in c-Arc-Colored Tournaments

A tournament is a digraph which corresponds to a complete asymmetric binary relation. As indicated previously, one can build a tournament as follows: take a complete undirected graph and assign a direction to each edge. The problems of finding PAC s-t paths and PAC circuits in c-arc-colored tournaments are challenging. For example, the complexity of determining a PAC circuit in a 2-arc-colored tournament is posed in [17,5]. Here, we propose and solve a weaker version of this problem, we show that deciding whether a c-arc-colored tournament contains a PAC circuit passing through a given vertex x is **NP**-complete. As a consequence, we prove that finding PAC s-t paths in tournaments is also **NP**-complete.

We also deal with the determination of PAC Hamiltonian s-t paths in arc-colored tournaments. When restricted to uncolored tournaments, one of the earliest results is Rédei's theorem, which states that every tournament has an Hamiltonian directed path (the endpoints are not specified). More recently, in [6] the authors gave a polynomial algorithm to find an Hamiltonian directed s-t path (if one exists) in an uncolored tournament. Recently in [11,5] (using a nice characterization) the authors show that the problem of finding PEC Hamiltonian s-t path is polynomial in c-edge-colored complete graphs for $c \geq 3$, solving a conjecture posed in [4] (the case $c = 2$ was previously solved in [4]). Unfortunately, we prove that these results cannot be extended to the directed case.

Thus, we begin with the following result:

Theorem 4. *Deciding whether a c-arc-colored tournament contains a* PAC *circuit visiting a given vertex x is **NP**-complete even for $c = \Omega(|V(D^c)|)$.*

Proof. Here, we only deal with $c = 2$ since our proof can be easily extended for $c = \Omega(|V(D^c)|)$ (see the Appendix for details). Thus, we start from the 2-arc-colored digraph $D^c = (V', A')$ built in Theorem 3 and we complete it in order to get a tournament T^c. The idea is to get a tournament whose PAC circuits passing through $x = s$ (if one exists) also visit vertex t. Then, directed paths from v to w in D (visiting at most one vertex from each pair of C), instance of the Path with Forbidden Pairs Problem, correspond to PAC circuits passing through s in T^c and vice-versa.

Recall that in the construction of D^c (see the proof of Theorem 3), we replace each arc $e \in A$ (resp., e from A'_2), except those which are incident to t, by a directed path of length two in A (resp., in A'_2) where the added vertex is denoted by v_e. If $e \in A$ (resp. $e \in A'_2$) then we suppose that $v_e \in V(A)$ (resp., $v_e \in V(A'_2)$).

Now, we show how to build the tournament T^c. The construction is done in four steps:

(1) Build a set of arcs A'_3 as follows. Add a *red* arc $\boldsymbol{ts}$ and a *blue* arc $\boldsymbol{us}$. Do $A(D^c) \leftarrow A(D^c) \cup A'_3$. Then, add a *blue* arc $\boldsymbol{tx}$ for each $x \notin N_{D^c}(t)$, a *blue* arc $\boldsymbol{xu}$ for each $x \notin N_{D^c}(u)$ and a *blue* arc $\boldsymbol{xs}$ for each $x \notin N_{D^c}(s)$. Do $A(D^c) \leftarrow A(D^c) \cup A'_3$.

(2) Build a set of arcs A'_4 as follows. Choose an arbitrary vertex v_e of $V(A)$ (resp., $V(A'_2)$) with an incoming *blue* (resp., *red*) arc $\boldsymbol{yv_e}$ (resp., $\boldsymbol{a_iv_e}$ or $\boldsymbol{b_iv_e}$), and add a *blue* (resp., *red*) arc $\boldsymbol{v_ex}$ for every $x \notin N_{D^c}(v_e)$. Let A'_4 be this new set of arcs and do $A(D^c) \leftarrow A(D^c) \cup A'_4$. Repeat the process for

the remaining vertices v_e of $V(A)$ (resp., $V(A'_2)$) by following an arbitrary order.

(3) Build a set of *blue* arcs $A'_5 = \{\boldsymbol{a_q x} : \forall x \notin N_{D^c}(a_q)\} \cup \{\boldsymbol{b_q y} : \forall y \notin (N_{D^c}(b_q) \cup \{a_q\})\}$. Recall that $\{a_q, b_q\}$ is the last pair of C. Set $A(D^c) \leftarrow A(D^c) \cup A'_5$.

(4) Build a set A'_6 of *blue* arcs with endpoints in $V(D^c) \setminus (\{s, u, t, a_q, b_q\} \cup \{v_e : v_e \in V(A) \cup V(A'_2)\})$ and arbitrary directions. Set $A(D^c) \leftarrow A(D^c) \cup A'_6$.

The construction is completed. It is clearly done within polynomial time, and T^c is a 2-arc-colored tournament. We now give some useful properties:

Property 2. The following properties hold:

(i) Any PAC circuit passing through s (resp., u) in T^c uses $\boldsymbol{ts}$ and one arc among $\{\boldsymbol{sa_1}, \boldsymbol{sb_1}\}$ (resp., uses exactly one arc among $\{\boldsymbol{a_q u}, \boldsymbol{b_q u}\}$ and one arc $\boldsymbol{uv_e} \in A$).

(ii) No PAC circuit passing through s in T^c uses an arc of A'_4.

(iii) No PAC circuit passing through s in T^c uses an arc of $A'_5 \cup A'_6$.

Proof. For (i). Due to step (1) of the above procedure, there is a unique *red* arc incident to s (resp., t) which is $\boldsymbol{ts}$. Thus, any PAC circuit passing through s also visits t. Moreover, vertex s only has two outgoing arcs $\boldsymbol{xa_1}$ and $\boldsymbol{xb_1}$ which are colored in *blue*.

Concerning vertex u, $\boldsymbol{a_q u}$ and $\boldsymbol{b_q}$ are the only *red* arcs incident to u. Thus, if a PAC circuit visits u then it contains one of these two arcs as incoming arc and one arc $\boldsymbol{uv_e} \in A$ as outgoing arc. Actually, vertex u has only arcs $\boldsymbol{uv_e} \in A$ and $\boldsymbol{us}$ as outgoing arcs and no PAC circuit can use the *blue* arc $\boldsymbol{us}$ since all arcs going out of s are *blue*.

For (ii). By contradiction, assume that there is a PAC circuit passing through s, $\rho = (v_1, e_1, \ldots, e_k, v_{k+1})$ with $v_1 = v_{k+1} = s$ and containing some arcs of A'_4. Consider the first arc $e_p \in A'_4$ used by ρ (i.e., $e_q \notin A'_4$ for $q = 1, \ldots, p-1$). By construction $e_p = \boldsymbol{v_e x}$ and from (i), we deduce $k > p > 1$ (i.e., $x \notin \{s, t\}$). Since $e_{p-1} \notin A'_4$ and $e_{p-1} \notin A'_3$ from (i), arc $e_{p-1} = \boldsymbol{yv_e} \in A \cup A'_2$. Thus, e_{p-1} has the same color as e_p, which is a contradiction.

For (iii). By contradiction. Firstly assume that there is a PAC circuit passing through s, $\rho = (v_1, e_1, \ldots, e_k, v_{k+1})$ with $v_1 = v_{k+1} = s$ and containing some arcs of A'_5. Like previously, consider the first arc $e_p \in A'_5$ of ρ (i.e., $e_q \notin A'_5$ for $q = 1, \ldots, p-1$). W.l.o.g., suppose that $e_p = \boldsymbol{a_q x}$ (the same result holds for $e_p = \boldsymbol{b_q x}$); we get $x \neq u$ from (i). Then, $e_{p-1} = \boldsymbol{v_e a_q} \in A$ is colored in *red* and from (ii) we deduce that $e_{p-2} = \boldsymbol{yv_e} \in A$ and is colored in *blue*. Since all arcs in A'_6 are *blue* like e_{p-2}, by induction we deduce that $e_q \in A$ for $q = 1, \ldots, p-1$. We obtain a contradiction since from (i) $e_1 \in A'_1$ (i.e., $e_1 \in \{\boldsymbol{sa_1}, \boldsymbol{sb_1}\}$).

Now, suppose that a PAC circuit passing through s, $\rho = (v_1, e_1, \ldots, e_k, v_{k+1})$ with $v_1 = v_{k+1} = s$ contains some arcs in A'_6. Consider the last arc $e_p = \boldsymbol{xy} \in A'_6$ used by ρ (i.e., $e_q \notin A'_6$ for $q = p+1, \ldots, k+1$). Since e_p is colored in *blue* and $y \neq t$ (from (i)), we deduce that e_{p+1} is colored in *red*. Then, we get $y = a_i$ or $y = b_i$ and $e_{p+1} = \boldsymbol{yv_e} \in A'_2$ since $e_{p+1} \notin A'_6$. Moreover, from (ii), $e_{p+2} = \boldsymbol{v_e z} \in A'_2$ is colored in *blue*. Now, since $e_k \in A$ (the arc of ρ incoming in vertex t) is also colored in *blue*, the PAC subpath of ρ from x to $t = v_k$ must contain arc $\boldsymbol{a_q u}$ or $\boldsymbol{b_q u}$ (using Property 1 of Theorem 3, it is the only way to flip arcs of A'_2 to arcs of A). Thus, this PAC circuit ρ can be decomposed into three PAC paths: ρ_1 from y to u, ρ_2 from u to s (and containing arc $e_{k+1} = \boldsymbol{ts}$) and ρ_3 from s to y. In particular, the PAC path ρ_3 begins with a *blue* arc (by (i)), only uses arcs in A'_2 and ends by a *blue* arc, which is impossible since ρ_3 does not contain u. Actually, path ρ_3 cannot use some arcs of A'_6. We have $e_2 = \boldsymbol{x_1 v_e} \in A'_2$ with $x_1 \in \{a_1, b_1\}$ (since the arc must be colored in *red*) and using (ii), arc $e_3 = \boldsymbol{v_e x_2}$ with $x_2 \in \{a_2, b_2\}$ is colored in *blue*. Thus, $e_4 \notin A'_5 \cup A'_6$. Then, the result follows by induction. Notice that it may exist a PAC circuit containing one arc $e = \boldsymbol{xy} \in A'_6$ (but not passing through s). In this case, this PAC circuit is composed of two PAC paths ρ_1 from y to u and ρ_2 from u to y: ρ_1 only uses arcs of A'_2 from y to a_q (or b_q) and uses arc $\boldsymbol{a_q u} \in A'_1$ (or $\boldsymbol{b_q u} \in A'_1$) while ρ_2 only uses arcs of A from u to x and uses arc $e = \boldsymbol{xy} \in A'_6$. •

Using Properties 1 and 2, we can easily see that we have a directed path from u to w in D and visiting at most one vertex from each pair of C, if and only if, we have a PAC circuit passing through s in T^c. □

Corollary 5. *Deciding whether a c-arc-colored tournament T^c contains a* PAC *s-t path is **NP**-complete even for $c = \Omega(|V(T^c)|^2)$.*

Proof. In the proof of Theorem 4, we have a PAC circuit passing through s if and only if we have a PAC s-t path in T^c. □

We finish the paper by considering the PAC Hamiltonian s-t path problem.

Theorem 5. *Deciding whether a 2-arc-colored tournament T^c contains a* PAC *Hamiltonian s-t path is **NP**-complete.*

Proof. We use a reduction from the directed Hamiltonian s'-t' path problem in general uncolored digraphs (DHPP in short). Given a digraph $D = (V, A)$ and two vertices s', t', DHPP asks whether a directed Hamiltonian s'-t' path exists. DHPP is **NP**-complete (see problem [GT39] page 199 in [14]).

Let $D = (V, A)$ be a digraph where $V = \{v^1, \ldots, v^n\}$ and $v^1 = s'$, $v^n = t'$, instance of DHPP. W.l.o.g., assume that $d^-_D(v^1) = d^+_D(v^n) = 0$. The construction of the 2-arc-colored tournament T^c is done in two steps: we first build a 2-arc-colored digraph D^c and then we complete D^c into T^c.

The 2-arc-colored digraph $D^c = (V', A')$ is built in the following way: $V' = \{v^i_{in}, v^i_{out} : i = 1, \ldots, n\}$ and $A' = A'_1 \cup A'_2$ where $A'_1 = \{\boldsymbol{v^i_{out} v^j_{in}} : \boldsymbol{v^i v^j} \in A\}$ and $A'_2 = \{\boldsymbol{v^i_{in} v^i_{out}} : i = 1, \ldots, n\}$. Arcs in A'_1 are colored in *red* while arcs in A'_2 are colored in *blue*. See Figure 2 for an illustration of D^c.

Next we build the tournament T^c from D^c as follows. For every missing arc in D^c, we apply the following procedure where $1 \le i < j \le n$ is assumed. If the endpoints of the missing arc are v^i_{in} and v^j_{in} (resp., v^i_{in} and v^j_{out}), add a *blue* arc $\boldsymbol{v^j_{in} v^i_{in}}$ (resp., $\boldsymbol{v^j_{out} v^i_{in}}$). If the endpoints of the missing arc are v^i_{out} and v^j_{in} (resp., v^i_{out} and v^j_{out}), add a *red* arc $\boldsymbol{v^j_{in} v^i_{out}}$ (resp., $\boldsymbol{v^j_{out} v^i_{out}}$). These new *blue* (resp., *red*) arcs define a set denoted by A''_2 (resp., A''_1).

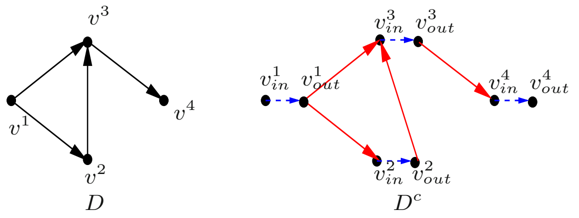


Fig. 2. A digraph D and the 2-arc-colored digraph D^c. Dotted arcs are colored in *blue* and rigid arcs are colored in *red*.

The construction is completed (see Figure 3 for an illustration). It is clearly done within polynomial time. The resulting tournament is 2-arc-colored. Its *blue* arcs belong to $A'_2 \cup A''_2$ while its *red* arcs belong to $A'_1 \cup A''_1$. Let us give an intermediate property.

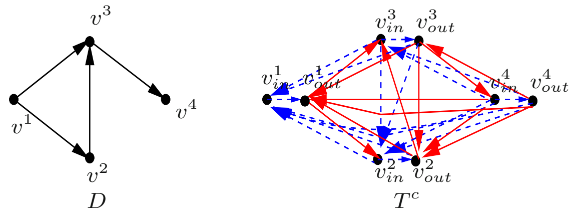


Fig. 3. A digraph D and the 2-arc-colored tournament T^c. Dotted arcs are colored in *blue* and rigid arcs are colored in *red*.

Property 3. No PAC path from v^1_{in} to v^n_{out} in T^c can use an arc of $A''_1 \cup A''_2$.

Proof. By contradiction suppose that a PAC path $\rho = (v_0, e_0, v_1, e_1, \ldots, e_k, v_{k+1})$ linking $v_0 = v^1_{in}$ to $v_{k+1} = v^n_{out}$ uses some arcs of $A''_1 \cup A''_2$. Consider the last arc $e_p \in A''_1 \cup A''_2$ used by ρ (that is $e_q \notin A''_1 \cup A''_2$ for $q = p+1, \ldots, k+1$). If $e_p = \boldsymbol{v^j_{in} v^i_{in}}$ or $e_p = \boldsymbol{v^j_{out} v^i_{in}}$ $(i < j)$ then it belongs to A''_2 and it is *blue*. We have $v^i_{in} \neq v^n_{out}$ so the path must contain an arc going out of v^i_{in} which does not belong to $A''_1 \cup A''_2$. This arc $e_{p+1} = \boldsymbol{v^i_{in} v^i_{out}}$ is *blue*, contradiction. Otherwise, $e_p = \boldsymbol{v^j_{in} v^i_{out}}$ $(i \neq j)$ or $e_p = \boldsymbol{v^j_{out} v^i_{out}}$. Therefore $e_p \in A''_1$ and it is *red*. We have $v^i_{out} \neq v^n_{out}$ since $\boldsymbol{v^n_{in} v^n_{out}}$ is the unique arc coming into v^n_{out}. Then, the path must contain an arc $e_{p+1} \notin A''_1 \cup A''_2$ going out of v^i_{out} but all arcs of $A'_1 \cup A'_2$ going out of v^i_{out} are *red* since they belong to A'_1, contradiction. □

We deduce from Property 3 that any PAC path from v^1_{in} to v^n_{out} in T^c only uses arcs of $A'_1 \cup A'_2$. Thus, D admits a directed Hamiltonian path from $s' = v^1$ to $v^n = t'$, if and only if, T^c has a PAC Hamiltonian path from $s = v^1_{in}$ to $t = v^n_{out}$. □

References

1. Abouelaoualim, A., Das, K.C., Faria, L., Manoussakis, Y., Martinhon, C.A., Saad, R.: Paths and trails in edge-colored graphs. Theoretical Computer Science 409(3), 497–510 (2008)
2. Amaldi, E., Galbiati, G., Maffioli, F.: On the minimum reload cost paths, tours and flows. In: Proc. CTW 2008, pp. 112–115 (2008)
3. Bang-Jensen, J., Gutin, G.: Alternating cycles and paths in edge-coloured multigraphs: a survey. Discrete Mathematics 165/166, 39–60 (1997)
4. Bang-Jensen, J., Gutin, G.: Alternating cycles and trails in 2-edge-coloured complete multigraphs. Discrete Mathematics 188, 61–72 (1998)
5. Bang-Jensen, J., Gutin, G.: Digraphs: Theory, Algorithms and Applications, 1st edn. Springer Monographs in Mathematics. Springer, London (2006)
6. Bang-Jensen, J., Manoussakis, Y., Thomassen, C.: A Polynomial Algorithm for Hamiltonian-Connectedness in Semicomplete Digraphs. J. Algorithms 13(1), 114–127 (1992)
7. Benkouar, A., Manoussakis, Y., Paschos, V.T., Saad, R.: On the complexity of Hamiltonian and Eulerian problems in edge-colored complete graphs. RAIRO Journal On Operations Research 30(4), 417–438 (1996)
8. Bondy, J.A., Murty, U.S.R.: Graph Theory (Graduate Texts in Mathematics). Springer, Heidelberg (2008)
9. Chou, W.S., Manoussakis, Y., Megalakaki, O., Spyratos, M., Tuza, Z.: Paths through fixed vertices in edge-colored graphs. Mathématiques et Sciences Humaines 127, 49–58 (1994)
10. Dorniger, D.: Hamiltonian circuits determining the order of chromossomes. Discrete Applied Mathematics 50, 159–168 (1994)
11. Feng, J., Giesen, H.-E., Guo, Y., Gutin, G., Jensen, T., Rafiey, A.: Characterization of edge-colored complete graphs with properly colored Hamilton paths. J. Graph Theory 53, 333–346 (2006)
12. Gabow, H., Maheshwari, S., Osterweil, L.: On two Problems in the Generation of Program Test Paths. IEEE Trans. on Soft. Eng., SE-2(3), 227–231 (1976)
13. Gamvros, I.: Satellite network design, optimization and management. PhD thesis, University of Maryland (2006)
14. 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)
15. Gourvès, L., Lyra, A., Martinhon, C., Monnot, J.: The minimum reload s-t path/trail/walk problems. In: Nielsen, M., Kucera, A., Miltersen, P.B., Palamidessi, C., Tuma, P., Valencia, F.D. (eds.) SOFSEM 2009. LNCS, vol. 5404, pp. 621–632. Springer, Heidelberg (2009)
16. Gourvès, L., Lyra, A., Martinhon, C., Monnot, J., Protti, F.: On s-t paths and trails in edge-colored graphs. In: Proc. LAGOS 2009. Electronic Notes in Discrete Mathematics, vol. 35, pp. 221–226 (2009)
17. Gutin, G., Sudakov, B., Yeo, A.: Note on alternating directed cycles. Discrete Mathematics 191, 101–107 (1998)
18. Gutin, G., Jung Kim, E.: Properly Coloured Cycles and Paths: Results and Open Problems. In: Lipshteyn, M., Levit, V.E., McConnell, R.M. (eds.) Graph Theory, Computational Intelligence and Thought. LNCS, vol. 5420, pp. 200–208. Springer, Heidelberg (2009)
19. Hu, T.C., Kuo, Y.S.: Graph folding and programmable logical arrays. Networks 17, 19–37 (1987)

20. Kotizig, A.: Moves without forbidden transitions in a graph. Math. Fyz. Cazopis 18, 76–80 (1968)
21. Pevzner, P.: DNA physical mapping and properly edge-colored eurelian cycles in colored graphs. Algorithmica 13, 77–105 (1995)
22. Pevzner, P.: Computational Molecular Biology: An Algorithmic Approach. The MIT Press, Cambridge (2000)
23. Szeider, S.: Finding paths in graphs avoiding forbidden transitions. Discrete Applied Mathematics 126, 261–273 (2003)
24. Yeo, A.: A note on Alternating Cycles in Edge-coloured Graphs. Journal of Combinatorial Theory, Series B-69, 222–225 (1997)

The Max k-Cut Game and Its Strong Equilibria[⋆]

Laurent Gourvès and Jérôme Monnot

LAMSADE, CNRS FRE 3234 & Université de Paris-Dauphine
{laurent.gourves,monnot}@lamsade.dauphine.fr

Abstract. An instance of the MAX $k-$CUT game is an edge weighted graph. Every vertex is controlled by an autonomous agent with strategy space $[1..k]$. Given a player i, his payoff is defined as the total weight of the edges $[i, j]$ such that player j's strategy is different from player i's strategy. The social welfare is defined as the weight of the cut, i.e. half the sum of the players payoff. It is known that this game always has a pure strategy Nash equilibrium, a state from which no single player can deviate. Instead we focus on strong equilibria, a robust refinement of the pure Nash equilibrium which is resilient to deviations by coalitions of any size. We study the strong equilibria of the MAX $k-$CUT game under two perspectives: existence and worst case social welfare compared to a social optimum.

1 Introduction

Given a graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}_+$, the MAX $k-$CUT problem is to partition V into k sets V_1, V_2 $\ldots V_k$ such that the sum of the weight of the edges having their endpoints not in the same part of the partition is maximum. In this paper we study a strategic game defined upon MAX $k-$CUT. Each vertex is controlled by a player with strategy set $\{1, 2, \ldots, k\}$. A player's utility is the total weight of the edges incident to her and such that her neighbor has a different strategy.

The game models a large class of situations where there are k available facilities and every agent must choose one. The facilities are inherently similar but their number is typically smaller than the number of agents (e.g. compartments in a train). Then the agents must share the facilities. In this game every agent is "hindered" by the other agents but solely by those who chose the same facility. So every agent makes his choice according to the agents that he wants to avoid. In the MAX $k-$CUT game, the weight of an edge $[i, j]$ represents the strength of interference that agents i and j exert on each others if they choose the same facility. The social welfare for a given state is defined as the total weight of the edges with corresponding endpoint agents making distinct choices of a facility (i.e. half the sum of the player's utility).

This paper is devoted to the existence and the quality of pure[1] equilibria in the MAX $k-$CUT game. Our work is motivated by the study of large scale distributed

[⋆] This work is supported by French National Agency (ANR), project COCA ANR-09-JCJC-0066-01.

[1] We only consider pure strategies so we often omit the word pure.

© Springer-Verlag Berlin Heidelberg 2010

systems which usually lack a central control authority. Instead these systems are operated by self interested entities. Though the uncoordinated decisions made by the entities often end up in a stable configuration (an equilibrium), these configurations are rarely socially optimal. Two main questions naturally arise in this context: For which instances an equilibrium exists? How far from the social optimum these equilibria can be?

When the focus is on pure Nash equilibria in the MAX $k-$CUT game, the answer to these two questions is known. For every instance (and every k) an optimal cut is a pure Nash equilibrium. Furthermore, the *price of anarchy* (PoA in short) [1], defined as the worst case ratio between the social welfare of a Nash equilibrium and the optimal social welfare, is $\frac{k-1}{k}$ [2]. This paper is devoted to the (even more) appealing concept of *strong equilibrium* (SE in short) [3]. This notion refines the NE because it considers deviations by coalitions of any size whereas NE are restricted to deviations by a single player. When it exists, a SE is a very robust state of the game, it is also more sustainable than a NE. Strong equilibria are the topic of many recent articles including [4,5,6,7,8].

We are interested in the existence of SE in the MAX $k-$CUT game and their quality with respect to socially optimal configurations. In particular, we resort to the *strong price of anarchy* (SPoA in short) [4] which is the price of anarchy restricted to strong equilibria.

Previous related work and Contribution. The MAX $k-$CUT game or similar games like the party affiliation game, the interference game or the consensus game have been studied in [9,10,5,6,2] from different perspectives: existence of a pure equilibrium, convergence time to an equilibrium, complexity for computing an equilibrium and worst case quality of an equilibrium. In this paper we only deal with the existence and the worst case quality of a pure equilibrium.

For the MAX $2-$CUT game, the picture is complete. A SE always exists because the state corresponding to an optimal cut is a SE. The PoA is $1/2$ by a well known result from local search theory and the SPoA is $2/3$ [5]. From now on we consider that $k \geq 3$. A NE always exists because the state corresponding to an optimal cut is a NE. In [2] it is shown that the PoA of the *unweighted* MAX $k-$CUT game is $\frac{k-1}{k}$ and one can easily extend the result to the weighted case. In [5] it is shown that an optimal cut is not necessarily a SE but the instance presented admits another optimal cut which turns out to be a SE. The state corresponding to an optimal cut is a 3-strong equilibrium (a state immune to deviations by coalitions of at most 3 players) but not necessarily a 4-strong equilibrium [5].

In Section 2 we provide some useful definitions and notations. The results presented in this paper deal with the existence of a SE (Section 3) and if a SE exists, we bound its quality compared to a social optimum (Section 4). In Section 3 we do not prove or disprove that every instance of the MAX $k-$CUT game admits a SE. Instead we give both negative and positive results related to this question. In Section 4 we give an upper bound of $\frac{2k-2}{2k-1}$ and a matching lower bound on the SPoA. It is noteworthy that the upper bound is derived without any assumption on the instance so it applies every time a SE exists. We conclude in Section 5. We conjecture that a SE always exists for the MAX $k-$CUT game, but we are not able to prove this for the moment.

2 Definitions and Notations

A *strategic game* is a tuple $\langle N, (\Sigma_i)_{i\in N}, (u_i)_{i\in N}\rangle$ where N is the set of players (we suppose that $|N| = n$), Σ_i is the set of strategies of player i and $u_i : \times_i \Sigma_i \to \mathbb{R}$ is player i's utility function. A *pure state* or *pure strategy profile* of the game is an element of $\Sigma := \times_i \Sigma_i$. Although players may choose a probability distribution over their strategy set, we only consider pure strategy profiles in this paper. Players are supposed to be rational, i.e. each of them plays in order to *maximize* his utility.

Given a state $a \in \Sigma$, (a_{-i}, b_i) denotes the state where a_i is replaced by b_i in a while the strategy of the other players remains unchanged. A state a is a *Nash equilibrium* (NE) if there no player $i \in N$ and a strategy $b_i \in A_i$ such that $u_i\big((a_{-i}, b_i)\big) > u_i(a)$.

Given two states a, a' and a coalition $C \subseteq N$, (a_{-C}, a') denotes the state where a_i is replaced by a'_i in a for all $i \in C$. A state a is a *strong equilibrium* (SE) if there is no non-empty coalition $C \subseteq N$ and a profile $a' \in A$ such that $u_i\big((a_{-C}, a')\big) > u_i(a)$ for all $i \in C$. A state a is an *r-strong equilibrium* (r-SE) if there is no non-empty coalition $C \subseteq N$ of size at most r and a profile $a' \in A$ such that $u_i\big((a_{-C}, a')\big) > u_i(a)$ for all $i \in C$. Therefore a SE is a NE, a NE is a 1-SE and a n-SE is SE (n is the number of players).

The *price of anarchy* (PoA) measures the performance of decentralized systems [1] via its Nash equilibria. More formally, let Γ be a family of strategic games, let γ be an instance of Γ, let A_γ be the strategy space of γ, let $\mathcal{Q} : A_\gamma \to \mathbb{R}_+$ be the social welfare, let $\mathcal{E}(\gamma)$ be the set of all pure Nash equilibria of γ and let o_γ be a social optimum for γ (i.e. $o_\gamma = \text{argmax}_{a\in A_\gamma} \mathcal{Q}(a)$). The *pure* price of anarchy of Γ is $\min_{\gamma\in\Gamma} \min_{a\in\mathcal{E}(\gamma)} \mathcal{Q}(a)/\mathcal{Q}(o_\gamma)$. If $\mathcal{SE}(\gamma)$ denotes the set of all strong equilibria of γ then the strong price of anarchy (SPoA) [4] is $\min_{\gamma\in\Gamma} \min_{a\in\mathcal{SE}(\gamma)} \mathcal{Q}(a)/\mathcal{Q}(o_\gamma)$.

3 On the Existence of Strong Equilibria

This section contains both negative and positive results on the existence of a SE in the MAX $k-$CUT game. The negative results are (often non trivial) observations that all proof techniques that we are aware of, to show the existence of a SE, fail. The positive results are (often tight) sufficient conditions for the existence of a SE, and the existence of a good approximation of it in every instance.

Negative results. The strategy profiles which correspond to optimal cuts play an important role because they are often stable states. When $k = 2$ and $k \geq 3$, optimal cuts are respectively strong equilibria and 3-strong equilibria [5]. An instance presented in [5] admits two optimal cuts: one is a SE while the other is not a 4-SE. It shows that an optimal cut is not necessarily a SE but it does not prevent (at least) one optimal cut to be a SE. In this paper we propose a new and simpler instance in which the unique optimal cut is not a SE. Consider the instance given on the left part of Figure 1. An exhaustive search can show that the given 3-cut is the only optimal solution. However, it is not a SE as nodes a, b, c and f can modify their strategy and benefit (see the right part of Figure 1).

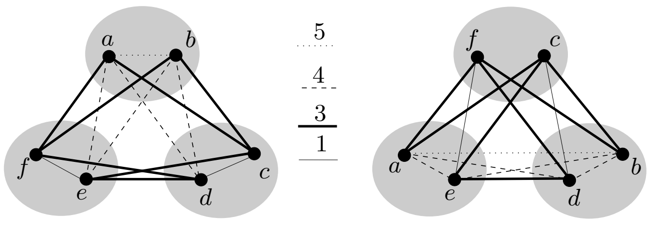


Fig. 1. Left: An optimal 3-cut with value 37. Right: Starting from the left configuration, vertices a, b, c and f can move and benefit but the value of the cut is 36.

A second way to prove the existence of a SE is to exhibit a strong potential function Φ_S and an order $\prec$ such that $\Phi_S(\sigma) \prec \Phi_S(\sigma')$ holds for every improving pair of strategy profiles (σ, σ')[2] [6]. This technique captures the fact that the players naturally converge to a SE (a state σ^* such that $\Phi_S(\sigma^*)$ is locally maximum for $\prec$) since every sequence of improvements is finite. However no pair $(\Phi_S, \prec)$ can exist for the MAX $k-$CUT game since the dynamics can cycle. We are given an instance of the MAX 3$-$CUT with 4 nodes and three strategy profiles (see Figure 2). At each deviation by a coalition, the utility of every member strictly increases but the three configurations form a cycle. It is noteworthy that the *interference game* studied by Harks, Klimm and Möhring [6], and for which they prove the existence of a SE by the strong potential function, is slightly different to the MAX k-CUT game. The slight difference makes both results (existence of a strong potential function for the interference game, and non existence of a strong potential function for the MAX $k-$CUT game) consistent.

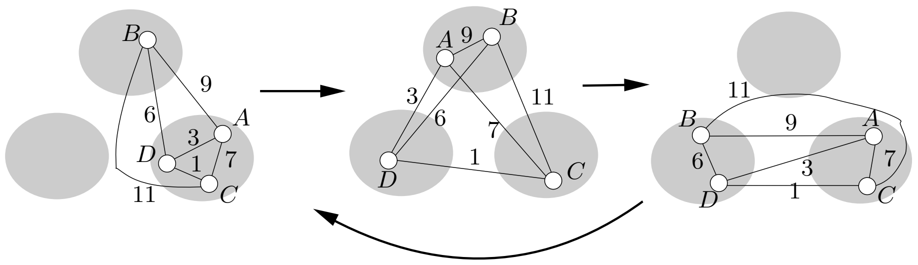


Fig. 2. A cycle disproving the existence of a strong potential function

[2] The cost (resp. the payoff) in σ of every player having a distinct strategy in σ' strictly decreases (resp. strictly increases) when switching to σ'.

A last attempt is to observe that the MAX $k-$CUT game is a *congestion game* [11,12]. Congestion games are extensively studied for two reasons, they always admit a pure strategy Nash equilibrium and they are general models for resource sharing in networks. Sufficient conditions on the strategy space to show the existence of a SE were derived [7,8]. These works are based on the notion of *bad* configurations in the strategy space. An instance of a congestion game without any bad configuration admits a SE (but a bad configuration does not prevent some instances to admit a SE). If we turn the MAX $k-$CUT game into a congestion game and consider the clique on 3 nodes then we get a bad configuration.

Positive results. A direct observation is that every k-colorable graph admits a SE: a k coloration induces a state σ^*, where every vertex with color c plays c, such that $u_i(\sigma^*) = \sum_{[i,j]\in E} w([i,j])$ for all $i \in V$. However this condition is not necessary: take a clique of size 3 for the MAX 2-CUT game. In every state σ there is a node i which satisfies $u_i(\sigma) < \sum_{[i,j]\in E} w([i,j])$ but this instance admits a SE.

Another direction to prove the existence of a SE is to limit the number players.

Proposition 1. *If* $|V| \leq k+2$ *then an optimal state of the* MAX k-CUT *game is a SE.*

Proof. Let σ be an optimal state. Hence σ is a NE. Suppose there is $j \in \{1,\dots,k\}$ and no player i such that $\sigma(i) = j$. For every pair of nodes i, i' such that $\sigma(i) = \sigma(i')$, it must be $w([i,i']) = 0$ since otherwise σ is not optimal. Thus G is $k-1$ colorable and σ must be a SE.

Now suppose that for every $j \in \{1,\dots,k\}$, there is at least one player i such that $\sigma(i) = j$. By contradiction, suppose that there is a coalition $C \subseteq V$ of players who can modify their strategy and benefit. Let σ' be the resulting strategy profile. A result of [5] states that an optimal state is a 3-SE, i.e. $|C| > 3$.

Let $V_1,\dots,V_k$ (resp. $V'_1,\dots,V'_k$) be the k partition corresponding to σ (resp. σ'). By hypothesis $|V_j| \geq 1$ for $j = 1..k$. Since $|V| \leq k+2$, we can consider two cases:

- $|V_1| = 3$ and $|V_j| = 1$ for $j = 2,\dots,k$. For every $i \in V_2 \cup \dots \cup V_k$, we know that $i \notin C$ because $u_i(\sigma)$ is maximum. Then $C \subset V_1$, $|C| \leq |V_1| = 3$, contradiction with $|C| > 3$.
- $|V_1| = |V_2| = 2$ and $|V_j| = 1$ for $j = 3..k$. For every $i \in V_3 \cup \dots \cup V_k$, we know that $i \notin C$ because $u_i(\sigma)$ is maximum. If $V_1 \cap \left(V'_3 \cup \dots \cup V'_k\right) \neq \emptyset$ or $V_2 \cap \left(V'_3 \cup \dots \cup V'_k\right) \neq \emptyset$ then it contradicts the fact that σ is a NE. Indeed, if a player i belongs to $V_1 \cap \left(V'_3 \cup \dots \cup V'_k\right)$, then it means that it can deviate unilaterally and improve its utility, contradiction with the fact that σ is a NE. Thus $\sigma(i) \in \{1,2\} \Rightarrow \sigma'(i) \in \{1,2\}$. If $V_1 \subseteq C$ then $u_i(\sigma) \geq u_i(\sigma')$ holds for every $i \in V_1$. We deduce that $V_1 \nsubseteq C$. It follows that $|C| \leq |V_1| - 1 + |V_2| = 3$, contradiction with $|C| > 3$. □

One can observe that Proposition 1 is tight when $k = 3$. Every instance with 5 nodes and $k = 3$ admits a SE by the proposition, but one cannot go beyond since for the 6 nodes instance of Figure 1, the optimal cut is not a SE.

Our last positive result is about the existence of an approximate strong equilibrium in every instance. Given a real $\epsilon \geq 0$, a state a is an *ϵ-approximate strong equilibrium* if there is no non-empty coalition $C \subseteq N$ and a profile $a' \in A$ such that $u_i\big((a_{-C}, a')\big) > (1+\epsilon)u_i(a)$ for all $i \in C$. Therefore a 0-approximate SE is a SE. Approximate equilibria are appealing concepts in game theory. They capture the fact a player does not deviate if his gain is negligible. Approximate equilibria are the topic of many recent articles including [13,14,15].

Theorem 1. *Every NE of the* MAX k-CUT *game is a $\frac{1}{k-1}$-approximate SE.*

Proof. Let σ be a NE. Take a player p and suppose w.l.o.g. that $\sigma(p) = k$. Let $E(p, \sigma, i)$ be the set of edges $[p, q]$ such that $\sigma(q) = i$. Let $W(p, \sigma, i) = \sum_{e \in E(p,\sigma,i)} w(e)$ when $E(p, \sigma, i) \neq \emptyset$ and $W(p, \sigma, i) = 0$ otherwise. The utility of p under σ is equal to $\sum_{i=1}^{k-1} W(p, \sigma, i)$. If p unilaterally replaces his strategy by j then his utility becomes $\sum_{\substack{i=1 \\ i \neq j}}^{k} W(p, \sigma, i)$. Since σ is a NE, $\sum_{i=1}^{k-1} W(p, \sigma, i) \geq \sum_{\substack{i=1 \\ i \neq j}}^{k} W(p, \sigma, i)$, which is equivalent to $W(p, \sigma, j) \geq W(p, \sigma, k)$ for every $j \in \{1, \ldots, k-1\}$. Sum up this inequality for every $j \in \{1, \ldots, k-1\}$ to get that $\frac{1}{k-1} \sum_{i=1}^{k-1} W(p, \sigma, i) \geq W(p, \sigma, k)$. The utility of p in any state σ' is at most $\sum_{i=1}^{k} W(p, \sigma, i)$. We deduce that

$$u_p(\sigma') \leq \sum_{i=1}^{k-1} W(p, \sigma, i) + W(p, \sigma, k) \leq (1 + \frac{1}{k-1}) \sum_{i=1}^{k-1} W(p, \sigma, i) = (1 + \frac{1}{k-1}) u_p(\sigma).$$

It follows that σ must be a $\frac{1}{k-1}$-approximate SE. $\square$

Since the MAX $k-$CUT game always possesses a NE, the existence of a $\frac{1}{k-1}$-approximate SE is guaranteed. Interestingly Theorem 1 is tight because there are instances where a NE is a $\frac{1}{k-1}$-approximate SE but not an ϵ-approximate SE for some $\epsilon < \frac{1}{k-1}$.

4 On the Quality of Strong Equilibria

In the previous section we identified some cases where a SE exists. Here we bound the strong price of anarchy but we do not make any assumption on the instance so the result applies for every instance admitting a SE.

Theorem 2. *When $k \geq 3$, the SPoA of the* MAX $k-$CUT *game is at least $\frac{2k-2}{2k-1}$.*

Proof. Let $k \geq 3$ and $G = (V, E)$ be an instance of the MAX $k-$CUT game. Let σ be a SE of G. Let σ^* be an optimal state of G.

Let E_{OS} (resp. E_{OO}) be the set of edges which are only in cut induced by σ (resp. σ^*). Let E_{COM} be the set of edges which are in common. Let OS, OO and COM be the weight of E_{OS}, E_{OO} and E_{COM} respectively. Suppose that the following inequality holds.

$$COM + kOS \geq (2k-2)OO \tag{1}$$

Add $(2k-2)COM$ on both sides to get $(2k-1)COM + kOS \geq (2k-2)(OO + COM)$. Since $k \geq 1 \Leftrightarrow k \leq 2k-1$, we deduce that $(2k-1)(COM + OS) \geq (2k-2)(OO + COM)$. Using $COM + OS = \mathcal{Q}(\sigma)$ and $COM + OO = \mathcal{Q}(\sigma^*)$, the result follows.

Now let us prove inequality (1). We partition V into k^2 sets $X_{i,j} := \{v \in V : \sigma(v) = i \text{ and } \sigma^*(v) = j\}$ for $i, j = 1, \ldots, k$ (see Figure 3 for an illustration). Given two disjoint sets $X \subseteq V$ and $Y \subseteq V$, $w(X,Y)$ denotes $\sum_{x \in X} \sum_{y \in Y} w([x,y])$. Similarly, $w(x,Y)$ denotes $\sum_{y \in Y} w([x,y])$ where $Y \subset V$ and $x \in V \setminus Y$.

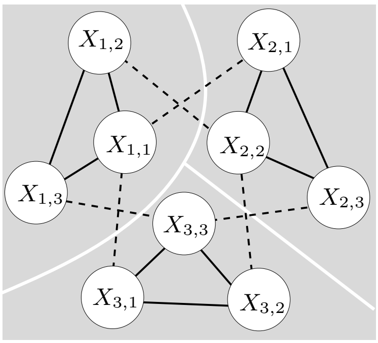


Fig. 3. Partition of V into k^2 sets according to σ and σ^*, case $k = 3$. Dashed edges belong to the cut induced by σ but they are not in the cut induced by σ^*. Solid edges belong to the cut induced by σ^* but they are not in the cut induced by σ. Non represented edges are in the intersection.

Let π be a permutation of $\{1, \cdots, k\}$. One can list $k!$ optimal cuts representing the same state σ^*, one per permutation, if $\sigma^*(v)$ is replaced by $\pi(\sigma^*(v))$ for all $v \in V$. Let us denote by $\pi(\sigma^*)$ the optimum state associated with π. Let V_π be the nodes of V which are misplaced according to $\pi(\sigma^*)$, i.e. $v \in V_\pi$ if $\sigma(v) \neq \pi(\sigma^*(v))$. Let $r_\pi = |V_\pi|$. We are going to rename the nodes of V_π so that $V_\pi = \{v_1, \cdots, v_{r_\pi}\}$.

Since σ is a SE, there is at least one node $v \in V_\pi$ such that $u_v(\sigma) \geq u_v(\sigma_{-V_\pi}, \pi(\sigma^*))$. In other words, v does not benefit if all nodes of V_π replace their strategy in σ by their strategy in $\pi(\sigma^*)$. Rename v by v_1. Again, since σ is a SE, there is at least one node $v \in V_\pi \setminus \{v_1\}$ such that $u_v(\sigma) \geq u_v(\sigma_{-V_\pi \setminus \{v_1\}}, \pi(\sigma^*))$. Rename v by v_2. The procedure is run until all nodes of V_π are renamed, that is $V_\pi = \{v_1, \cdots, v_{r_\pi}\}$.

Let us define V_π^ℓ as $\{v_\ell, v_{\ell+1}, \cdots, v_{r_\pi}\}$ for $1 \leq \ell \leq r_\pi$. For every $v_\ell \in V_\pi$, one has

$$u_{v_\ell}(\sigma) \geq u_{v_\ell}(\sigma_{-V_\pi^\ell}, \pi(\sigma^*)) \tag{2}$$

Take a vertex $v_\ell \in V_\pi$ and suppose that $v_\ell \in X_{i_\ell, j_\ell}$. There are three cases where the weight of an edge $[v_\ell, y]$ is present in $u_{v_\ell}(\sigma)$ but not in $u_{v_\ell}(\sigma_{-V_\pi^\ell}, \pi(\sigma^*))$:

- **CASE 1.** y is not misplaced and he plays in σ the strategy that v_ℓ plays in $\pi(\sigma^*)$. In other words, $y \in X_{i',j'}$ where $i' = \pi(j')$ and $i' = \pi(j_\ell)$; thus, $j' = j_\ell$ since π is a permutation. In this case $[v_\ell, y] \in E_{OS}$.
- **CASE 2.** y is misplaced, he was renamed *after* v_ℓ, he does not play the same strategy as v_ℓ in σ but he plays the same strategy as v_ℓ in $\pi(\sigma^*)$. In other words, $y \in X_{i',j'} \cap V_\pi^{\ell+1}$ where $i' \neq \pi(j')$, $i' \neq i_\ell$ and $j' = j_\ell$. In this case $[v_\ell, y] \in E_{OS}$.
- **CASE 3.** y is misplaced, he was renamed *before* v_ℓ, he plays in σ the strategy that v_ℓ plays in $\pi(\sigma^*)$. In other words, $y \in X_{i',j'} \cap (V_\pi \setminus V_\pi^\ell)$ where $i' \neq \pi(j')$, $i' \neq i_\ell$ and $i' = \pi(j_\ell)$ (hence, $j' \neq j_\ell$). In this case $[v_\ell, y] \in E_{COM}$.

There are three cases where the weight of an edge $[v_\ell, y]$ is present in $u_{v_\ell}(\sigma_{-V_\pi^\ell}, \pi(\sigma^*))$ but not in $u_{v_\ell}(\sigma)$:

- **CASE 4.** y is not misplaced and he plays the same strategy as v_ℓ in σ. In other words, $y \in X_{i',j'}$ where $\pi(j') = i'$ and $i' = i_\ell$. In this case $[v_\ell, y] \in E_{OO}$.
- **CASE 5.** y is misplaced, he was renamed *after* v_ℓ, he plays the same strategy as v_ℓ in σ but he does not play the same strategy as v_ℓ in $\pi(\sigma^*)$. In other words, $y \in X_{i',j'} \cap V_\pi^{\ell+1}$ where $i' = i_\ell$, $j' \neq j_\ell$ and $i' \neq \pi(j')$. In this case $[v_\ell, y] \in E_{OO}$.
- **CASE 6.** y is misplaced, he was renamed *before* v_ℓ and he plays the same strategy as v_ℓ in σ. In other words, $y \in X_{i',j'} \cap (V_\pi \setminus V_\pi^\ell)$ where $i' \neq \pi(j')$ and $i' = i_\ell$. In this case $[v_\ell, y] \notin E_{OO} \cup E_{COM} \cup E_{OS}$ if y plays the same strategy as v_ℓ in $\pi(\sigma^*)$, i.e. $j' = j_\ell$, otherwise $[v_\ell, y] \in E_{OO}$.

Using cases 1 to 6, one can rewrite inequality (2) as follows.

$$w(v_\ell, X_{\pi(j_\ell),j_\ell}) + \sum_{i'=1, i' \neq i_\ell, i' \neq \pi(j_\ell)}^{k} w(v_\ell, X_{i',j_\ell} \cap V_\pi^{\ell+1}) + \sum_{j'=1, j' \neq j_\ell}^{k} w(v_\ell, X_{\pi(j_\ell),j'} \cap (V_\pi \setminus V_\pi^\ell)) \geq \sum_{j'=1}^{k} w(v_\ell, X_{i_\ell, \pi^{-1}(i_\ell)}) + \sum_{j'=1, j' \neq j_\ell, j' \neq \pi^{-1}(i_\ell)}^{k} w(v_\ell, X_{i_\ell,j'} \cap V_\pi^{\ell+1}) + \sum_{j'=1, j' \neq \pi^{-1}(i_\ell)}^{k} w(v_\ell, X_{i_\ell,j'} \cap (V_\pi \setminus V_\pi^\ell)) \tag{3}$$

Using $w(v_\ell, X_{i_\ell,j'} \cap (V_\pi \setminus V_\pi^\ell)) \geq 0$ (the weight of an edge is always non negative) and the previous inequality, we get that:

$$w(v_\ell, X_{\pi(j_\ell),j_\ell}) + \sum_{i'=1, i' \neq i_\ell, i' \neq \pi(j_\ell)}^{k} w(v_\ell, X_{i',j_\ell} \cap V_\pi^{\ell+1}) + \sum_{j'=1, j' \neq j_\ell}^{k} w(v_\ell, X_{\pi(j_\ell),j'} \cap (V_\pi \setminus V_\pi^\ell)) \geq \sum_{j'=1}^{k} w(v_\ell, X_{i_\ell, \pi^{-1}(i_\ell)}) + \sum_{j'=1, j' \neq j_\ell, j' \neq \pi^{-1}(i_\ell)}^{k} w(v_\ell, X_{i_\ell,j'} \cap V_\pi^{\ell+1}) + \sum_{j'=1, j' \neq \pi^{-1}(i_\ell), j' \neq j_\ell}^{k} w(v_\ell, X_{i_\ell,j'} \cap (V_\pi \setminus V_\pi^\ell)) \tag{4}$$

Actually, we have "removed" the weight of edges $[v_\ell, y] \notin E_{OO} \cup E_{COM} \cup E_{OS}$ which appear on Case 6. Observe that the last two terms of inequality (4) can be grouped as follows:

$$\begin{aligned} w(v_\ell, X_{\pi(j_\ell),j_\ell}) + \sum_{i'=1, i'\neq i_\ell, i'\neq \pi(j_\ell)}^{k} w(v_\ell, X_{i',j_\ell} \cap V_\pi^{\ell+1}) + \\ \sum_{j'=1, j'\neq j_\ell}^{k} w(v_\ell, X_{\pi(j_\ell),j'} \cap (V_\pi \setminus V_\pi^{\ell})) \geq \sum_{j'=1}^{k} w(v_\ell, X_{i_\ell,\pi^{-1}(i_\ell)}) + \\ \sum_{j'=1, j'\neq j_\ell, j'\neq \pi^{-1}(i_\ell)}^{k} w(v_\ell, X_{i_\ell,j'} \cap V_\pi) \end{aligned} \qquad (5)$$

Summing inequality (5) for $\ell = 1, \cdots, r_\pi$, i.e. for each $v_\ell \in V_\pi$, we get that

$$\begin{aligned} \sum_{\ell=1}^{r_\pi} \Big(w(v_\ell, X_{\pi(j_\ell),j_\ell}) + \sum_{i'=1, i'\neq i_\ell, i'\neq \pi(j_\ell)}^{k} w(v_\ell, X_{i',j_\ell} \cap V_\pi^{\ell+1}) + \\ \sum_{j'=1, j'\neq j_\ell}^{k} w(v_\ell, X_{\pi(j_\ell),j'} \cap (V_\pi \setminus V_\pi^{\ell}))\Big) \geq \sum_{\ell=1}^{r_\pi} \Big(\sum_{j'=1}^{k} w(v_\ell, X_{i_\ell,\pi^{-1}(i_\ell)}) + \\ \sum_{j'=1, j'\neq j_\ell, j'\neq \pi^{-1}(i_\ell)}^{k} w(v_\ell, X_{i_\ell,j'} \cap V_\pi)\Big) \end{aligned} \qquad (6)$$

Now we give an intermediate property (proof in the appendix).

Property 1. For any edge $[x, y]$, if $w([x, y])$ appears in the left-hand part of inequality (6) then it appears once.

Cases 1 and 2 state that if $w[x, y]$ appears in the left-hand part of inequality (6) then $[x, y] \in E_{OS}$. Using the fact that $w[x, y]$ appears at most once (by Property 1), we deduce that

$$\sum_{\ell=1}^{r_\pi} \Big(w(v_\ell, X_{\pi(j_\ell),j_\ell}) + \sum_{i'=1, i'\neq i_\ell, i'\neq \pi(j_\ell)}^{k} w(v_\ell, X_{i',j_\ell} \cap V_\pi^{\ell+1})\Big) \leq OS \qquad (7)$$

Case 3 states that if $w[x, y]$ appears in the left-hand part of inequality (6) then $[x, y] \in E_{COM}$ and $\{x, y\} \subseteq V_\pi$. Moreover $w([x, y])$ appears at most once, we deduce that

$$\sum_{\ell=1}^{r_\pi} \Big(\sum_{j'=1, j'\neq j_\ell}^{k} w(v_\ell, X_{\pi(j_\ell),j'} \cap (V_\pi \setminus V_\pi^{\ell}))\Big) \leq \sum_{i=1}^{k} \sum_{j=1, j\neq \pi^{-1}(i)}^{k} \sum_{j'=1, j'\neq j}^{k} w(X_{i,j}, X_{\pi(j),j'}) \qquad (8)$$

Using inequalities (7) and (8), we obtain the following upper bound on the left-hand part of inequality (6):

$$OS + \sum_{i=1}^{k} \sum_{j=1, j \neq \pi^{-1}(i)}^{k} \sum_{j'=1, j' \neq j}^{k} w(X_{i,j}, X_{\pi(j),j'}) \tag{9}$$

Now we give a second intermediate property (proof in the appendix).

Property 2. Let x and y be two vertices of $X_{i,j}$ and $X_{i',j'}$ respectively. Among the $k!$ possible permutations π of $\{1, \cdots, k\}$, exactly $(k-1)!$ of them satisfy simultaneously: $\pi(j) \neq i$, $\pi(j') \neq i'$ and $i' = \pi(j)$.

Now sum up inequality (9) for all permutations π of $\{1, \ldots, k\}$. We can give the following upper bound of the result: $k!OS + (k-1)!COM$. Indeed every edge in E_{OS} appears exactly once for each permutation π. Concerning the edges $[x, y] \in E_{COM}$, each one appears at most $(k-1)!$ times by Property 2.

Now we focus on the right part of inequality (6). Take an edge $[x, y]$ such that $x \in X_{i,j}$ and $y \in X_{i',j'}$. The weight of $[x, y]$ does not appear in the right part of inequality (6) if $i \neq i'$. $w([x, y])$ appears once in the right part of inequality (6) if $\pi(j) = i = i'$ and $\pi(j') \neq i'$; in this case $x \notin V_\pi$ whereas $y \in V_\pi$. $w([x, y])$ appears twice in the right part of inequality (6) if $i \neq i'$, $\pi(j) \neq i$ and $\pi(j') \neq i'$; in this case $x, y \in V_\pi$.

By definition a vertex which is not in V_π must be in $X_{i,\pi^{-1}(i)}$ for some $i \in \{1, \cdots, k\}$. Then inequality (6) is equal to

$$\sum_{i=1}^{k} \Big(\sum_{j=1, j \neq \pi^{-1}(i)}^{k} w(X_{i,\pi^{-1}(i)}, X_{i,j}) + \sum_{j=1, j \neq \pi^{-1}(i)}^{k} \sum_{j'=1,, j' \neq \pi^{-1}(i)}^{k} w(X_{i,j}, X_{i,j'}) \Big). \tag{10}$$

If we sum up inequality (10) over the $k!$ permutations of $\{1, \cdots, k\}$ then every term $w(X_{i,j}, X_{i,j'})$, for $j \neq j'$, appears exactly $(k-1)!(2k-2)$ times. Indeed, a set $X_{i,j}$ satisfies $\pi(j) = i$ exactly $(k-1)!$ times, whereas $\pi(j) \neq i$ holds $k! - (k-1)!$ times. In addition it can not be $\pi(j) = \pi(j') = i$ because $j \neq j'$: $2(k-1)! + 2(k! - 2(k-1)!) = (k-1)!(2k-2)$. Therefore summing up inequality (10) over the $k!$ permutations of $\{1, \cdots, k\}$ gives $(k-1)!(2k-2)OO$.

Finally, summing up inequality (6) over all possible permutations of $\{1, \cdots, k\}$, we get that $k!OS + (k-1)!COM \geq (k-1)!(2k-2)OO$ which is equivalent to inequality (1). □

The following matching upper bound on the SPoA of the MAX $k-$CUT game can be derived.

Proposition 2. *The SPoA of the* MAX $k-$CUT *game is at most* $(2k-2)/(2k-1)$.

Proof. Consider an instance with $2k$ vertices $\{v_1, \cdots, v_k\} \cup \{u_1, \cdots, u_k\}$ and the following $2k-1$ edges of weight 1: $[v_1, v_i]$ for $i = 2..k$, $[u_k, u_i]$ for $i = 1..k-1$

and $[v_1, u_k]$. See Figure 4. If every v_i plays i while every u_j plays j then the state is optimal and it has weight $2k-1$. If every v_i plays i while every u_j plays $j+1 \bmod k$ (i.e. u_k plays 1, u_1 plays 2, etc) then the state is a SE of weight $2k-2$. Indeed every node in $\{v_2, \cdots, v_k\} \cup \{u_1, \cdots, u_{k-1}\}$ has the maximum utility that he can expect in this instance so none of them has incentive to deviate. Now if v_1 or u_k moves then one of his incident edge would not be the cut anymore while only $[v_1, u_k]$ can enter the cut. Then v_1 and u_k can not alone or together increase their utility. □

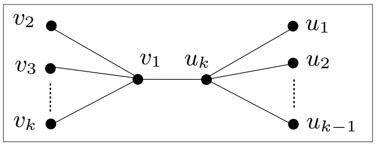


Fig. 4. An instance for the upper bound on the SPoA

5 Concluding Remarks and Open Questions

The main question remaining open is to prove or disprove that every instance of the MAX $k-$CUT game possesses a strong equilibrium. The technique used by Harks, Klimm and Möhring [6] considers all improving pairs of strategies while Holzman and Law-Yone [7] require minimal improving pairs of strategies (an improving pair of strategies is not minimal if a proper subset of the coalition can also perform an improvement). The cycle presented in Figure 2 is made of three improvements which are not minimal. Then it would be interesting to investigate the existence of a strong potential function restricted to minimal improvements.

A Nash equilibrium of the MAX k-CUT game a $\frac{1}{k-1}$-approximate SE, can we prove the existence of an ϵ-approximate SE for some $\epsilon < \frac{1}{k-1}$? It is known from [5] that every instance of the game possesses a 3-SE, a stronger notion of equilibrium than the NE. However, it is not difficult to build an instance containing a 3-SE σ which is also a $\frac{1}{k-1}$-approximate SE, but σ is not an ϵ'-approximate SE for $\epsilon' < \frac{1}{k-1}$. A promising direction would be to bound the ϵ such that every optimal cut is an ϵ-approximate SE. This ϵ cannot be 0 since an optimal cut is not necessarily a SE and a better lower bound on this ϵ can be derived from the instance of Figure 1.

The *price of stability* (PoS) is a well studied ratio whose definition is close to the price of anarchy [13]. It is the worst case ratio between the social welfare of the best NE and a socially optimal state. Since an optimal cut is a NE in the MAX $k-$CUT game, the PoS is 1. It is natural to restrict this notion to strong equilibria [4]. We know that the price of stability for strong equilibria is 1 when $k = 2$ and strictly less than 1 when $k \geq 3$ (see the instance of Figure 1). It would be interesting to give an explicit lower bound for the case $k \geq 3$.

References

1. Koutsoupias, E., Papadimitriou, C.: Worst case equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
2. Hoefer, M.: Cost Sharing and Clustering under Distributed Competition. PhD thesis, Universität Konstanz (2007)
3. Aumann, R.J.: Acceptable points in games of perfect information. Pacific Journal of Mathematics 10, 381–417 (1960)
4. Andelman, N., Feldman, M., Mansour, Y.: Strong price of anarchy. Games and Economic Behavior 65, 289–317 (2009)
5. Gourvès, L., Monnot, J.: On strong equilibria in the max cut game. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 608–615. Springer, Heidelberg (2009)
6. Harks, T., Klimm, M., Möhring, R.H.: Strong nash equilibria in games with the lexicographical improvement property. In: Leonardi, S. (ed.) WINE 2009. LNCS, vol. 5929, pp. 463–470. Springer, Heidelberg (2009)
7. Holzman, R., Law-Yone, N.: Strong equilibrium in congestion games. Games and Economic Behavior 21, 85–101 (1997)
8. Rozenfeld, O., Tennenholtz, M.: Strong and correlated strong equilibria in monotone congestion games. In: Spirakis, P.G., Mavronicolas, M., Kontogiannis, S.C. (eds.) WINE 2006. LNCS, vol. 4286, pp. 74–86. Springer, Heidelberg (2006)
9. Christodoulou, G., Mirrokni, V.S., Sidiropoulos, A.: Convergence and approximation in potential games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 349–360. Springer, Heidelberg (2006)
10. Fabrikant, A., Papadimitriou, C.H., Talwar, K.: The complexity of pure nash equilibria. In: STOC, pp. 604–612 (2004)
11. Rosenthal, R.W.: A class of games possessing pure-strategy nash equilibria. International Journal of Game Theory 2, 65–67 (1973)
12. Monderer, D., Shapley, L.S.: Potential games. Games and Economic Behavior 14, 124–143 (1996)
13. Anshelevich, E., Dasgupta, A., Kleinberg, J.M., Tardos, É., Wexler, T., Roughgarden, T.: The price of stability for network design with fair cost allocation. In: Proc. of FOCS 2004, pp. 295–304 (2004)
14. Chien, S., Sinclair, A.: Convergence to approximate nash equilibria in congestion games. In: Proc. of SODA 2007, pp. 169–178 (2007)
15. Christodoulou, G., Koutsoupias, E., Spirakis, P.G.: On the performance of approximate equilibria in congestion games. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 251–262. Springer, Heidelberg (2009)

Appendix

Property 1. For any edge $[x, y]$, if $w([x, y])$ appears in the left-hand part of inequality (6) then it appears once.

Proof.
- If neither x nor y belong to V_π then $w([x, y])$ can not appear since in this case $[x, y] \in E_{COM}$ or $[x, y] \notin E_{COM} \cup E_{OS} \cup E_{OO}$.
- If only x (or only y) is in V_π then $w([x, y])$ can appear at most once, in the term $w(v_{\ell^*}, X_{\pi(j_{\ell^*}),j_{\ell^*}})$ where $x = v_{\ell^*}$.

- If x and y both belong to V_π then there are ℓ^* and ℓ^{**} such that $x = v_{\ell^*}$ and $y = v_{\ell^{**}}$. Without loss of generality, suppose that $\ell^* < \ell^{**}$, $v_{\ell^*} \in X_{i_{\ell^*},j_{\ell^*}}$ and $v_{\ell^{**}} \in X_{i_{\ell^{**}},j_{\ell^{**}}}$. $w([v_{\ell^*}, v_{\ell^{**}}])$ can appear when $\ell = \ell^*$ and when $\ell = \ell^{**}$, i.e. in the terms $\sum_{i'=1,i'\neq i_{\ell^*},i'\neq\pi(j_{\ell^*})}^{k} w(v_{\ell^*}, X_{i',j_{\ell^*}} \cap V_\pi^{\ell^*+1})$ and $\sum_{j'=1,j'\neq j_{\ell^{**}}}^{k} w(v_{\ell^{**}}, X_{\pi(j_{\ell^{**}}),j'} \cap (V_\pi \setminus V_\pi^{\ell^{**}}))$. However the first term imposes $j_{\ell^{**}} = j_{\ell^*}$ whereas the second one imposes $j_{\ell^{**}} \neq j_{\ell^*}$, contradiction. □

Property 2. Let x and y be two vertices of $X_{i,j}$ and $X_{i',j'}$ respectively. Among the $k!$ possible permutations π of $\{1, \cdots, k\}$, exactly $(k-1)!$ of them satisfy simultaneously: $\pi(j) \neq i$, $\pi(j') \neq i'$ and $i' = \pi(j)$.

Proof. We first observe that $j \neq j'$ by $\pi(j') \neq i'$ and $i' = \pi(j)$. Moreover $i \neq i'$ by $\pi(j) \neq i$ and $i' = \pi(j)$. Then we conduct a case study where $i \neq i'$ and $j \neq j'$. Let a, b, c, d are four distinct elements of $\{1, \cdots, k\}$:

- Case $i = a$, $j = a$, $i' = b$ and $j' = b$. π must satisfy $b = \pi(a)$. Since a permutation is a bijection, it follows that $\pi(b) \neq b$ and $\pi(a) \neq a$. Then π satisfies the three assertions iff $b = \pi(a)$ and there are $(k-1)!$ permutations satisfying $b = \pi(a)$.
- Case $i = a$, $j = b$, $i' = b$ and $j' = a$. π must satisfy $b = \pi(b)$. It follows that $\pi(b) \neq a$ and $\pi(a) \neq b$. Then π satisfies the three assertions iff $b = \pi(b)$.
- Case $i = a$, $j = b$, $i' = b$ and $j' = c$. π must satisfy $b = \pi(b)$. It follows that $\pi(b) \neq a$ and $\pi(c) \neq b$. Then π satisfies the three assertions iff $b = \pi(b)$.
- Case $i = b$, $j = c$, $i' = a$ and $j' = b$. π must satisfy $a = \pi(c)$. It follows that $\pi(b) \neq a$ and $\pi(c) \neq b$. Then π satisfies the three assertions iff $a = \pi(c)$.
- Case $i = a$, $j = b$, $i' = c$ and $j' = c$. π must satisfy $c = \pi(b)$. It follows that $\pi(c) \neq c$ and $\pi(b) \neq a$. Then π satisfies the three assertions iff $c = \pi(b)$.
- Case $i = c$, $j = c$, $i' = a$ and $j' = b$. π must satisfy $a = \pi(c)$. It follows that $\pi(c) \neq c$ and $\pi(b) \neq a$. Then π satisfies the three assertions iff $a = \pi(c)$.
- Case $i = a$, $j = b$, $i' = c$ and $j' = d$. π must satisfy $c = \pi(b)$. It follows that $\pi(d) \neq c$ and $\pi(b) \neq a$. Then π satisfies the three assertions iff $c = \pi(b)$. □

Kernel and Fast Algorithm for Dense Triplet Inconsistency

Sylvain Guillemot[1] and Matthias Mnich[2,⋆]

[1] Lehrstuhl für Bioinformatik, Friedrich-Schiller Universität Jena, Ernst-Abbe Platz 2, 00743 Jena, Germany
sylvain.guillemot@uni-jena.de

[2] Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
m.mnich@tue.nl

Abstract. We study the parameterized complexity of inferring supertrees from sets of rooted triplets, an important problem in phylogenetics. For a set L of labels and a dense set $\mathcal{R}$ of triplets distinctly leaf-labeled by 3-subsets of L we seek a tree distinctly leaf-labeled by L and containing all but at most p triplets from $\mathcal{R}$ as homeomorphic subtree. Our results are the first polynomial kernel for this problem, with $O(p^2)$ labels, and a subexponential fixed-parameter algorithm running in time $2^{O(p^{1/3}\log p)} + O(n^4)$.

1 Introduction

In phylogenetics, distinctly leaf-labeled trees represent the evolutionary history of a set of species, each species corresponding to a label. *Supertree methods* are widely used in this field, in order to construct a large tree from smaller trees on overlapping subsets of species. The simplest approach in this setting consists in inferring the smallest possible informative trees. Such trees are either *rooted triplets*, rooted binary trees on three labels, or *quartets*, unrooted ternary trees on four labels. Quartet methods received prominent attention over the last decade, whereas triplets methods were somewhat overlooked though they enjoy similar interesting properties and may be less computationally expensive.

Let L be a set of labels. A set $\mathcal{R}$ of rooted binary (unrooted ternary) trees distinctly leaf-labeled by subsets of L is *consistent* if there exists a rooted binary (unrooted ternary) tree distinctly leaf-labeled by L and containing every element of $\mathcal{R}$ as homeomorphic subtree; and *inconsistent* otherwise. Deciding consistency for a set of triplets is polynomial-time solvable [1]; in contrast, this problem is NP-hard for quartets [19]. For an inconsistent triplet set $\mathcal{R}$, two approaches have been studied to obtain a consistent triplet set from it.

The first approach is to find a minimum-cardinality subset $L' \subseteq L$ such that removing all leaves labeled by elements from L' from triplets in $\mathcal{R}$ yields a

⋆ Matthias Mnich has been partly supported by the Netherlands Organisation for Scientific Research (NWO), grant 639.033.403.

© Springer-Verlag Berlin Heidelberg 2010

consistent set of trees. The problem of finding L' is the dual of the maximum agreement supertree problem [16,14], and we refer to it as MINIMUM LABEL INCONSISTENCY (MLI). The parameterization of MLI by $|L'|$ is denoted as p-MLI; this problem is NP-hard and fixed-parameter intractable [6]. The restriction of p-MLI to instances that are *dense*, that is $\mathcal{R}$ contains exactly one triplet for each 3-subset of L, is fixed-parameter tractable; call this problem p-DENSE MLI. Fixed-parameter tractability of p-DENSE MLI follows from that for dense triplet sets, consistency has a characterization in terms of obstructions involving at most four labels and three triplets. More precisely, for a dense triplet set $\mathcal{R}$ on n labels, we can build in time $O(n^4)$ the set of obstructions, and in time $O(n^3)$ either find an obstruction or decide if $\mathcal{R}$ is consistent [14]. These results lead to a $O(4^p n^3)$-time algorithm for p-DENSE MLI [14].

A fixed-parameter algorithm. The other approach is to remove a minimum-size set $\mathcal{R}'$ of triplets from $\mathcal{R}$ such that the set $\mathcal{R} \setminus \mathcal{R}'$ is consistent. The problem of finding $\mathcal{R}'$ is the ROOTED TRIPLET INCONSISTENCY (RTI) problem [9,10]. We denote the restriction of RTI to dense instances, and parameterized by the size of $\mathcal{R}'$, by p-DENSE RTI. This restriction of RTI is still NP-hard [10]. Our first result is a simple $O(4^p n^3)$-time algorithm for p-DENSE RTI, based on characterization of consistency in terms of obstructions and given in Section 3. For general instances, the parameterization of RTI by $|\mathcal{R}'|$ is not fixed-parameter tractable unless some unlikely collapse of complexity classes occurs [10].

A polynomial kernel. Any fixed-parameter tractable problem Π has a *kernelization algorithm*, or *kernel*, which is an algorithm that given a pair (x, p) outputs in time polynomial in $|x| + p$ a pair (x', p') such that $(x, p) \in \Pi$ if and only if $(x', p') \in \Pi$ and $|x'|, p' \leq g(p)$, where g is some computable function. The function g is referred to as the size of the kernel, and if $g(p) = p^{O(1)}$ then Π is said to admit a polynomial kernel. Kernels are important to practically solve instances of NP-hard problems through data reduction, but not every fixed-parameter tractable problem admits a polynomial kernel [8]. Our second result is the first polynomial kernel for p-DENSE RTI: in Section 4, we describe a kernelization algorithm which produces in time $O(n^4)$ a kernel with $O(p^2)$ labels.

Subexponential fixed-parameter algorithm. While interesting by itself, our polynomial kernel serves as the basis for our third result. In Section 5, we present a subexponential fixed-parameter algorithm for p-DENSE RTI. The algorithm applies the method of *chromatic coding*, which was recently introduced by Alon et al. [3] to solve the tournament feedback arc set problem (FAST) in time $O(n^3 + 2^{O(p^{1/2} \log p)})$. Similarly, chromatic coding provided a subexponential fixed-parameter algorithm for the DENSE BETWEENNESS problem parameterized by solution size p, running in time $O(n^4 + 2^{O(p^{1/3} \log p)})$ [18]. Let us remark that the trees we are constructing possess a more complex structure than the linear orderings encountered in ranking problems such as FAST and BETWEENNESS.

Chromatic coding is a variant of the *color coding* technique by Alon et al. [4]. It requires several ingredients: (i) a kernel of quadratic size, (ii) an algorithm solving instances of size n and colored with k colors in time $n^{O(k)}$, (iii) a coloring lemma stating for a given solution of size p, a "proper coloring" of the solution with $k = O(p^{1/3})$ colors exists and can be found in subexponential time. We obtain (i) and (ii) for p-Dense RTI, while (iii) follows from results of Alon et al. [3,18]. Overall, we obtain an algorithm for p-Dense RTI running in time $O(n^4 + 2^{O(p^{1/3} \log p)})$. This algorithm provides an exponential speed-up over our search-tree based $O(4^p n^3)$-time algorithm. Note that p can be as large as $\binom{n}{3}$, and for this value our algorithm yields a running time of $2^{O(n \log n)}$ which is asymptotically not better than the brute-force algorithm which enumerates every tree with n labels and checks consistency for each such tree. However, for practical instances p can be expected to be much smaller than $\binom{n}{3}$, which makes our result valuable in real-life applications.

2 Preliminaries

Let L be a set of labels. We say that T is a *tree over* L if T is a rooted tree whose leaves are injectively labeled by the elements of L; we denote by $L(T)$ the set of labels appearing at the leaves of T. We use a parenthesized notation for trees: if $T_1, ..., T_m$ are trees over L with disjoint label sets, we denote by $(T_1, ..., T_m)$ the tree over L whose root has each T_i has a child subtree.

Given $L' \subseteq L$, the *restriction of* T *to* L', denoted by $T|L'$, is the homeomorphic subtree of T containing leaves labeled by elements of L'. When T is a binary tree over L with three leaves, we say that T is a *rooted triplet* over L. We denote by $ab|c$ the rooted triplet containing three labels a, b, c with a, b grouped together in a same child subtree of the root. We say that $\mathcal{R}$ is a *triplet set over* L if $\mathcal{R}$ is a set of rooted triplets over L. $\mathcal{R}$ is *dense* (resp. *minimally dense*) if for each $x, y, z \in L$ distinct, there is at least (resp. exactly) one $t \in \mathcal{R}$ such that $L(t) = \{x, y, z\}$. Given $L' \subseteq L$, the *restriction of* $\mathcal{R}$ *to* L' is the triplet set $\mathcal{R}|L'$ which contains the triplets $t \in \mathcal{R}$ such that $L(t) \subseteq L'$.

Given a binary tree T over L and a rooted triplet $t = ab|c$, we say that t is *consistent* with T (or symmetrically that T is consistent with t) if $T|\{a, b, c\} = t$. We denote by $rt(T)$ the set of rooted triplets over L which are consistent with T. Given $\mathcal{R}$ triplet set over L, we say that $\mathcal{R}$ is *consistent* with T if $\mathcal{R} \subseteq rt(T)$. Note that if $\mathcal{R}$ is minimally dense, this must be an equality; in this case we also say that $\mathcal{R}$ *represents* T. A *conflict* in $\mathcal{R}$ is a set $C \subseteq L$ such that $\mathcal{R}|C$ is not consistent. A *t-conflict* in $\mathcal{R}$ is an inconsistent set $S \subseteq \mathcal{R}$. An *st-conflict* in $\mathcal{R}$ is a t-conflict of the form $\{ab|c, cd|b, bd|a\}$ or $\{ab|c, cd|b, ad|b\}$.

The following Proposition gives a local characterization of consistency for minimally dense sets. It is analogous to results for quartets originating in the work of Bandelt and Dress [5,13].

Proposition 1. *Let $\mathcal{R}$ be a minimally dense triplet set over L. The following are equivalent: (i) $\mathcal{R}$ is consistent; (ii) $\mathcal{R}$ contains no conflict of size 4; (iii) $\mathcal{R}$ contains no t-conflict of size 3; (iv) $\mathcal{R}$ contains no st-conflict.*

Let $\mathcal{R}$ be a triplet set over L, and let T be a binary tree over L. We denote by $i(\mathcal{R}, T) := (\mathcal{R}|L(T)) \backslash rt(T)$ the set of triplets of $\mathcal{R}$ inconsistent with T, and we define $I(\mathcal{R}, T) = |i(\mathcal{R}, T)|$.

The RTI problem takes a triplet set $\mathcal{R}$ over L, and seeks a binary tree T over L with $I(\mathcal{R}, T)$ minimum. The value of the optimum is denoted by $\mathsf{MRTI}(\mathcal{R})$. Note that this is equivalent to: (i) removing a minimum number of triplets to obtain consistency, (ii) editing a minimum number of triplets to obtain consistency.

3 A Simple Fixed-Parameter Algorithm

For the needs of the algorithm, we consider an annotated version of the problem where certain triplets are *locked*, meaning that they cannot be edited. We use a bounded-search approach: at each step, we identify an st-conflict, edit one triplet and lock it. The following Lemma, which can be checked by exhaustive verification, will guarantee that for a given st-conflict, four different editions cover all possible cases.

Lemma 1. *Any tree on $\{a, b, c, d\}$ contains one of the triplets $bc|a, ac|b, bd|c, ab|d$.*

This yields a simple fixed-parameter algorithm, as follows. We recall that p is the maximum number of deletions allowed, and that $n = |L|$ is the number of labels.

Theorem 1. *The p-*DENSE RTI *problem can be solved in $O(4^p n^3)$ time.*

Proof. We use bounded search. Given a minimally dense triplet set $\mathcal{R}$, using the algorithm of [14] (Theorem 5), in $O(n^3)$ time we can either find an st-conflict or conclude that $\mathcal{R}$ is consistent. If we find an st-conflict $\{t_1, t_2, t_3\}$ with $t_1 = ab|c, t_2 = cd|b$ and $t_3 \in \{bd|a, ad|b\}$, then we branch on the four following cases: (i) transform t_1 into $bc|a$, (ii) transform t_1 into $ac|b$, (iii) transform t_2 into $bd|c$, (iv) transform t_3 into $ab|d$. In each case we lock the new triplet and decrement p. The correctness of the branching step follows from Lemma 1, and this leads to an algorithm running in time $O(4^p n^3)$. □

Note that the above algorithm can be adapted to run in $O(4^p n + n^4)$ time (using ideas similar to [13]): we build in $O(n^4)$ time the set $\mathcal{C}$ of st-conflicts, then at each branching step we identify an st-conflict, edit a triplet and update $\mathcal{C}$ in $O(n)$ time.

4 A Polynomial Kernel

In this section, we present three reduction rules which will lead to a problem kernel for the p-DENSE RTI problem with a quadratic number of labels. For convenience, we describe the problem kernel for the annotated version of the problem, since this formulation allows an earlier detection of certain negative instances (when an already locked triplet has to be further edited). However, we note that the problem kernel applies to the unannotated version as well.

Let $t = ab|c \in \mathcal{R}$. A *sunflower (with center t)* is a family $C_1, ..., C_m$ of st-conflicts such that (i) all the C_i's contain t, (ii) the C_i's have distinct fourth labels, i.e. $L(C_i) \cap L(C_j) = \{a, b, c\}$ whenever $i \neq j$. The following observation will allow us to prove the correctness of our reduction rules.

Observation 1. *Let $C_1, ..., C_m$ be a sunflower with center t. Then the sets $C_1\backslash\{t\}$, ..., $C_m\backslash\{t\}$ are disjoint.*

There is a simple sufficient condition for failure, formulated by the following rule, whose correctness is immediate in view of Observation 1.

Rule 1. If a locked triplet is the center of a sunflower of size $> p$, then fail.

The second rule will allow us to handle unlocked triplets which are involved in a large number of conflicts. It relies on the following observation.

Lemma 2. *Let t_1, t_2 be two triplets such that $L(t_1) = \{a, b, c\}, L(t_2) = \{b, c, d\}$. Then:*

1. *either there is a unique tree T on $\{a, b, c, d\}$ consistent with t_1, t_2;*
2. *or for each triplet t_3 on $\{a, b, d\}$, the set $\{t_1, t_2, t_3\}$ is consistent.*

Proof. By symmetry, it suffices to consider the following cases:

– $t_1 = bc|a, t_2 = bc|d$: then we are in Case 2, since
  - for $t_3 = ab|d$, the tree $T = (((b, c), a), d)$ is consistent with $\{t_1, t_2, t_3\}$;
  - for $t_3 = ad|b$, the tree $T = ((b, c), (a, d))$ is consistent with $\{t_1, t_2, t_3\}$;
  - for $t_3 = bd|a$, the tree $T = (((b, c), d), a)$ is consistent with $\{t_1, t_2, t_3\}$.

– $t_1 = bc|a, t_2 = bd|c$: then we are in Case 1 since only the tree $T = (((b, d), c), a)$ is consistent with t_1, t_2.

– $t_1 = ba|c, t_2 = bd|c$: then we are in Case 2, since
  - for $t_3 = ab|d$, the tree $T = (((a, b), d), c)$ is consistent with $\{t_1, t_2, t_3\}$;
  - for $t_3 = ad|b$, the tree $T = (((a, d), b), c)$ is consistent with $\{t_1, t_2, t_3\}$;
  - for $t_3 = bd|a$, the tree $T = ((a, (b, d)), c)$ is consistent with $\{t_1, t_2, t_3\}$.

– $t_1 = ba|c, t_2 = cd|b$: then we are in Case 1 since only the tree $T = ((a, b), (c, d))$ is consistent with t_1, t_2.

This proves the Lemma. □

Corollary 1. *Let $C = \{t_1, t_2, t_3\}$ be an st-conflict. There exists a unique way to edit t_3 in a triplet t'_3 such that $C' = \{t_1, t_2, t'_3\}$ is consistent.*

Proof. C must involve the four labels a, b, c, d and w.l.o.g. we can assume that $L(t_1) = \{a, b, c\}, L(t_2) = \{b, c, d\}$. Then we are in the conditions of Lemma 2, and we must be in Case 1 since C is an st-conflict. It follows that there is a unique tree T on $\{a, b, c, d\}$ consistent with t_1, t_2, hence the triplet $T|L(t_3)$ is the only way to edit t_3 to achieve consistency. □

By Corollary 1, if a triplet t belongs to an st-conflict C, in order to achieve consistency there is a unique way to edit t into a triplet t' without editing the other triplets of C. Let $S(t, C)$ denote this alternative triplet t'. We are now ready to formulate the rule and prove its correctness.

Rule 2. If an unlocked triplet t is the center of a sunflower $C_1, ..., C_m$ with $m > 2p$: let t' be the majority element among the triplets $S(t, C_i)$. Transform t into t', lock it, and decrease p.

Lemma 3. *Rule 2 is correct.*

Proof. Suppose that there is a solution S editing $\leq p$ triplets. Let $C_1, ..., C_i$ be the st-conflicts containing a triplet distinct from t which was edited in S. By Observation 1, it follows that $i \leq p$. For the remaining st-conflicts C_j $(i+1 \leq j \leq C_m)$, t is the only triplet edited in S, hence by Corollary 1 $S(t, C_j)$ equals the triplet t' into which t has been edited. Since $m > 2p$, it follows that t' is the majority element among the triplets $S(t, C_j)$ $(1 \leq j \leq m)$. We conclude that in any solution S, t is edited into t'. □

Suppose that $(\mathcal{R}, p)$ is an instance reduced with respect to Rules 1 and 2. Let L be the label set of $\mathcal{R}$, partition L into L_1 (the set of labels which appear in some st-conflict) and $L_2 = L \backslash L_1$. We can bound the size of L_1, according to the following Lemma.

Lemma 4. *If $(\mathcal{R}, p)$ is a positive instance, then $|L_1| \leq 2p^2 + 3p$.*

Proof. Suppose that $\mathcal{R}$ can be made consistent by editing triplets $t_1, ..., t_i$ with $i \leq p$. Then $L_1 = L_1' \cup L_1''$, where L_1' is the set of labels appearing in some t_j and $L_1'' = L_1 \backslash L_1'$. Clearly, $|L_1'| \leq 3p$.

We show that $|L_1''| \leq 2p^2$. For each $x \in L_1''$, let C_x be an st-conflict containing x, then C_x must contain one triplet t_j, let $j := j(x)$. Now, since $(\mathcal{R}, p)$ is reduced with respect to Rules 1 and 2, a sunflower with center t_j has size $\leq 2p$. It follows that for each $1 \leq j \leq i$, there are at most $2p$ elements $x \in L_1''$ such that $j(x) = j$. We conclude that $|L_1''| \leq 2pi \leq 2p^2$. □

It remains to handle the labels of L_2. In fact, we will simply remove them.

Rule 3. Remove the labels of L_2.

In the following, we justify that labels not appearing in an st-conflict can be safely removed. Note that the situation is similar for the FAST problem, where we can safely remove vertices not involved in a directed triangle without changing the optimum. As pointed out in [15], for edge-modification problems in general it is not safe to remove vertices outside of an obstruction (unlike the case of vertex-deletion problems). However, it is interesting to observe that this holds in most known kernelizations for edge-modification problems.

Lemma 5. *Rule 3 is correct.*

Proof (Sketch). Say that an *L-tree* is a rooted tree with each leaf x labeled by a subset $L_x \subseteq L$, such that the sets L_x partition L. A binary L-tree T induces a set of triplets $rt(T)$ where: $uv|w \in rt(T)$ if and only if there exists x, y, z leaves of T with $u \in L_x, v \in L_y, w \in L_z$ and $xy|z$ a triplet of T. We say that T *complies* with $\mathcal{R}$ if $rt(T) \subseteq \mathcal{R}$. Given $S \subseteq L$, we say that T *resolves* S if each $u \in S$ corresponds to a leaf of T labeled by $\{u\}$. The definition of L_2 entails the following result:

Claim 1. There exists a binary L-tree T which complies with $\mathcal{R}$ and which resolves L_2.

To prove the correctness of Rule 3, we now show that: if $\mathcal{R}' = \mathcal{R}\backslash L_2$, then $\mathsf{MRTI}(\mathcal{R}') = \mathsf{MRTI}(\mathcal{R})$. On the one hand, $\mathsf{MRTI}(\mathcal{R}') \leq \mathsf{MRTI}(\mathcal{R})$ follows from the inclusion $\mathcal{R}' \subseteq \mathcal{R}$. On the other hand, let us show that $\mathsf{MRTI}(\mathcal{R}) \leq \mathsf{MRTI}(\mathcal{R}')$. Suppose that T is an L-tree given by Claim 1, and let S be an optimum solution for $\mathcal{R}'$. Let S' be the tree obtained from T by substituting each leaf labeled by $L' \subseteq L_1$ with the tree $S|L'$. We then have $I(\mathcal{R}, S') = I(\mathcal{R}\backslash rt(T), S') \leq I(\mathcal{R}', S)$, since for a triplet $uv|w \in \mathcal{R}\backslash rt(T)$ we have $u, v, w \in L_1$, and $uv|w$ is present in S if and only it is in S'. We conclude that $\mathsf{MRTI}(\mathcal{R}) \leq \mathsf{MRTI}(\mathcal{R}')$, as claimed. □

Putting all together, we obtain the following result regarding the kernelization of p-Dense RTI.

Theorem 2. p-Dense RTI *has a kernel with* $\leq 2p^2 + 3p$ *labels, that is constructible in* $O(n^4)$ *time.*

Proof. Let $(\mathcal{R}, p)$ be an instance reduced with respect to Rules 1,2 and 3. By Lemma 4, we can reject if the number of labels is greater than $2p^2+3p$. Otherwise, this is a problem kernel for p-Dense RTI with $\leq 2p^2 + 3p$ labels.

We justify that the kernel can be constructed in $O(n^4)$ time. The idea is to maintain: (i) the set $\mathcal{C}$ of st-conflicts, (ii) for each triplet $t \in \mathcal{R}$, the size $n(t)$ of a largest sunflower with center t, (iii) for every $0 \leq i \leq n$, the set $\mathcal{R}_i = \{t \in \mathcal{R} : n(t) = i\}$. This information is collected in $O(n^4)$ time at the beginning of the algorithm. Now, assuming that we have this information, thanks to the sets $\mathcal{R}_i$ we can in $O(n)$ time find a triplet t to which Rule 1 or 2 applies. Then, applying Rule 1 clearly takes constant time. Applying Rule 2 takes $O(n)$ time since after editing the chosen triplet $t = ab|c$, we have: (a) to update the set $\mathcal{C}$ by considering for each $d \in L$, the new st-conflicts which may appear in the set $\{a, b, c, d\}$; (b) for each $d \in L$, for each triplet t with $|L(t) \cap \{a, b, c, d\}| = 3$, to update $n(t)$ and the sets $\mathcal{R}_i$ accordingly. Since each application of Rule 2 decrements p, computing an instance reduced for Rule 2 takes $O(pn)$ time, which is $O(n^4)$ since $p = O(n^3)$. Finally, applying Rule 3 takes $O(n^4)$ time, which is the time required to build L_2 by traversing $\mathcal{C}$. □

5 A Subexponential Fixed-Parameter Algorithm

We now present our subexponential fixed-parameter algorithm (Theorem 3 below). We will need the following terminology. Consider an instance $(\mathcal{R}, p)$ of p-Dense RTI, where $\mathcal{R}$ is a minimally dense triplet set over L. In this section, we call *peacemaker* of $\mathcal{R}$ a set $\mathcal{S} \subseteq \mathcal{R}$ such that $\mathcal{R}\backslash\mathcal{S}$ is consistent. As stated in Section 1, solving p-Dense RTI for $(\mathcal{R}, p)$ amounts to finding a peacemaker of $\mathcal{R}$ of size $\leq p$.

5.1 Solving the Problem on Colored Instances

In the case of triplets, if S is a peacemaker of size $\leq p$, we say that S is *colorful* (for a given coloring of the labels) if no triplet of S is monochromatic. We show how to solve the problem on colored instances, i.e. how to find a minimum colorful peacemaker.

Proposition 2. *Suppose that $\mathcal{R}$ is a minimally dense triplet set on n labels, which is k-colored. Then we can compute a minimum colorful peacemaker using $O(k^3(6n)^k)$ time and $O((2n)^k)$ space.*

Proof. Let $L_1, ..., L_k$ be the color classes. If there exists a colorful peacemaker, then each $\mathcal{R}|L_i$ must be consistent, and represent a tree T_i. Then we have to compute the minimum of $I(\mathcal{R}, T)$ for each tree T over L containing every T_i as a subtree.

A *position* is a tuple $\pi = (\pi[1], ..., \pi[k])$, where each $\pi[i]$ is a node of T_i, or the special value $\perp$. The *label set* of π is $L(\pi) = \cup_i L(\pi[i])$, where $L(\pi[i])$ denotes the set of labels occurring under the node $\pi[i]$. The *root position* is the position $\pi_\top = (r_1, ..., r_k)$, where r_i is the root of T_i. The positions are ordered as follows: $\pi \preceq \pi'$ holds iff for every $i \in [k]$, either $\pi[i] = \perp$, or $\pi[i]$ is a descendant of $\pi'[i]$. We let $\prec$ denote the strict part of $\preceq$. Say that π is *terminal* if each $\pi[i]$ is either a leaf or $\perp$. For a given position π, let $I(\pi)$ denote the minimum of $I(\mathcal{R}, T)$ for each tree T such that $L(T) = L(\pi)$ and $T|L_i = T_i|L(\pi[i])$ (for every $1 \leq i \leq k$). We will compute the values $I(\pi)$ by dynamic programming. At the end of the algorithm, $I(\pi_\top)$ will be the desired value. Clearly, if π is a terminal position then $I(\pi) = 0$.

We now consider nonterminal positions. Let π be a nonterminal position. A *decomposition* of π is a pair of positions (π_1, π_2) such that for each $i \in [k]$,

- if $\pi[i] = \perp$, then $\pi_1[i] = \pi_2[i] = \perp$;
- if $\pi[i]$ is a leaf x, then $\{\pi_1[i], \pi_2[i]\} = \{x, \perp\}$;
- if $\pi|i]$ is an internal node u with children v_1, v_2, then $\{\pi_1[i], \pi_2[i]\}$ is either $\{u, \perp\}$ or $\{v_1, v_2\}$.

Let $D = (\pi_1, \pi_2)$ be a decomposition of π. Say that D is *proper* if π_1, π_2 are distinct from π; observe that in this case, $\pi_1 \prec \pi$ and $\pi_2 \prec \pi$. Consider the set L' formed of the components of the positions π_1, π_2, and define the triplet set $\mathcal{Q}_{\pi,D}$ over L' which contains $uv|w$ whenever there is $a, b \in \{1, 2\}$ distinct, $1 \leq x, y, z \leq k$ such that $\pi_a[x] = u, \pi_a[y] = v, \pi_b[z] = w$. Define the triplet set $\mathcal{R}_{\pi,D}$ over $L(\pi)$ as the set of triplets $xy|z$ such that there exists $uv|w \in \mathcal{Q}_{\pi,D}$ with $x \in L(u), y \in L(v), z \in L(w)$. Define $I(\pi, D) := |\mathcal{R}_{\pi,D} \backslash \mathcal{R}| + I(\pi_1) + I(\pi_2)$. The values $I(\pi)$ can be computed thanks to the following claim.

Claim 2. If π is nonterminal, then $I(\pi)$ is the minimum of $I(\pi, D)$ over each proper decomposition D of π.

This claim yield a dynamic-programming algorithm for computing the values $I(\pi)$, where the positions π are ordered by the relation $\prec$. The runtime analysis is omitted due to space limitations. $\square$

5.2 Colorings of Hypergraphs

For the p-Dense RTI problem, we will need to find proper colorings of an hypothetical peacemaker of size $\leq p$. As in [3], this can be done either probabilistically (by generating random colorings) or deterministically (by explicit constructions of families of colorings). In our case, the correctness of the probabilistic construction depends on a conjecture, but we include it since it holds the promise of smaller hidden constants than the deterministic construction.

We use a generalization of the results by Alon et al. [3] to colorings of r-uniform hypergraphs, obtained by the same authors to solve the Dense Betweenness problem [18]. Let $H = (V, E)$ be a r-uniform hypergraph. Let $c : V \to [k]$ be a coloring of H. An edge $e \in E$ is called *monochromatic under* c if all elements of e have the same color. We say that c is a *proper coloring* of H if H contains no monochromatic edges. Ideally, we would like to generalize a construction of random colorings of [3] from graphs to r-uniform hypergraphs.

Conjecture 1. *Let $H = (V, E)$ be a r-uniform hypergraph, and let $m = |E|$. There exists constants α, β (depending only on r) such that if we randomly color the vertices with $k = (\alpha m)^{1/r}$ colors, then H is properly colored with probability $p = e^{-(\beta m)^{1/r}}$.*

For integers n, m, k, r, a family $\mathcal{F}$ of functions from $[n]$ to $[k]$ is called a *universal* (n, m, k, r)*-coloring family* if for any r-uniform hypergraph H on the set of vertices $[n]$ with at most m edges, there exists an $f \in \mathcal{F}$ which is a proper coloring of H. While we only conjecture the correctness of the random construction, the deterministic construction of [3] can be readily adapted to r-uniform hypergraphs.

Proposition 3 ([18]). *There exists an explicit universal $(\alpha m^2, m, O(m^{1/r}), r)$-coloring family $\mathcal{F}$ of size $|\mathcal{F}| \leq 2^{\tilde{O}(m^{1/r})}$.*

By combining the above results, we obtain the following subexponential fixed-parameter algorithms for the problem.

Theorem 3. *p-Dense RTI can be solved by a deterministic algorithm using $O(n^4 + 2^{O(p^{1/3} \log p)})$ time and $2^{O(p^{1/3} \log p)}$ space. Moreover, assuming Conjecture 1, p-Dense RTI can be solved by a randomized algorithm using $O(n^4 + 2^{O(p^{1/3} \log p)})$ time and $2^{O(p^{1/3} \log p)}$ space.*

Proof. We first run the kernelization algorithm of Section 2 to build in $O(n^4)$ time a kernel $(\mathcal{R}, p')$ with $|L(\mathcal{R})| = O(p^2)$ and $p' \leq p$. The randomized algorithm is obtained by repeating r times the following: (i) choose a random coloring of $\mathcal{R}$ with k colors, (ii) find a minimum colorful peacemaker using the result of Proposition 2, and accept if and only if one execution finds a colorful peacemaker of size $\leq p'$. Conjecture 1 implies that we can choose $r = 2^{O(p^{1/3})}$ and $k = O(p^{1/3})$. The deterministic algorithm is obtained by constructing a family $\mathcal{F}$ as stated in Theorem 3. Then for each $f \in \mathcal{F}$, we find a minimum colorful peacemaker of $\mathcal{R}$ under the coloring f. We accept if and only if only one f yields a peacemaker of size $\leq p'$. Both algorithms applied to the kernel $(\mathcal{R}, p')$ clearly have space and time requirements bounded by $2^{O(p^{1/3} \log p)}$. □

6 Concluding Remarks

In this article, we have obtained a quadratic kernel and a subexponential fixed-parameter algorithm for the p-Dense RTI problem. Can the kernel size be further improved? A subquadratic or even linear kernel would be practically interesting; note that a linear kernel was recently obtained for the FAST problem [7]. Regarding the subexponential fixed-parameter algorithm, is it possible to improve the running time to $2^{O(p^{1/3})}$, as it was done for the FAST problem [11]? Finally, we would like to know whether the $(n-2)$-approximation for RTI from [12] could be improved in the dense case. Analogies with FAST suggest that Dense RTI could admit a constant-factor approximation, and even a PTAS [2,17].

References

1. Aho, A.V., Sagiv, Y., Szymanski, T.G., Ullman, J.D.: Inferring a Tree from Lowest Common Ancestors with an Application to the Optimization of Relational Expressions. SIAM Journal on Computing 10(3), 405–421 (1981)
2. Ailon, N., Charikar, M., Newman, A.: Aggregating inconsistent information: Ranking and clustering. Journal of the ACM 55(5) (2008)
3. Alon, N., Lokshtanov, D., Saurabh, S.: Fast FAST. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009. LNCS, vol. 5555, pp. 49–58. Springer, Heidelberg (2009)
4. Alon, N., Yuster, R., Zwick, U.: Color-Coding. Journal of the ACM 42(4), 844–856 (1995)
5. Bandelt, H.-J., Dress, A.: Reconstructing the shape of a tree from observed dissimilarity data. Advances in Applied Mathematics 7, 309–343 (1986)
6. Berry, V., Nicolas, F.: Maximum agreement and compatible supertrees. Journal of Discrete Algorithms 5(3), 564–591 (2007)
7. Bessy, S., Fomin, F., Gaspers, S., Paul, C., Perez, A., Saurabh, S., Thomassé, S.: Kernels for Feedback Arc Set in Tournaments. In: FSTTCS 2009, pp. 37–47 (2009)
8. Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. Journal of Computer and System Sciences 75(8), 423–434 (2009)
9. Bryant, D.: Building Trees, Hunting for Trees, and Comparing Trees: Theory and Methods in Phylogenetic Analysis. PhD thesis, University of Canterbury, Christchurch, New Zealand (1997)
10. Byrka, J., Guillemot, S., Jansson, J.: New Results on Optimizing Rooted Triplets Consistency. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 484–495. Springer, Heidelberg (2008)
11. Feige, U.: Faster FAST. arXiv.org (November 2009), http://arxiv.org/abs/0911.5094
12. Gąsienec, L., Jansson, J., Lingas, A., Östlin, A.: On the complexity of constructing evolutionary trees. Journal of Combinatorial Optimization 3(2-3), 183–197 (1999)
13. Gramm, J., Niedermeier, R.: A fixed-parameter algorithm for minimum quartet inconsistency. Journal of Computer and System Sciences 67(4), 723–741 (2003)
14. Guillemot, S., Berry, V.: Fixed-parameter tractability of the Maximum Agreement Supertree problem. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 274–285. Springer, Heidelberg (2007)

15. Guo, J.: Problem Kernels for NP-Complete Edge Deletion Problems: Split and Related Graphs. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 915–926. Springer, Heidelberg (2007)
16. Jansson, J., Ng, J.H.K., Sadakane, K., Sung, W.K.: Rooted Maximum Agreement Supertrees. Algorithmica 43(4), 293–307 (2005)
17. Kenyon-Mathieu, C., Schudy, W.: How to rank with few errors. In: STOC 2007, pp. 95–103 (2007)
18. Saurabh, S.: Chromatic coding and universal (hyper-) graph coloring families. Parameterized Complexity News, pp. 49–58 (June 2009), http://mrfellows.net/Newsletters/2009June_FPT_News.pdf
19. Steel, M.: The complexity of reconstructing trees from qualitative characters and subtrees. Journal of Classification 9, 91–116 (1992)

Incremental List Coloring of Graphs, Parameterized by Conservation

Sepp Hartung[⋆] and Rolf Niedermeier

Institut für Informatik, Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{sepp.hartung, rolf.niedermeier}@uni-jena.de

Abstract. Incrementally k-list coloring a graph means that a graph is given by adding stepwise one vertex after another, and for each intermediate step we ask for a vertex coloring such that each vertex has one of the colors specified by its associated list containing some of in total k colors. We introduce the "conservative version" of this problem by adding a further parameter $c \in \mathbb{N}$ specifying the maximum number of vertices to be recolored between two subsequent graphs (differing by one vertex). This "conservation parameter" c models the natural quest for a modest evolution of the coloring in the course of the incremental process instead of performing radical changes. We show that the problem is NP-hard for $k \geq 3$ and $W[1]$-hard when parameterized by c. In contrast, the problem becomes fixed-parameter tractable with respect to the combined parameter (k, c). We prove that the problem has an exponential-size kernel with respect to (k, c) and there is no polynomial-size kernel unless $NP \subseteq coNP/poly$. Finally, we provide empirical findings for the practical relevance of our approach in terms of an effective graph coloring heuristic.

1 Introduction

We study an incremental version of the graph coloring problem:

Incremental Conservative k-List Coloring (IC k-List Coloring)

Input: A graph $G = (V, E)$, a k-list coloring f for $G[V \setminus \{x\}]$ and $c \in \mathbb{N}$.
Question: Is there a k-list coloring f' for G such that $|\{v \in V \setminus \{x\} : f(v) \neq f'(v)\}| \leq c$?

Herein, a function $f : V \to \{1, \ldots, k\}$ is called a *k-coloring* for a graph $G = (V, E)$ when $f(u) \neq f(v)$ for all $\{u, v\} \in E$. For color lists $L(v) \subseteq \{1, \ldots, k\}$, $v \in V$, a k-coloring for G is called a *k-list coloring* when $f(v) \in L(v)$ for all $v \in V$. Occasionally, we also study IC k-Coloring, which is the special case that $L(v) = \{1, \ldots, k\}$ for all $v \in V$.

⋆ Partially supported by the IBM Germany Research & Development GmbH, Böblingen.

© Springer-Verlag Berlin Heidelberg 2010

Intuitively, IC k-List Coloring models that a graph is built by sequentially adding vertices (together with the incident edges). Thereby, referring to the added vertex by x, the task is to efficiently compute a k-list coloring for G from a known k-list coloring of $G[V \setminus \{x\}]$. It can be seen as an incremental version of List Coloring, where one has to find a k-list coloring from scratch. As will turn out, the introduction of the *conservation parameter* c in the above definition helps in making the otherwise hard problem (fixed-parameter) tractable. Notably, conservation has a natural justification from applications where one may rather prefer an "evolution" of the coloring through the incremental process than a "revolution".

Related Work. We start with describing the related incremental clustering problem Incremental Constrained k-Center (IC k-Center). Here, we are given a discrete distance function on objects and a partition of the objects into k clusters such that the maximum distance within each cluster is minimized. Then, after adding one new object, the task is to compute a new "k-clustering" under the constraint that at most c objects of the previous clustering change their cluster. Here the conservation parameter c reflects the fact that in many settings users would not accept a radical change of the clustering, since this may cause a big loss of information acquired in a costly process for the previous clustering. It is easy to see that IC k-Center can be interpreted as a special case of IC k-List Coloring. Moreover, IC k-Center is only one example for the field of constrained clustering [1].

Incremental coloring is also related to the Precoloring Extension problem (PrExt), which is the special case of List Coloring where each color list contains either one or all colors. In other words, the task is to extend a partial k-coloring to the entire graph. It has been shown that on general graphs PrExt is not fixed-parameter tractable with respect to parameter treewidth [9] but, other than List Coloring, it becomes fixed-parameter tractable when parameterized by vertex cover size [12]. Moreover, it is NP-complete for fixed $k \geq 3$ on planar bipartite graphs [14] and $W[1]$-hard with respect to the number of precolored vertices for chordal graphs [15].

Our incremental coloring setting based on the conservation parameter c can also be interpreted as a local search where c measures the degree of locality. Recently, there has been strong interest in analyzing the parameterized complexity of l-local search in terms of some locality parameter l [11, 16]. Our locality measure c can also be seen as "transition cost" between old and new solutions—to keep this cost small has been identified as an important target [18].

A further related field is reoptimization [2]. Here, starting with an optimal solution of an optimization problem, one asks how to compute a solution for a locally modified instance more efficiently by using the known solution for the old instance instead of starting the computation from scratch. Without adding the conservation parameter the reoptimization of coloring problems remains hard.

Our Results. We initiate a study of IC k-List Coloring in terms of parameterized complexity [8, 13, 17], considering the two parameters k (number of colors) and c (number of recolored vertices). We show that the problem is NP-hard for fixed $k \geq 3$ and $W[1]$-hard when parameterized by c. In contrast, it becomes fixed-parameter tractable with respect to the combined parameter (k, c). We show that IC k-List Coloring has a $3(k-1)^c$-vertex problem kernel and that there is no hope for a polynomial-size problem kernel with respect to (k, c). Finally, we provide first empirical evidence for the practical relevance of "parameterizing by conservation" by demonstrating how our algorithms can be successfully employed as subroutines in a very effective (local search) heuristic for graph coloring.

Preliminaries. For a graph $G = (V, E)$, we set $V(G) := V$ and $E(G) := E$. Analogously, for a path $P = [v_1, \ldots, v_j]$, we write $V(P) := \{v_1, \ldots, v_j\}$ and $E(P) := \{(v_i, v_{i+1}) : 1 \leq i < j\}$ for all directed edges on P. For a graph $G = (V, E)$ and a vertex set $S \subseteq V$, we write $G[S]$ to denote the graph induced by S in G, that is, $G[S] := (S, \{e \in E : e \subseteq S\})$. We define the (open) neighborhood of a vertex v by $N(v) := \{u \in V : \{u, v\} \in E\}$. Moreover, for a given k-coloring f of G and a color i, we set $N(v, i) := \{u \in V : f(u) = i \wedge \{u, v\} \in E\}$.

A parameterized problem is called *fixed-parameter tractable* if it can be solved in $f(k) \cdot n^{O(1)}$ time, where f is a computable function depending only on the parameter k, not on the input size n [8, 13, 17]. A *parameterized reduction* from a language L to another parameterized language L' 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 $(x, k) \in L \leftrightarrow (x', k') \in L'$. Based on that, [8] established a hierarchy of complexity classes (basic class $W[1]$) in order to classify (likely) fixed-parameter intractable problems.

Problem kernelization is the reduction of an instance I with parameter k by data reduction rules to a smaller instance I' with parameter $k' \leq k$ in polynomial time such that (I, k) is a yes-instance if and only if (I', k') is a yes-instance [8, 17]. If the size of I' is bounded by a (polynomial) function in k, then the instance (I', k') is called a (polynomial) kernel.

2 Parameterized Complexity

In this section, we first study the parameterized complexity of IC k-List Coloring with respect to the single parameters k (number of colors) and c (number of recolored vertices). Since we encounter computational hardness with respect to these single parameterizations, we proceed with a simple search tree strategy showing that IC k-List Coloring is fixed-parameter tractable with respect to the combined parameter (k, c). Since k-Coloring is NP-complete for $k \geq 3$, the following may come without surprise.

Theorem 1. IC k-List Coloring *for fixed* $k \geq 3$ *is* NP*-complete.*

Proof. Containment in NP is obvious. We reduce k-Coloring to IC k-List Coloring as follows. Let $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ be an instance of k-Coloring. We set $G_i := G[\{v_1, \ldots, v_i\}]$ for $i \leq n$. To decide the k-colorability of G, we proceed inductively: Obviously, if any G_i, $1 \leq i \leq n$, is not k-colorable, then G also is not. Note that $G_n = G$. Assume that G_{i-1} is k-colorable. For each vertex of G, we fix its color list to be $\{1, \ldots, k\}$. Moreover, we choose $c := |V|$. Clearly, G_i is k-colorable iff G_i can be incrementally colored by recoloring at most c vertices in a k-coloring for G_{i-1}. Thus, we can decide the question about the k-colorability of G inductively by deciding at most n recoloring problems, implying the NP-hardness of IC k-List Coloring for $k \geq 3$. □

Using the above proof strategy, the result that k-List Coloring is $W[1]$-hard with respect to the parameter treewidth [9] can be transferred to IC k-List Coloring. Moreover, it also follows that IC k-List Coloring is NP-complete for fixed $k \geq 3$ for all hereditary graph classes[1] where the ordinary List Coloring problem is NP-hard for fixed $k \geq 3$, e. g., planar bipartite and chordal graphs. The proof strategy is also used in Section 4 to devise an empirically effective heuristic for graph coloring.

We proceed by considering the parameterized complexity of IC k-List Coloring with respect to the parameter c (for unbounded k). In contrast to the parameter k, when c is a constant, then IC k-List Coloring clearly becomes polynomial-time solvable. However, it is $W[1]$-hard, again excluding hope for fixed-parameter tractability.

In order to show the $W[1]$-hardness with respect to c, we present a parameterized reduction from the $W[1]$-complete k-Multicolored Independent Set problem [9, 10]. The problem is to decide for a given k-coloring f for a graph $G = (V, E)$ whether there exists a *multicolored k-independent set*, that is, a vertex subset $S \subseteq V$ with $|S| = k$ such that $\forall u, v \in S : \{u, v\} \notin E \wedge f(u) \neq f(v)$.

Theorem 2. IC k-List Coloring *is* $W[1]$*-hard with respect to the parameter* c.

Proof. Let $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ be a k-colored graph (through a coloring f), taken as an instance for k-Multicolored Independent Set. We construct a graph $G' = (V', E')$ from G such that by choosing $c := 2k$ the instance (G', f', L, c) is a yes-instance of IC k-List Coloring iff G is a yes-instance of k-Multicolored Independent Set. Herein, f' is an $(n+1)$-coloring of $G'[V' \backslash \{x\}]$ and L stands for the color lists $L(v)$, $v \in V'$. Note that x is the vertex added in the incremental process. We add $k+1$ new vertices to V, setting $V' := V \cup \{x, s_1, \ldots, s_k\}$. For $v_i \in V$, set $f'(v_i) := i$ in G'. Moreover, the color list for v_i is $L(v_i) := \{i, n+1\}$. To complete the construction, it remains to define f' for the vertices from $\{x, s_1, \ldots, s_k\}$, the edge set E' with $E \subseteq E'$, and the color lists for x and all s_i. To this end, note that each vertex s_i one-to-one corresponds to the subset of vertices from V colored i. Set $f'(s_i) := n+1$ for $1 \leq i \leq k$ and $L(s_i) := \{f'(v) : v \in V_i\} \cup \{n+1\}$, where $V_i := \{v \in V : f(v) = i\}$. Note that $L(s_i) \cap L(s_j) = \{n+1\}$ for $i \neq j$. Moreover, we add edges between s_i and all vertices from V_i. Finally, we set $f'(x) := n+1$, $L(x) := \{n+1\}$, and make x adjacent to all vertices from $\{s_i, \ldots, s_k\}$. This completes the construction.

[1] A graph class is hereditary if it is closed under taking induced subgraphs.

The idea behind the construction of G' is that those vertices in $V \subset V'$ which can be recolored to the color $n+1$ one-to-one correspond to the vertices in a multicolored k-independent set. We omit the correctness proof. $\square$

The results presented so far are negative in terms of fixed-parameter tractability, motivating the study of the *combined* parameter (k, c). A simple search strategy leads to fixed-parameter tractability in this case.

Theorem 3. IC k-List Coloring *can be solved in* $\mathcal{O}(k \cdot (k-1)^c \cdot |V|)$ *time, that is, it is fixed-parameter tractable with respect to the combined parameter* (k, c)*.*

Proof. Clearly, if for the inserted vertex x it holds that $\{f(v) : v \in N(x)\} \subset L(x)$, then there is a "free color" left for x and x can be colored using this free color. Otherwise, $\{f(v) : v \in N(x)\} \supseteq L(x)$. Hence, a recoloring is necessary. First, branch into the $|L(x)| \leq k$ possibilities how to color x. In each branch, at least one of the neighbors of x has the same color as x and hence needs to be recolored. Now, we have at most $k-1$ options to do so. This process continues until all "color conflicts" have disappeared or in total c vertex recolorings have been performed without obtaining a k-coloring. It is easy to see that this strategy leads to a search tree of size $\mathcal{O}((k-1)^c)$ (depth c and branching factor $k-1$). From this, the running time $\mathcal{O}(k \cdot (k-1)^c \cdot |V|)$ follows. $\square$

Note that all results above also hold for IC k-Coloring; we have a stronger focus on IC k-List Coloring since this more general problem allows for an elegant formulation of data reduction rules.

3 Data Reduction and Kernelization

By developing a polynomial-time executable *data reduction rule*, next we describe how to the transform an instance of IC k-List Coloring into an equivalent but size-reduced instance, known as problem kernel in parameterized algorithmics.

An Exponential-Size Kernel. Assume that the graph $G = (V, E)$, $c \in \mathbb{N}$, and the k-list coloring f for $G[V \setminus \{x\}]$ form a yes-instance for IC k-List Coloring. Furthermore, let the k-list coloring f' for G be a *recoloring*, that is the number of elements in the corresponding *recoloring set* $S := \{v \in V \setminus \{x\} : f'(v) \neq f(v)\} \cup \{x\}$ is at most $c+1$. Intuitively speaking, the set S contains all recolored vertices (including x).

Our kernelization approach makes use of the following observations. If there exists a connected component Z in $G[S]$ such that $x \notin Z$, then one can simply remove Z by setting $f'(v) = f(v)$ for all $v \in Z$, obtaining a smaller recoloring set. Hence, we can assume without loss of generality that for every vertex $v \in S$ there exists a path from x to v in $G[S]$. Actually, there must exist a so-called *conflict path* for every vertex, that is, a simple path from x to v with the special property that for every edge (u, w) on the path $f'(u) = f(w)$. The non-existence of a conflict path for a vertex $v \in S$ implies $f'(u) \neq f(v)$ for all $u \in N(v)$, thus, we can again remove v, setting $f'(v) = f(v)$. Therefore, one can view the recoloring of u as the reason why w must also be recolored. The following lemma summarizes the above observations.

Lemma 1. *The graph $G[S]$ contains a conflict path of length $\leq c$ for every vertex in S.*

In order to describe the idea behind our data reduction rule, assume that $v \in S$ and denote the corresponding conflict path in $G[S]$ by P_v. Clearly, if $V(P_v)$ denotes the set of all vertices on P_v, then $V(P_v) \subseteq S$. Consider an arbitrary edge (u, w) on P_v. Since $f'(u) = f(w)$, it follows that $N(u, f(w)) \subseteq S$. By summing up these vertices, this can be viewed as the *costs* (#recolored vertices) of a conflict path P_v. Utilizing this idea, our data reduction rule computes for some vertex a *possible conflict path* of minimum cost and removes the vertex when the costs are greater than the conservation parameter c.

Removing a vertex v means to remove v and all its incident edges and to delete the color $f(v)$ in the color list of all neighbors. This makes sure that we can reinsert v with color $f(v)$ in any solution for the kernel. Actually, the possibility to conveniently remove and reinsert a vertex is the main reason why we work with *list* coloring.

As the name suggests, a possible conflict path for a vertex v is a path which could become a conflict path for v when $v \in S$. Therefore, a possible conflict path P_v is a simple path such that $f(u) \in L(w)$ for each edge $(u, w) \in E(P_v)$. Next, a cost function $l : V \to \mathbb{N}$ gives for each vertex a lower bound for the number of vertices that need to be recolored when reaching the vertex on a possible conflict path. Using this, we now formulate our data reduction rule; its correctness can be directly inferred from Lemma 1.

Reduction Rule 1. *If there is a vertex v with $l(v) > c$, then remove v.*

We define our *cost function l* in an iterative manner. We start with an empty set M and initialize $l(x) := 0$ and $l(v) := \infty$ for all $v \in V \backslash \{x\}$. The set M contains the vertices for which the cost function is already computed. Next, we choose a vertex $v \in V \backslash M$ with minimum cost function value and add it to M (in the first step we add x). Next, consider a neighbor u of v with $f(u) \in L(v)$. When v is the ancestor of u on a cheapest possible conflict path for u, then the cost function value $l(u)$ is the sum of $l(v)$ and $|N(v, f(u)) \setminus M|$. Thus, we update the cost function value by setting $l(u) := \min\{l(u), l(v) + |N(v, f(u)) \setminus M|\}$. This process will be continued while $M \neq V$. Furthermore, with the help of a priority queue, the cost function l can be computed in $\mathcal{O}(|V| \log |V| + |E|)$ time.

Using the cost function described above, Rule 1 leads to the following.

Theorem 4. IC k-List Coloring *admits a $(3 \cdot (k-1)^c)$-vertex kernel, which can be computed in $\mathcal{O}(|V| \log |V| + |E|)$ time.*

Proof. We show that the exhaustive application of Rule 1 leads to the asserted kernel. In order to bound the size of a reduced graph G, at first we construct a worst-case graph T. Next, we prove that the size of T is bounded by the asserted kernel size. We complete our proof by showing that $|V(G)| \leq |V(T)|$.

The graph T is a tree in which the distance of all leaves to the root is exactly c. We set $L(v) := \{1, \ldots, k\}$ for all $v \in V(T)$. In addition, the root has one child of each color and all other inner vertices have one child of each color except their own color. The cost function l_T assigns each vertex its distance to the root. Thus, the instance (T, c, k) for IC k-List Coloring is reduced with respect to Rule 1.

For a reduced graph and its corresponding cost function, for instance, T and l_T, we define a partition of $V(T)$ by $V_T^j := \{v \in V(T) : l_T(v) = j\}$ for $0 \leq j \leq c$. By construction, it follows that $V_T^0 = \{x\}$ and $|V_T^j| = k \cdot (k-1)^{j-1}$. Thus, the overall size bound for $V(T)$ is as follows:

$$|V(T)| = 1 + k \cdot \sum_{j=1}^{c} (k-1)^{j-1} = 1 + k \cdot \left(\frac{1-(k-1)^c}{2-k} \right)$$
$$= 1 + \frac{k}{k-2}((k-1)^c - 1) \leq 3 \cdot (k-1)^c.$$

Next, we prove the kernel size for the reduced graph G. Let l_G be the corresponding cost function and consider the partition V_G^j for $0 \leq j \leq c$ of $V(G)$. In the following, we will show that $|V_G^j| \leq |V_T^j|$, implying $|V(G)| \leq |V(T)|$.

Consider a vertex $v \in V_G^j$ for some $1 \leq j \leq c$ and a cheapest possible conflict path $P_v = [x, \ldots, w, v]$ for v. Because w is the ancestor of v on P_v, it holds that $l_G(w) < l_G(v)$. Suppose that $w \in V_G^{j-t}$ for some $1 \leq t \leq j$. Then, by definition it follows that $l_G(v) - l_G(w) = t$. Thus, there exist at most t vertices with color $f(v)$ in V_G^j, whose ancestor on a cheapest conflict path is w. Formally, defining the ancestor function by $p(v) := w$ and $p^{-1}(w) := \{v \in V_G^j : p(v) = w\}$ we can infer that $|p^{-1}(w) \cap N(w, f(v))| \leq t$. Considering all colors, it follows that $|p^{-1}(w)| \leq t \cdot (k-1)$.

To show $|V_G^j| \leq |V_T^j|$, next, by removing all vertices in $p^{-1}(w)$ from G we construct a graph $\tilde{G}$. Then, we insert tree T_w with w as root into $\tilde{G}$. Similarly to the structure of T, every inner node of T_w has exactly one child of each color and all leaves of T_w have distance exactly t to the root w. Thus, $V_{\tilde{G}}^j$ contains all $(k-1)^t$ leaves of T_w. Assuming $k \geq 3$, it follows that $(k-1)^t \geq t \cdot (k-1) \geq |p^{-1}(w)|$ and, by this, we can infer that $|V_G^j| \leq |V_{\tilde{G}}^j|$. This process can be executed for all ancestors w on a cheapest possible conflict path for each $v \in V_G^j$. Since the structure of T_w is similar to that of T, we obtain that $|V_G^j| \leq |V_{\tilde{G}}^j| \leq |V_T^j|$. □

Non-Existence of a Polynomial Kernel. [4] showed that a *compositional* parameterized problem (whose unparameterized formulation is NP-complete) does not have a polynomial kernel, unless $NP \subseteq coNP/poly$. A parameterized problem is compositional if there exists an algorithm which receives as input a sequence of instances $(I_1, c), \ldots, (I_r, c)$ and outputs an instance (I, c') such that (I, c') is a yes-instance iff (I_j, c) is a yes-instance for some $1 \leq j \leq r$. Furthermore, the running time of the algorithm has to be bounded by a polynomial in $\sum_{j=1}^{r} |I_j| + c$ and $c' \leq \text{poly}(c)$.

To prove the non-existence of a polynomial kernel, we first show that IC 3-Coloring is compositional.

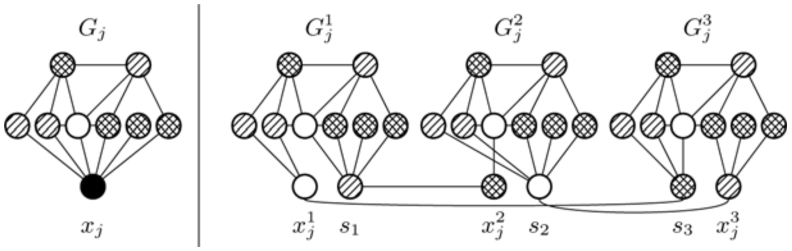


Fig. 1. Example for the composition algorithm for IC 3-Coloring. The left part shows a graph G_j of an input instance. The right part shows the three duplications of G_j and the modifications on them. To keep the figure simple, the neighbors of the constant vertices, which prevent their recoloring, are omitted.

Theorem 5. IC 3-Coloring *has no polynomial kernel, unless NP $\subseteq$ coNP/poly.*

Proof. Suppose that we are given a sequence of instances $(G_1, x_1, c), \ldots, (G_r, x_r, c)$ of the IC 3-Coloring problem. Our composition algorithm makes a case distinction depending on the size of r relative to c.

Case 1 ($3r > 2^c$): We use the search tree algorithm introduced in Section 2 and simply solve all instances. Depending on whether we found a yes-instance or not, we construct a trivial yes- or no-instance as a result of our composition algorithm. We need $\mathcal{O}(2^c \cdot |G_j|)$ time per instance, hence, the overall running time is bounded by $\mathcal{O}(r \cdot 2^c \cdot \max_{1 \le j \le r} |G_j|)$ or $\mathcal{O}(r^2 \cdot \max_{1 \le j \le r} |G_j|)$.

Case 2 ($3r \le 2^c$): In this more complicated case, we construct a new graph G by connecting the graphs $G_1, \ldots, G_r$ by a binary tree. The rough idea is as follows. Starting at the root x of the binary tree, it is then possible to traverse the tree on a conflict path for reaching any subgraph G_j. Then, every recoloring of G_j for some $1 \le j \le r$ is a recoloring for G, and vice versa. We first describe the construction of G in detail and then prove its correctness.

Starting with an empty graph G, we first insert a *constant vertex* s_i of color i for each color $i \in \{1, 2, 3\}$. By adding $c' + 1$ neighbors of each color (except for s_i's color), we make sure that the color of a constant vertex will never change.

In order to insert the graph G_j for some $1 \le j \le r$ into the graph G, we have to assign a 3-coloring to G_j. Therefore, preserving the color of the vertices, we duplicate the graph G_j three times (referred to as G_j^1, G_j^2, G_j^3). We adjust the graphs for all $i \in \{1, 2, 3\}$ as follows: Using the permutation $\pi = (1, 2, 3)$ (cycle notation), we set the color of x_j^i in G_j^i to $\pi(i)$. Furthermore, we remove all edges in G_j^i between the vertex x_j^i and every vertex in $N_\pi(x_j^i) := N(x_j^i, \pi(i)) \cup N(x_j^i, \pi^2(i))$. After this, G_j^i is correctly colored. To prevent a recoloring of the vertices in $N_\pi(x_j^i)$ with color i, we insert an edge from the constant vertex s_i to each vertex in $N_\pi(x_j^i)$. Adding the edge $\{x_j^i, s_{\pi^2(i)}\}$ then completes the construction of G_j^i. Figure 1 shows an example.

Next, for all colors $1 \le i \le 3$ and $1 \le j \le r$, we insert the graphs $\{G_j^i\}$ into G and connect the vertices $\{x_j^i\}$ through a balanced binary tree. The first property of the tree is that every inner vertex connects two differently colored vertices and we set the third color to it. Second, balanced means that each vertex x_j^i has the same distance to the root. Both properties can be fulfilled by "filling up" the tree with constant vertices.

Finally, we prove the correctness of our composition algorithm. The algorithm outputs the instance (G, x, c') with $c' := c + \lceil \log(3r) \rceil$. The construction of G can be done in polynomial time and since $3r \leq 2^c$, it follows that $c' \in \mathcal{O}(c^2)$.

In order to show equivalence, the first property of the binary tree implies that (starting at the root x) each inner vertex serves as a "switch" between on which of its both children a conflict path can continue. Hence, by a path through the tree we can reach each vertex x_j^i. Note that, because of the second property, such a conflict path requires $\lceil \log(3r) \rceil$ recoloring operations.

Consider the situation where a conflict path through the tree reaches x_j^i. Recalling that x_j^i has color $\pi(i)$ and G contains the edge $\{x_j^i, s_{\pi^2(i)}\}$, one has to recolor x_j^i with color i. Now, when G_j can be recolored by at most c changes such that x_j obtains color i, then this recoloring is also a proper recoloring for G_j^i. The reverse direction also holds, because we excluded the possibility of recoloring the vertices in $N_\pi(x_j^i)$ with color i. □

According to the framework of Bodlaender et al. [3, 4], there are two ways to show that a problem does not admit a polynomial kernel. First, as we did for IC 3-Coloring, one can show that the problem is compositional. Second, one can reduce by a *polynomial parameter transformation* a problem for which the non-existence of a polynomial kernel is already known to the problem in question. Adding the color list $L(v) = \{1, 2, 3\}$ for each vertex v is a simple polynomial parameter transformation from IC 3-Coloring to IC k-List Coloring. The generalization of this idea leads to the following.

Corollary 1. IC k-List Coloring *has no polynomial kernel, unless $NP \subseteq coNP/poly$.*

We close the section with the following strengthening of Theorem 5, showing that there is also no hope to find a polynomial kernel even when we restrict IC k-List Coloring to planar bipartite or chordal graphs. The correctness follows from the fact that the composition algorithm for IC 3-Coloring (proof of Theorem 5) composes the sequence of instances by a binary tree.

Corollary 2. IC k-List Coloring *for fixed $k \geq 3$ restricted to planar bipartite or chordal graphs has no polynomial kernel, unless $NP \subseteq coNP/poly$.*

4 Implementation and Experiments

To demonstrate the practical relevance and usefulness of IC k-List Coloring, we have implemented our search tree algorithm (see Section 2) as a subroutine of a popular greedy algorithm for the graph coloring problem. We will show that the derived algorithm outperforms an often used heuristic algorithm called

Iterated Greedy [5] in terms of quality and running time. Furthermore, we provide practical evidence that in this graph coloring approach the conservation parameter c can often be set to pretty small values, typically smaller than k (number of colors).

Graph Coloring Instances. We performed our tests on a collection of graph coloring instances, previously used in the "Graph Coloring and its Generalizations"-Symposium (2002) [6]. Before that, some of these graph coloring instances were studied in the *DIMACS Implementation Challenge* (1993) [7].

Altogether, the collection of instances contains 64 graphs, where the number of vertices ranges from 25 to 4730 (avg: 1067). The average density of the graphs (ratio of the number of vertices to the number of edges) is 15%. The instances cover a wide range of graph classes such as *Leighton and latin square graphs.*

Implementation Details. The implementation of all algorithms is written in Java, it is open source and freely available.[2] All experiments were run on an AMD Athlon 64 3700+ machine with 2.2 GHz, 1 M L2 cache, and 3 GB main memory running under the Debian GNU/Linux 5.0 operating system with Java 1.6.0_12 (option: -Xmx256M).

Next, we describe some details of the implementation of the graph coloring algorithms. Our algorithm is based on the following greedy strategy. Processing the vertices in descending order according to their degree, color a vertex with an already used color whenever possible, otherwise use a new color for the vertex. There exist many strategies how to choose a color from all possible already used colors. We implemented the strategies *Simple* (choose the first color according to any ordering), *Largest First* (choose the color which is used most often) and *Random* (random color). For all our results we ran the algorithm with all strategies and list the best result that was found during these trials.

The greedy algorithm suffers from the fact that the color of an already colored vertex cannot be revoked. This is the point where our search tree algorithm comes into play. Consider the situation where the greedy algorithm "fails", that is, during the coloring of a graph $G = (V, E)$ with $V := \{v_1, \dots, v_n\}$ a vertex v_i for some $1 < i \leq n$ cannot be colored with the already used colors $\{1, \dots, k\}$, since v_i has at least one neighbor of each color in $G_i := G[v_1, \dots, v_i]$. Instead of adding a new color $k+1$ for vertex v_i, we try to solve an IC k-List Coloring instance on the graph G_i with v_i as the uncolored vertex x by our search tree algorithm to get a k-coloring for G_i. If our search tree algorithm cannot find a k-coloring for G_i, then we color the vertex v_i with the new color $k+1$.

Our search tree implementation incorporates the idea, also used in our data reduction rule (Rule 1), to check after each recoloring whether the number of conflicts is at most c (conservation parameter). Using our cost function (see Section 3), the fact that the cost of a cheapest possible conflict path is greater than c implies that the number of conflicts is greater than c. Thus, our search tree algorithm will never recolor a vertex which would be reduced by Rule 1.

[2] Source code & a full list of results: `http://theinf1.informatik.uni-jena.de/inc_cluster/`

Table 1. Summary of our experiments. The first column k for each algorithm denotes the best number of colors which was found and the last column provides the running time in seconds. Each value was obtained as the average over four runs (standard deviation in brackets). For the Iterated Greedy algorithm *#iter* denotes the number of iterations. For our search tree based algorithm, c denotes the conservation parameter.

name	greedy		Iterated Greedy			search tree		
	k	time (s)	k	#iter	time (s)	k	c	time (s)
ash608GPIA	8	0.2	5.0 [0.0]	1058.8	33.2 [1.3]	5.0 [0.0]	8	0.2 [0.0]
DSJC1000.1	29	0.1	27.5 [0.6]	1243.8	24.5 [5.0]	25.0 [0.0]	6	0.5 [0.1]
DSJC500.1	17	0.0	16.8 [0.5]	1217.5	6.7 [2.3]	14.8 [0.5]	8	0.3 [0.1]
latin_square_10	148	0.3	109 [1.4]	2765.3	68.8 [9.2]	116.8 [2.6]	4	1.5 [0.6]
le450_15a	18	0.0	18.0 [0.0]	1000	5.0 [0.0]	16.0 [0.0]	7	1.2 [0.2]
qg.order40	44	0.3	42.0 [0.0]	1033.8	59.9 [1.7]	41.0 [0.0]	5	4.8 [0.3]
queen16_16	26	0.0	20.3 [0.5]	1295	2.2 [0.7]	19.3 [0.5]	7	0.4 [0.2]
school1_nsh	31	0.0	14.0 [0.0]	1443.8	3.7 [0.3]	23.0 [5.4]	6	0.2 [0.1]
wap03	55	4.0	53.8 [0.5]	1006.3	475.5 [3.4]	50.0 [0.0]	5	3.9 [0.3]

The potential to find a k-coloring for G_i (if possible) depends on the choice of the value for the conservation parameter c. On the one hand, the higher the value, the higher the potential; on the other hand, the value of c makes the "major contribution" to our algorithm's running time. Based on preliminary experiments, we choose $c \leq 8$ maximal under the constraint $k \cdot (k-1)^c \leq 10^{10}$.

We compare our above described algorithm to the *Iterated Greedy Algorithm* [5], which is also based on the described greedy algorithm. Its main idea is to iteratively run the greedy algorithm (mixing the described strategies how to choose a possible color). Thereby, the algorithm makes use of the fact that if the greedy algorithm processes the vertices with respect to an already known k-coloring, it will not produce a worse coloring. Clearly, the hope is, while using a "smart permutation" of the vertices, to get a better coloring (in terms of number of colors). We implemented Iterated Greedy in basically the same manner as proposed in [5], meaning that we adopt the strategies how generating "smart permutations" and aborting the iteration when 1000 times no better coloring was found.

Results. Our experimental findings are as follows. Table 1 contains the results for some important instances. Our algorithm (called *search tree* in Table 1) finds for 89% and Iterated Greedy for 83% of the instances a better coloring than the naive greedy algorithm. Thereby, our algorithm could decrease the number of colors by about 12% and Iterated Greedy by about 11%. Furthermore, our algorithm is by a factor of 50 and Iterated Greedy is by a factor of 170 slower than the greedy algorithm. In other words, the greedy algorithm needs 23 seconds, our algorithm needs 19 minutes and Iterated Greedy needs 1 hour 5 minutes for coloring all instances.

In summary, in most cases our algorithm is clearly superior to the Iterated Greedy algorithm, both in terms of number of used colors and running time. We emphasize that the potential of our algorithm is bounded by having chosen an upper bound of 8 for the conservation parameter c.

5 Conclusion

We believe that the incremental setting combined with the parameterization by conservation provides a fruitful approach to numerous other optimization problems besides coloring, e. g. clustering problems such as Incremental Constrained k-Center. There remain numerous challenges for future research even when restricting the focus to coloring problems. Among others, for IC k-List Coloring it is open to achieve a problem kernel of $\mathcal{O}(c^k)$ vertices contrasting our $\mathcal{O}(k^c)$-vertex kernel. Improving on the upper bounds of our simple search tree algorithm is desirable. By means of a reduction from List Coloring we have shown that IC k-List Coloring is also NP-hard for restricted graph classes such as planar bipartite or chordal graphs. However, it would be interesting to study whether improvements in terms of parameterized algorithms and data reduction rules are achievable for such restricted graph classes.

Acknowledgements. We thank M. Wurst (IBM Germany) for helpful discussions. Furthermore, we are grateful to J. Uhlmann and M. Weller for their comments.

References

[1] Basu, S., Davidson, I., Wagstaff, K.: Constrained Clustering: Advances in Algorithms, Theory, and Applications. Chapman & Hall, Boca Raton (2008)

[2] 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)

[3] Bodlaender, H.L., Thomasse, S., Yeo, A.: Analysis of data reduction: Transformations give evidence for non-existence of polynomial kernels. Technical report, UU-CS-2008-030, Institute of Information and Computing Sciences, Utrecht University, Netherlands (2008)

[4] Bodlaender, H.L., Downey, R.G., Fellows, M.R., Hermelin, D.: On problems without polynomial kernels. J. Comput. System Sci. 75(8), 423–434 (2009)

[5] Culberson, J.C., Luo, F.: Exploring the k-colorable landscape with iterated greedy. DIMACS Series in Discrete Math. and Theor. Comput. Sci., pp. 245–284 (1996)

[6] DIMACS. Graph coloring and its generalizations (2002), `http://mat.gsia.cmu.edu/COLOR02` (Accessed, December 2009)

[7] DIMACS. Maximum clique, graph coloring, and satisfiability. Second DIMACS implementation challenge (1995), `http://dimacs.rutgers.edu/Challenges/` (Accessed December 2009)

[8] Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)

[9] Fellows, M.R., Fomin, F.V., Lokshtanov, D., Rosamond, F.A., Saurabh, S., Szeider, S., Thomassen, C.: On the complexity of some colorful problems parameterized by treewidth. In: Dress, A.W.M., Xu, Y., Zhu, B. (eds.) COCOA. LNCS, vol. 4616, pp. 366–377. Springer, Heidelberg (2007)

[10] Fellows, M.R., Hermelin, D., Rosamond, F.A., Vialette, S.: On the parameterized complexity of multiple-interval graph problems. Theor. Comput. Sci. 410(1), 53–61 (2009a)

[11] Fellows, M.R., Rosamond, F.A., Fomin, F.V., Lokshtanov, D., Saurabh, S., Villanger, Y.: Local search: Is brute-force avoidable? In: Proc. 21st IJCAI, pp. 486–491 (2009b)
[12] Fiala, J., Golovach, P.A., Kratochvíl, J.: Parameterized complexity of coloring problems: Treewidth versus vertex cover. In: Chen, J., Cooper, S.B. (eds.) TAMC 2009. LNCS, vol. 5532, pp. 221–230. Springer, Heidelberg (2009)
[13] Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
[14] Kratochvíl, J.: Precoloring extension with fixed color bound. Acta Math. Uni. Comenianae 62(2), 139–153 (1993)
[15] Marx, D.: Parameterized coloring problems on chordal graphs. Theor. Comput. Sci. 351(3), 407–424 (2006)
[16] Marx, D.: Searching the k-change neighborhood for TSP is W[1]-hard. Oper. Res. Lett. 36(1), 31–36 (2008)
[17] Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
[18] Shachnai, H., Tamir, G., Tamir, T.: Minimal cost reconfiguration of data placement in storage area network. In: Bampis, P. E. (ed.) WAOA 2009. LNCS, vol. 5893, pp. 229–241. Springer, Heidelberg (2010)

Schnyder Greedy Routing Algorithm

Xin He[1,⋆] and Huaming Zhang[2,⋆⋆]

[1] Department of Computer Science and Engineering
SUNY at Buffalo, Buffalo, NY, 14260, USA
xinhe@buffalo.edu

[2] Computer Science Department
University of Alabama in Huntsville, Huntsville, AL, 35899, USA
hzhang@cs.uah.edu

Abstract. Geometric routing by using virtual locations is an elegant way for solving network routing problem. In its simplest form, greedy routing, a message is forwarded to a neighbor that is closer to the destination. One major drawback of this approach is that the virtual coordinates requires $\Omega(n \log n)$ bits to represent, which makes this scheme infeasible in some applications.

In this paper, we introduce a modified version of greedy routing which we call *generalized greedy routing algorithm*. Instead of relying on decreasing distance for routing decision, our routing algorithms use other criterion to determine routing path, solely based on local information. We present simple generalized greedy routing algorithms based on Schnyder coordinates (consisting of three integers between 0 and $2n$), which are derived from Schnyder realizer for plane triangulations and Schnyder wood for 3-connected plane graphs. The algorithms are simple and can be easily implemented in linear time.

1 Introduction

Routing certainly is one of the most important algorithmic problems in networking. Extensive research has been devoted to discover efficient routing algorithms. (For example, see [5,21]). Routing was previously done via routing protocols. Such solutions are space inefficient and require considerable setup overhead, which makes it infeasible for some networks such as wireless sensor networks.

Recently, a new approach *geometric routing* has been proposed. This approach uses geometric coordinates of the vertices to compute the routing paths. *Greedy routing* is one of the simplest geometric routing, in which a vertex simply forwards messages to a neighbor that is closer to the destination.

Greedy routing is simple, but also has some problems. First, GPS devices used to determine geometric coordinates are expensive and increase the energy consumption. This should be avoided, esp. for sensor networks. More importantly, bad geographical locations of network nodes can lead to situations in which the

⋆ Research supported in part by NSF Grant CCR-0635104.

⋆⋆ Research supported in part by NSF Grant CCR-0728830.

© Springer-Verlag Berlin Heidelberg 2010

routing fails because a *void position* has been reached. Namely, a packet has reached a vertex all whose neighbors are no closer from the destination than the vertex itself.

An elegant solution was proposed in [18] to solve these problems: For geometric routing of a graph G, we could ignore its pre-defined geometric coordinates (e.g., GPS coordinates) Instead, we could use graph drawing, based on the structure of G, to compute *virtual coordinates* for the vertices of G. Geometric routing algorithms then rely on such virtual coordinates rather than on the real geographic ones to compute routes. *Greedy drawing* is introduced as a solution for greedy routing. Simply speaking, a greedy drawing is a drawing in which greedy routing works.

Intense research has been done on greedy routing schemes that assign network nodes to virtual coordinates in a natural metric space. Papadimitriou and Ratajczak [17] showed that any 3-connected planar graph can be embedded in R^3 that supports greedy geometric routing. However, a non-standard metric function is used in [17]. They conjectured that such embeddings are possible in R^2. This conjecture has drawn a lot of interests [2,4,6,13,14,15,16]. Greedy embeddings in R^2 were first obtained only for graphs containing power diagrams [4], then for graphs that contain Delaunay triangulations [15], and then existentially (but not algorithmically) for plane triangulations [6]. Leighton and Moitra [14] proved this conjecture positively by designing a greedy embedding algorithm for any 3-connected planar graph in R^2. A similar result was independently found in [2]. However, neither of the two papers give the time efficiency analysis of their algorithms.

Eppstein and Goodrich in [8] pointed out another problem of greedy routing: They are still not yet practically feasible because, in the worst case, the virtual coordinates, produced by greedy drawing and used in greedy routing, require $\Omega(n \log n)$ bits to represent them. Hence, these greedy drawing based routing algorithms have the same space usage as traditional routing table approaches. The main obstacle for the applicability of greedy routing is not only the existence of a greedy drawing, but also the existence of a *succinct greedy drawing*, in which the virtual coordinates are represented in $O(\log n)$ bits. However, Angelini et al. [1] showed that succinct greedy drawing does not always exist in R^2. They proved that there are infinitely many trees which are greedy drawable, but all greedy drawings need exponential size grids. This again shows that greedy routing may not be practically feasible even such drawing exists. In a recent work [13], Goodrich and Strash show that succinct greedy drawing for the Euclidean metric in R^2, for 3-connected planar graphs, with coordinates that can be represented succinctly with $O(\log n)$ bits. They use a sophisticated coordinate system for R^2. No result on running time is given in their paper.

The essence of the geometric routing is the following: When an origin vertex u needs to send a message to a destination vertex w, it forwards the message to a neighbor t, solely based on the location of w and the location information (namely the locations of u and all neighbors of u.) In the greedy routing scheme, the decision is based on decreasing distance. For this approach to work, however, the decision needs not be based on decreasing distance. As long as the decision is made locally, this approach works fine.

In this paper, we introduce the *generalized greedy routing algorithm*. A generalized greedy routing algorithm uses other criterion, instead of relying on decreasing distance for routing decisions, to determine routing path. In Section 2 we present a generalized greedy routing algorithm for plane triangulations based on Schnyder realizer. In Section 3, we present a generalized greedy routing algorithm for 3-connected plane graphs based on Schnyder woods. Both algorithms use Schnyder coordinates (consisting of three integers between 0 and $2n$). In order to send a message from the origin u to the destination w, our routing algorithm determines the routing path from the Schnyder coordinates of u, w and all neighbors of u. The algorithms are natural and simple to implement.

2 Generalized Greedy Routing for Plane Triangulations

Most definitions we use are standard. We abbreviate the words "counterclockwise" and "clockwise" as ccw and cw respectively.

Definition 1. [19,20]: Let G be a plane triangulation of n vertices with three external vertices v_1, v_2, v_3 in ccw order. A *Schnyder realizer (or simply realizer)* $\mathcal{R} = \{T_1, T_2, T_3\}$ of G is a partition of its internal edges into three sets T_1, T_2, T_3 of directed edges such that the following hold:

- The internal edges incident to v_i are in T_i and directed toward v_i.
- For each internal vertex v of G, v has exactly one edge leaving v in each of T_1, T_2, T_3. The ccw order of the edges incident to v is: leaving in T_1, entering in T_3, leaving in T_2, entering in T_1, leaving in T_3, and entering in T_2. Each entering block may be empty. (See Fig 1 (1)).

Fig 1 (2) shows a realizer of a plane triangulation G. The solid lines (dashed lines and dotted lines, respectively) are the edges in T_1 (T_2 and T_3, respectively).

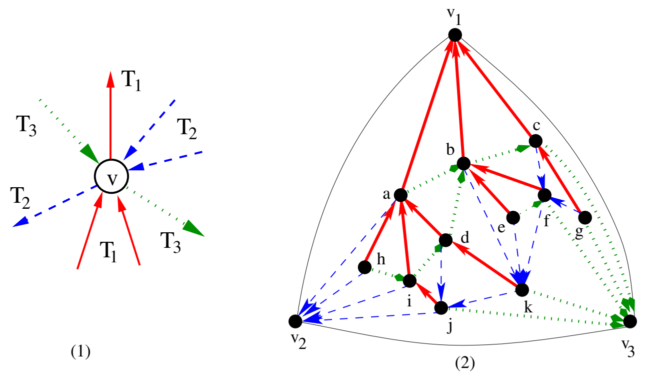


Fig. 1. (1) Edges around a vertex v. (2) A triangulation G and a realizer $\mathcal{R}$.

In [19,20], Schnyder showed that every plane triangulation has a realizer which can be constructed in linear time. Each T_i of a realizer is a tree rooted at v_i, spanning all vertices of G except v_{i-1} and v_{i+1}. (We assume a cyclic structure on the set $\{1, 2, 3\}$. Namely if $i = 3$ then $i + 1 = 1$; if $i = 1$ then $i - 1 = 3$).

For each internal vertex u of G and $i \in \{1, 2, 3\}$, $P_i(u)$ denotes the path in T_i from u to the root v_i of T_i. It was shown in [20] that $P_1(v), P_2(v)$ and $P_3(v)$ have only the vertex v in common, and for two vertices $u \neq v$ and two indices $i \neq j$, $P_i(u)$ and $P_j(v)$ can have at most one common vertex. Let $p_i(w)$ denote the parent of w in T_i. If $u = p_i(w)$, then w is an *i-child* of u. If there is a path in T_i from w to u, then u is an *i-ancestor* of w, and w is an *i-descendant* of u. $R_i(u)$ denotes the region of G bounded by the paths $P_{i-1}(u), P_{i+1}(u)$ and the exterior edge (v_{i-1}, v_{i+1}). We also use $R_i(u)$ to denote the set of vertices in the region $R_i(u)$. Let $R_i^\circ(u)$ denote the interior of $R_i(u)$. Namely $R_i^\circ(u) = R_i(u) - (P_{i-1}(u) \cup P_{i+1}(u))$. We further partition $R_i^\circ(u)$ into three subsets:

1. The *i-Descendants*: $D_i(u) = \{v \mid v \text{ is an } i\text{-descendent of } u\}$.
2. The *i-Left-Cousins*: $LC_i(u) = \{v \mid v \text{ is an } i\text{-descendent of a vertex } w \in P_{i+1}(u) \text{ where } w \neq u\}$.
3. The *i-Right-Cousins*: $RC_i(u) = \{v \mid v \text{ is an } i\text{-descendent of a vertex } w \in P_{i-1}(u) \text{ where } w \neq u\}$.

For instance, in the realizer shown in Fig 1 (2): $P_2(a) = \{a, v_2\}$. $P_3(a)$=$\{a, b, c, v_3\}$. $R_1(a)$ consists of all vertices of G, except v_1. $R_1^\circ(a)$=$\{d, e, f, g, h, i, j, k\}$. $D_1(a) = \{d, h, i, j, k\}$. $LC_1(a) = \emptyset$. $RC_1(a) = \{e, f, g\}$.

Definition 2. Let u and v be two vertices of G and $i \in \{1, 2, 3\}$.

1. $x_i(u)$ denotes the number of faces in the region $R_i(u)$.
2. The *Schnyder coordinate* of u is: $S(u) = (x_1(u), x_2(u), x_3(u))$. (Note that $x_1(u) + x_2(u) + x_3(u) = 2n - 5 =$ the number of internal faces of G.)
3. The *direction signature (or simply signature)* of v with respected to u is: $DS_u(v) = (sign(x_1(v) - x_1(u)), sign(x_2(v) - x_2(u)), sign(x_3(v) - x_3(u)))$ (If $a = b$, define $sign(a - b) = 0$.)

For example, consider the realizer shown in Fig 1 (2). We have $S(a) = (19, 3, 1)$ and $S(e) = (4, 12, 7)$. Thus $DS_a(e) = (- + +)$.

When a vertex u wants to send a message to the destination vertex w, our routing algorithm first calculates the signature $DS_u(w)$, which indicates to which "direction" the message should be sent. The following lemma classifies the direction signatures of vertices in the different regions.

Lemma 1. *Let u be an interior vertex of G.*
1. $\forall v \in D_1(u)$, $DS_u(v) = (- + +)$ *2.* $\forall v \in P_3(u)$, $DS_u(v) = (- - +)$
3. $\forall v \in P_2(u)$, $DS_u(v) = (- + -)$ *4.* $\forall v \in RC_1(u)$, $DS_u(v) = (-?+)$
5. $\forall v \in LC_1(u)$, $DS_u(v) = (-+?)$
The symbol ? indicates that the corresponding component can be $-$, $+$ or 0.

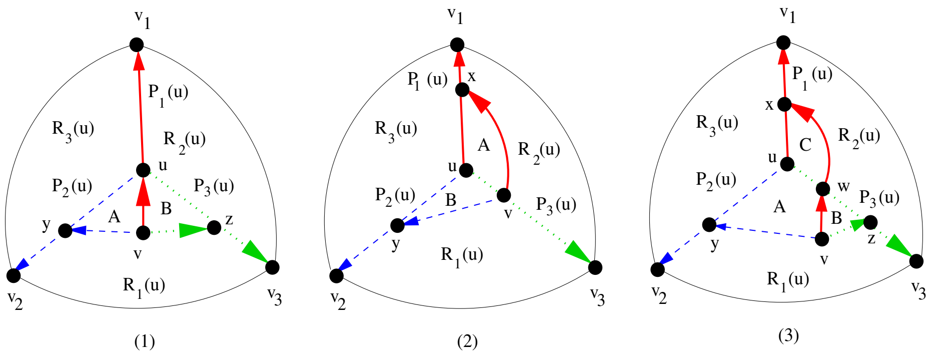


Fig. 2. The illustrations of proof of Lemma 1

Proof. We only prove the statements (1) and (2). The proof of other statements are similar.

(1) Let v be a vertex in $D_1(u)$. Let y be the first common vertex of $P_2(u)$ and $P_2(v)$. Let z be the first common vertex of $P_3(u)$ and $P_3(v)$. (See Fig 2 (1).) Clearly, we have $R_1(v) \subset R_1(u)$. The inclusion is proper because the regions A and B are in $R_1(u)$, but not in $R_1(v)$. Thus $sign(x_1(v) - x_1(u)) = -$. We also have $R_2(u) \subset R_2(v)$. The inclusion is proper because the region B is in $R_2(v)$, but not in $R_2(u)$. Thus $sign(x_2(v) - x_2(u)) = +$. Similarly, $R_3(u) \subset R_3(v)$. The inclusion is proper because the region A is in $R_3(v)$, but not in $R_3(u)$. Thus $sign(x_3(v) - x_3(u)) = +$.

(2) Let v be a vertex in $P_3(u)$. Let x be the first common vertex of $P_1(u)$ and $P_1(v)$. Let y be the first common vertex of $P_2(u)$ and $P_2(v)$. (See Fig 2 (2).) Clearly, we have $R_3(u) \subset R_3(v)$. The inclusion is proper because the regions A and B are in $R_3(v)$, but not in $R_3(u)$. Thus $sign(x_3(v) - x_3(u)) = +$. We also have $R_2(v) \subset R_2(u)$. The inclusion is proper because the region A is in $R_2(u)$, but not in $R_2(v)$. Thus $sign(x_2(v) - x_2(u)) = -$. Similarly, we can show $sign(x_1(v) - x_1(u)) = -$.

By symmetry, we can determine the direction signatures in other regions. These results are illustrated in Fig 3.

Definition 3. Let $\alpha = (\alpha_1, \alpha_2, \alpha_3)$ be a direction signature.
α is the i^+ *signature* if $\alpha_i = +$, and $\alpha_j = -$ for $j = i - 1$ and $j = i + 1$.
α is the i^- *signature* if $\alpha_i = -$, and $\alpha_j = +$ for $j = i - 1$ and $j = i + 1$.

As indicated in Fig 3, we have the following:

Lemma 2. *Let u be an interior vertex of G and $i \in \{1, 2, 3\}$.*

1. *u has exactly one neighbor, which is $p_i(u)$, with the i^+ signature.*
2. *The number of neighbors t of u with the i^- signature equals the number of i-children of u, which can be 0, or 1, or more.*
3. *For any vertex w of G, if $DS_u(w)$ is the i^- signature, then $w \in R_i^\circ(u)$.*

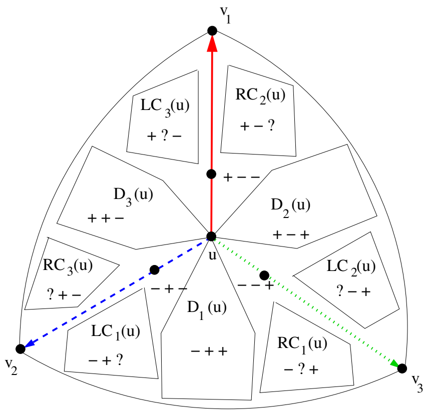


Fig. 3. The direction signatures for vertices in different regions

Let u be the origin, and w the destination of a message. The following routing algorithm determines the neighbor t of u to forward the message, solely based on the the Schnyder coordinates $S(u)$, $S(w)$ and $S(t)$ of all neighbors t of u.

Schnyder Routing Algorithm 1:

1. If w is a neighbor of u, forward the message from u to w and stop.
2. Compute the direction signature $\alpha = DS_u(w)$. If a component of α is 0, replace it by $-$. Let β be the resulting signature. (If there's no 0 component in α, let $\beta = \alpha$.)
3. If β is the i^+ signature, forward the message to $t = p_i(u)$, which is the unique neighbor of u with the i^+ signature.
4. If β is the i^- signature, find a neighbor t of u such that $DS_t(w)$ is also the i^- signature, forward the message to t.

Theorem 1. *For any two vertices u and w in a plane triangulation G, Schnyder Routing Algorithm 1 always forwards a message from u to w in finite steps. The algorithm can be implemented in linear time.*

Proof. We prove the following statements:

1. Schnyder Routing Algorithm 1 can always find a neighbor t of u to forward the message.
2. If $w \in R_i(u)$ for $i \in \{1, 2, 3\}$, then $w \in R_i(t)$ and $x_i(t) < x_i(u)$.

We prove these statements case by case.

Case 1. $\alpha = DS_u(w) = (\alpha_1, \alpha_2, \alpha_3)$ has a 0 component. The two remaining components of α are $+$ and $-$. Without loss of generality, assume $\alpha = (-0+)$. (Other cases are symmetric). Thus $\beta = (--+)$ is the 3^+ signature. As shown in Fig 3, we must have $w \in RC_1(u)$. In this case, the algorithm forwards the message to $t = p_3(u)$. Note that we have $w \in R_1(t)$. Since $DS_u(t) = (--+)$, we have $x_1(t) < x_1(u)$.

Case 2. $\alpha = \beta$ has no 0 component, and it is the i^+ signature. Without loss of generality, we assume $\beta = \alpha = (- - +)$. (The other cases are symmetric). As shown in Fig 3, we must have $w \in RC_1(u) \cup P_3(u) \cup LC_2(u)$. Although the algorithm cannot distinguish to which subset w belongs, it always forwards the message to $t = p_3(u)$ in this case. Note that if $w \in R_1(u)$ then $w \in R_1(t)$ and if $w \in R_2(u)$ then $w \in R_2(t)$. Since $DS_u(t) = (- - +)$, we have $x_1(t) < x_1(u)$.

Case 3. $\alpha = \beta$ has no 0 component, and it is the i^- signature. Without loss of generality, we assume $\beta = \alpha = (- + +)$. (The other cases are symmetric). As shown in Fig 3, we must have $w \in D_1(u) \cup RC_1(u) \cup LC_1(u)$.

Case 3a: $w \in D_1(u)$. Let t be the 1-child of u that is a 1-ancestor of w. Then $DS_t(w) = (- + +)$. So we can find a neighbor t of u that satisfies the condition specified in the algorithm.

Note that u might have more than one neighbor satisfying this condition. For example, u may have another 1-child t' (that is not a 1-ancestor of w) such that $DS_{t'}(w) = (- + +)$. The algorithm might forward the message to t' instead of t. As we will see later, this will work fine.

Case 3b: $w \in RC_1(u)$. Let $t = p_3(u)$ be the parent of u in T_3. We want to show $DS_t(w) = (- + +)$. There are two cases:

(i) $w \in D_1(t)$. Then we have $DS_t(w) = (- + +)$. (See Fig 4 (1).)

(ii) $w \in RC_1(t)$. Then $\gamma = (\gamma_1, \gamma_2, \gamma_3) = DS_t(w) = (-?+)$. (See Fig 4 (2).) We need to show $\gamma_2 = +$. Note that $DS_u(t) = (- - +)$. Thus $x_2(t) < x_2(u)$. Since $DS_u(w) = (- + +)$, we have $x_2(w) > x_2(u)$. Hence $x_2(w) > x_2(t)$. Therefore $\gamma_2 = sign(x_2(w) - x_2(t)) = +$ as to be shown.

Hence we can find a neighbor t of u that satisfies the condition in the algorithm. (Note that u might have another neighbor t' such that $DS_{t'}(w) = (-++)$. The algorithm might forward the message to t' instead of t. This is fine).

Case 3c: $w \in LC_1(u)$. Symmetric to Case 3b.

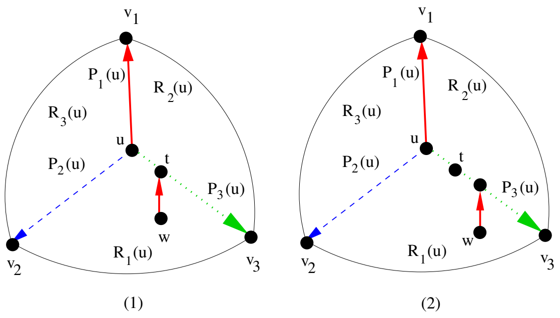


Fig. 4. The two subcases for Case 3b

In all three subcases, the algorithm forwards the message to a neighbor t of u such that $DS_t(w) = (- + +)$. By Lemma 2 (3), we have $w \in R_1^\circ(t) \subseteq R_1(t)$. Also, in all three subcases, u forwards the message to a neighbor t such that $DS_u(t) = (- + +)$, or $(- - +)$. In either case we have $x_1(t) < x_1(u)$.

Next, we show that the algorithm can always forward the message from u to the destination w in finite steps. This follows from the following observations:

1. If $w \in R_i(u)$, then the message is sent to a neighbor t with $w \in R_i(t)$.
2. Since $x_i(t) < x_i(u)$, the message is sent through a path from u to w that is strictly $x_i(*)$ decreasing. So the process stops in at most $x_i(u)$ steps.

The only computation needed by the algorithm is the calculation of Schnyder coordinates. This can be done in linear time [20].

3 Generalized Greedy Routing for 3-Connected Plane Graphs

In this section, we present a generalized greedy routing algorithm for 3-connected plane graphs. The algorithm is based on Schnyder wood, which generalizes the realizer concept from plane triangulation to 3-connected plane graph [7]:

Definition 4. Let G be a 3-connected plane graph with three external vertices v_1, v_2, v_3 in ccw order. A *Schnyder wood* of G is a triplet of rooted spanning trees $\{T_1, T_2, T_3\}$ of G with the following properties:

1. For $i \in \{1, 2, 3\}$, the root of T_i is v_i, the edges of G are directed from children to parent in T_i.
2. Each edge e of G is contained in at least one and at most two spanning trees. If e is contained in two spanning trees, then it has different directions in the two trees.
3. For each $v \notin \{v_1, v_2, v_3\}$ of G, v has exactly one edge leaving v in each of T_1, T_2, T_3. The ccw order of the edges incident to v is: leaving in T_1, entering in T_3, leaving in T_2, entering in T_1, leaving in T_3, and entering in T_2. Each entering block may be empty. An edge with two opposite directions is considered twice. The first and the last incoming edges are possibly coincident with the outgoing edges. (Fig 5 (1) and (2) show two examples of edge pattern around an interval vertex v.)
4. For $i \in \{1, 2, 3\}$, all the edges incident to v_i belong to T_i.

We color the edges in T_1 by red, T_2 blue, and T_3 green. According to the definition, each edge of G is assigned one or two colors, and is said to be *1-colored* or *2-colored*, respectively. It was shown in [7] that every 3-connected plane graph has a Schnyder wood, which can be computed in linear time. Fig 5 (3) shows an example of a Schnyder wood of a 3-connected plane graph G.

For each vertex v of G and $i \in \{1, 2, 3\}$, $P_i(v)$ denotes the path in T_i from v to the root v_i of T_i. The subpath of the external face of G with end vertices v_1 and v_2 and not containing v_3 is denoted by $ext(v_1, v_2)$. The subpaths $ext(v_2, v_3)$ and $ext(v_3, v_1)$ are defined similarly. The three paths $P_1(u), P_2(u)$ and $P_3(u)$ divide G into three regions $R_1(u), R_2(u), R_3(u)$. Define $R_i^\circ(u) = R_i(u) - (P_{i-1}(u) \cup P_{i+1}(u))$. As before, $R_i^\circ(u)$ can be partitioned into three subsets:

1. The *i-Descendants*: $D_i(u) = \{v \in R_i^\circ(u) \mid$ the first i-ancestor of v in $P_{i-1}(u) \cup P_{i+1}(u)$ is $u\}$.
2. The *i-Left-Cousins*: $LC_i(u) = \{v \in R_i^\circ(u) \mid$ the first i-ancestor of v in $P_{i-1}(u) \cup P_{i+1}(u)$ is in $P_{i+1}(u) - \{u\}\}$.

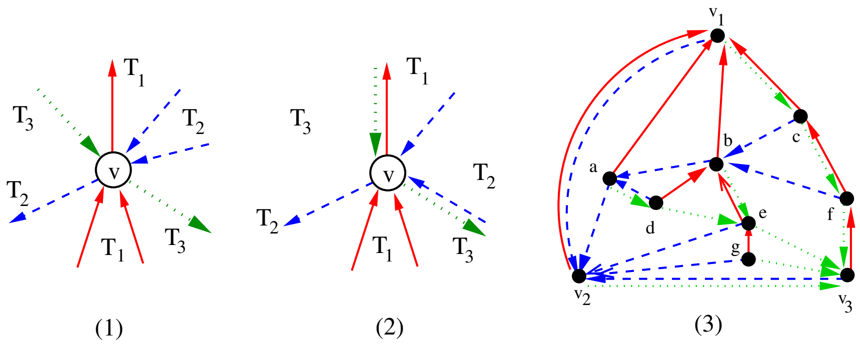


Fig. 5. (1) and (2) Two examples of edge pattern around an internal vertex v; (3) A 3-connected graph G with its Schnyder wood

3. The *i-Right-Cousins*: $RC_i(u) = \{v \in R_i^\circ(u) \mid$ the first i-ancestor of v in $P_{i-1}(u) \cup P_{i+1}(u)$ is in $P_{i-1}(u) - \{u\}\}$.

For example, consider the graph shown in Fig 5 (3). We have $R_1(b) = \{v_2, a, b, d, e, g, v_3\}$. $P_2(b) = \{b, a, v_2\}$, $P_3(b) = \{b, e, v_3\}$. $D_1(b) = \{d\}$, $LC_1(b) = \emptyset$ and $RC_1(b) = \{g\}$. (Note that although g is a 1-descendent of b, since the first 1-ancestor of g is $e \in P_3(b) - \{b\}$, $g \notin D_1(b)$.) The properties of Schnyder wood have been studied in [3,7,9,10,11] and are summarized in the following lemmas.

Lemma 3. *Let $G = (V, E)$ be a 3-connected plane graph and $\mathcal{R} = (T_1, T_2, T_3)$ a Schnyder wood of G.*

1. *For each v of G, $P_1(v)$, $P_2(v)$, $P_3(v)$ have only the vertex v in common.*
2. *For $i, j \in \{1, 2, 3\}$ $(i \neq j)$ and two vertices u and v, the intersection of $P_i(u)$ and $P_j(v)$ is either empty or a common subpath.*
3. *For vertices v_1, v_2, v_3 the following hold: $P_1(v_2) = P_2(v_1) = ext(v_1, v_2)$; $P_2(v_3) = P_3(v_2) = ext(v_2, v_3)$; $P_3(v_1) = P_1(v_3) = ext(v_3, v_1)$.*

Lemma 4. [9,10] *Let $G = (V, E)$ be a 3-connected plane graph with a Schnyder wood $\mathcal{R} = \{T_1, T_2, T_3\}$.*

1. *If $e = (a, b)$ is an unidirectional edge directed from a to b in T_i, then $R_i(a) \subset R_i(b)$, $R_{i-1}(a) \supset R_{i-1}(b)$ and $R_{i+1}(a) \supset R_{i+1}(b)$.*
2. *If $e = (a, b)$ is bidirectional directed from a to b in T_i and from b to a in T_{i+1}, then $R_{i-1}(a) = R_{i-1}(b)$, $R_i(a) \subset R_i(b)$ and $R_{i+1}(a) \supset R_{i+1}(b)$.*
3. *If $e = (a, b)$ is bidirectional directed from a to b in T_i and from b to a in T_{i-1}, then $R_{i+1}(a) = R_{i+1}(b)$, $R_i(a) \subset R_i(b)$ and $R_{i-1}(a) \supset R_{i-1}(b)$.*

Fig 6 (1) illustrates the statement 1 in Lemma 4 with $i = 1$. Fig 6 (2) illustrates the statement 2 in Lemma 4 with $i = 3$. Fig 6 (3) illustrates the statement 3 in Lemma 4 with $i = 3$.

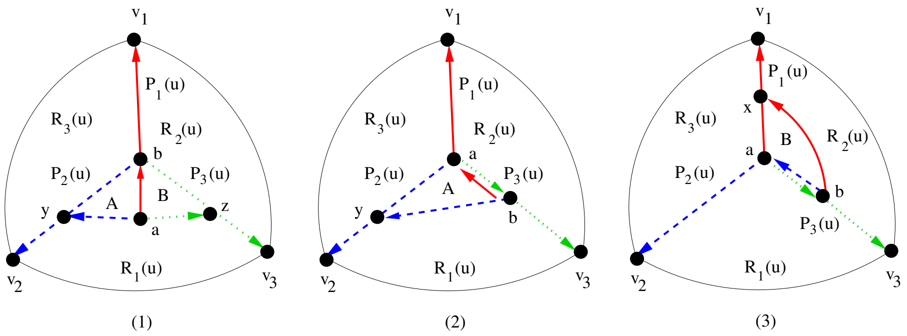


Fig. 6. Illustration of Lemma 4

Lemma 5. *Let u be any vertex of G.*
1. $\forall v \in D_1(u)$, $DS_u(v) = (- + +)$ 2. $\forall v \in P_3(u)$, $DS_u(v) = (\ominus \ominus +)$
3. $\forall v \in P_2(u)$, $DS_u(v) = (\ominus + \ominus)$ 4. $\forall v \in RC_1(u)$, $DS_u(v) = (-?+)$
5. $\forall v \in LC_1(u)$, $DS_u(v) = (-+?)$
The notation $\ominus$ means that the corresponding component in the signature can be either 0 or $-$.

Proof. (1) Let t be a 1-child of u that is not $p_2(u)$ nor $p_3(u)$. Thus the edge $e = (t, u)$ is in T_1 and directed from t to u. We claim e must be an unidirectional edge. If not, then e is directed from u to t in T_2 (or T_3, respectively). But then $t = p_2(u) \in P_2(u)$ (or $t = p_3(u) \in P_3(u)$, respectively). By the statement (1) in Lemma 4 (with $i = 1$), we have $x_1(t) - x_1(u) < 0$, $x_2(t) - x_2(u) > 0$ and $x_3(t) - x_3(u) > 0$. Thus $DS_u(t) = (- + +)$.

Now consider a vertex $v \in D_1(u)$. Let t be the 1-child of u such that v is a 1-descendent of t. Since $v \in D_1(u)$, t is not $p_2(u)$ nor $p_3(u)$. When we move from t to v along a T_1 path, the first component of Schnyder coordinate always decreases by Lemma 4 (with $i = 1$). Similarly, the second component of Schnyder coordinate either increases or remains unchanged, the third component of Schnyder coordinate either increases or remains unchanged. Thus $x_1(v) < x_1(t) < x_1(u)$, $x_2(v) \geq x_1(t) > x_2(u)$, and $x_3(v) \geq x_3(t) > x_3(u)$. Hence $DS_u(v) = (- + +)$.

(2) When we move from u toward v_3 along the path $P_3(u)$, depending on whether we move through a unidirectional or bidirectional edge, by Lemma 4 (with $i = 3$), the first component of Schnyder coordinate either remains unchanged or decreases, the second component of Schnyder coordinate either remains unchanged or decreases, the third component of Schnyder always increases. Thus we have $DS_u(v) = (\ominus \ominus +)$.

(3) Symmetric to (2).

(4) Consider a vertex $v \in RC_1(u)$. Let t be the first 1-ancestor of v that is in $P_3(u)$. By (2), we have $DS_u(t) = (\ominus \ominus +)$. When we move from t toward v along a T_1 path, by Lemma 4 (with $i = 1$), the first component of Schnyder coordinate always decreases, the third component of Schnyder coordinate either increases

or remains unchanged. Thus $DS_u(v) = (-?+)$. (Note that no claim can be made for the second component of the signature $DS_u(v)$.)

(5) Symmetric to (4).

By symmetry, we can determine the direction signatures in other regions. These results are illustrated in Fig 7.

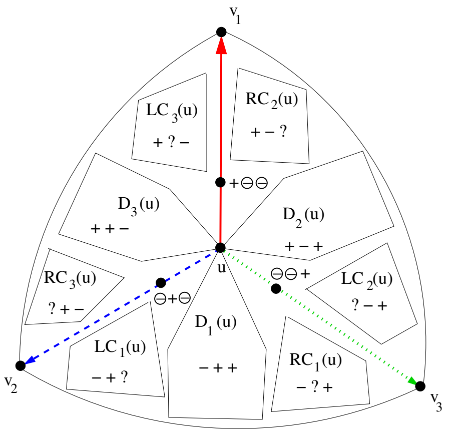


Fig. 7. The direction signatures for vertices in different regions

Definition 5. A direction signature $\alpha = (\alpha_1, \alpha_2, \alpha_3)$ is called a *fuzzy* i^+ *signature* if $\alpha_i = +$, and $\alpha_j = -$ or 0, for $j = i - 1$ and $j = i + 1$.

As indicated in Fig 7, we have the following counter-part of Lemma 2:

Lemma 6. *Let u be an interior vertex of G and $i \in \{1, 2, 3\}$.*

1. *u has exactly one neighbor, which is $p_i(u)$, with the fuzzy i^+ signature.*
2. *The number of neighbors t of u with the i^- signature equals the number of its i-children that are not $p_{i+1}(u)$ nor $p_{i-1}(u)$.*
3. *For any vertex w of G, if $DS_u(w)$ is the i^- signature, then $w \in R_i^\circ(u)$.*

Compare Fig 7 with Fig 3, we can see the only difference is that the signature $DS_u(p_i(u))$ is a fuzzy i^+ signature in Fig 7, while it is a i^+ signature in Fig 3. Since $p_i(u)$ is the only neighbor of u with the fuzzy i^+ signature, u can still recognize which of its neighbors is $p_i(u)$.

Schnyder Routing Algorithm 2:

1. If w is a neighbor of u, forward the message from u to w and stop.
2. For each neighbor t of u, compute the signature $\gamma = DS_u(t)$. If γ is a fuzzy i^+ signature, replace any 0 component by $-$. Let δ be the resulting signature. If γ has no 0 component, let $\delta = \gamma$.

   Compute the direction signature $\alpha = DS_u(w)$. If a component of α is 0, replace it by $-$. Let β be the resulting signature. (If there's no 0 component in α, then $\beta = \alpha$.)

3. If β is the i^+ signature, forward the message to $t = p_i(u)$. (Note that t is the unique neighbor of u with the i^+ signature.)
4. If β is the i^- signature, find a neighbor t of u such that $DS_t(w)$ is also the i^- signature, forward the message to t.

The only difference between Schnyder Routing Algorithm 2 and Schnyder Routing Algorithm 1 is that if a neighbor t of u has a fuzzy i^+ signature γ (in this case t must be $p_i(u)$), we replace it by the i^+ signature δ. The following theorem is the counter-part of Theorem 1. The proof is almost identical to the proof of Theorem 1, and hence omitted.

Theorem 2. *For any two vertices u and w in a 3-connected plane graph G, Schnyder Routing Algorithm 2 always forwards a message from u to w in finite steps. The algorithm can be implemented in linear time.*

References

1. Angelini, P., Battista, G.D., Frati, F.: Succinct greedy drawings do not always exist. In: Proc. GD 2009 (to appear, 2009)
2. Angelini, P., Frati, F., Grilli, L.: An algorithm to construct greedy drawings of triangulations. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 26–37. Springer, Heidelberg (2009)
3. Bonichon, N., Felsner, S., Mosbah, M.: Convex drawings of 3-connected planar graph. Algorithmica 47, 399–420 (2007)
4. Ben-Chen, M., Gotsman, C., Wormser, C.: Distributed computation of virtual coordinates. In: Proc. SoCG 2007, pp. 210–219 (2007)
5. Comer, D.: Internetworking with TCP/IP, Principles, Protocols, and Architecture, vol. 1. Prentice-Hall, Inc., Upper Saddle River (2006)
6. Dhandapani, R.: Greedy drawings of triangulations. In: SODA 2008, pp. 102–111 (2008)
7. Di Battista, G., Tamassia, R., Vismara, L.: Output-sensitive Reporting of Disjoint Paths. Algorithmica 23(4), 302–340 (1999)
8. Eppstein, D., Goodrich, M.T.: Succinct greedy graph drawing in the hyperbolic plane. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 14–25. Springer, Heidelberg (2009)
9. Felsner, S.: Convex Drawings of Planar Graphs and the Order Dimension of 3-Polytopes. Order 18, 19–37 (2001)
10. Felsner, S.: Geodesic Embeddings and Planar Graphs. Order 20, 135–150 (2003)
11. Felsner, S., Zickfeld, F.: Schnyder Woods and Orthogonal Surfaces. Discreate Comput. Geom. 40, 103–126 (2008)
12. Funke, S.: Topological hole detection in wireless sensor networks and its applications. In: DIALM-POMC 2005: Proc. the 2005 joint workshop on Foundations of mobile computing, pp. 44–53 (2005)
13. Goodrich, M.T., Strash, D.: Succinct Greedy Geometric Routing in the Euclidean Plane, submitted to arXiv: 0812.3893v3 (October 2009)
14. Leighton, T., Moitra, A.: Some results on greedy embeddings in metric spaces. In: Proc. FOCS 2008, pp. 337–346 (2008)
15. Lillis, K.M., Pemmaraju, S.V.: On the Efficiency of a Local Iterative Algorithm to Compute Delaunay Realizations. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 69–86. Springer, Heidelberg (2008)

16. Muhammad, R.B.: A distributed geometric routing algorithm for ad hoc wireless networks. In: Proc. 4th Inter. Conf. on Info. Tech (ITNG 2007), pp. 961–963 (2007)
17. Papadimitriou, C.H., Ratajczak, D.: On a conjecture related to geometric routing. Theoretical Computer Science 344(1), 3–14 (2005)
18. Rao, A., Papadimitriou, C.H., Shenker, S., Stoica, I.: Geographic routing without location information. In: Proc. Mobicom 2003, pp. 96–108 (2003)
19. Schnyder, W.: Planar graphs and poset dimension. Order 5, 323–343 (1989)
20. Schnyder, W.: Embedding planar graphs on the grid. In: Proc. of the First Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 138–148 (1990)
21. Tanenbaum, A.S.: Computer networks, 4th edn. Prentice-Hall, Englewood Cliffs (2003)

Exploiting Restricted Linear Structure to Cope with the Hardness of Clique-Width[⋆]

Pinar Heggernes[1], Daniel Meister[2], and Udi Rotics[3]

[1] Department of Informatics, University of Bergen, Norway
pinar.heggernes@ii.uib.no

[2] Theoretical Computer Science, RWTH Aachen University, Germany
meister@cs.rwth-aachen.de

[3] Netanya Academic College, Netanya, Israel
rotics@netanya.ac.il

Abstract. Clique-width is an important graph parameter whose computation is NP-hard. In fact we do not know of any algorithm other than brute force for the exact computation of clique-width on any graph class of unbounded clique-width, other than square grids. Results so far indicate that proper interval graphs constitute the first interesting graph class on which we might have hope to compute clique-width, or at least its variant linear clique-width, in polynomial time. In TAMC 2009, a polynomial-time algorithm for computing linear clique-width on a subclass of proper interval graphs was given. In this paper, we present a polynomial-time algorithm for a larger subclass of proper interval graphs that approximates the clique-width within an *additive* factor 3. Previously known upper bounds on clique-width result in arbitrarily large difference from the actual clique-width when applied on this class. Our results contribute toward the goal of eventually obtaining a polynomial-time exact algorithm for clique-width on proper interval graphs.

1 Introduction

Clique-width is a graph parameter that has many algorithmic applications [4]. NP-hard problems that are expressible in a certain type of extended monadic second-order logic admit algorithms with running time $f(k) \cdot n$ on input graphs of clique-width k with n vertices, where function f depends only on k [5]. Unfortunately it is NP-hard to compute the clique-width of a given graph [8]. This hardness result is also true for its variant linear clique-width, which gives an upper bound on clique-width. Fellows et al. ask whether the computation of clique-width is fixed parameter tractable when parametrised by the clique-width of the input graph [8]. This question is still open. Furthermore, we do not know of an algorithm with running time c^n, where c is a constant.

Clique-width has received a lot of attention recently [1,3,7,8,13,14,15,16,17]. Nevertheless, positive results known on the computation of clique-width so far

[⋆] This work was supported by Deutsche Forschungsgemeinschaft, grant RO 927/7, and by the Reseach Council of Norway.

© Springer-Verlag Berlin Heidelberg 2010

are very restricted. There exist efficient algorithms that, for fixed integer k, decide whether the clique-width of a given graph is more than k or bounded above by some exponential function in k [12,18,19]. Graphs of clique-width at most 3 can be recognised in polynomial time, and their exact clique-width can be computed efficiently [6,2]. Examples of such graph classes are cographs, trees and distance-hereditary graphs [9]. Graphs of bounded treewidth have also bounded clique-width, and their clique-width can be computed in polynomial time [7]. Regarding classes of unbounded clique-width, the class of square grids is the only class for which a polynomial-time clique-width computation algorithm is known [9].

Proper interval graphs have unbounded clique-width and the best approximation of their clique-width so far is maximum clique size plus 1, which can be arbitrarily larger than the actual clique-width. Still this is the most promising class of graphs of unbounded clique-width with respect to whether or not we will be able to obtain a polynomial-time algorithm for the exact computation of their clique-width. There are additive approximation algorithms for grids and for very regular-structured subclasses of proper interval graphs and permutation graphs [9]. The first polynomial-time algorithm to compute linear clique-width on a graph class of unbounded clique-width was given by Heggernes et al. in a TAMC 2009 paper for path powers, which form a subclass of proper interval graphs [10].

In this paper, we continue this line of research and attack the hardness of clique-width by exploiting the linear structure of proper interval graphs. This time we study a significantly larger subclass of proper interval graphs than path powers. We give a polynomial-time approximation algorithm for computing the clique-width and linear clique-width of these graphs within an additive factor of at most 3. Previously known upper bound results do not give an additive approximation when applied to this graph class. Furthermore, the studied graphs constitute the largest graph class on which an algorithm for computing or additively approximating the clique-width is known. The main difference between previously considered graph classes and the graph class that we study in this paper is that our graphs have a much more irregular structure.

2 Definitions, Notation and Clique-Width

We consider simple finite undirected graphs. For a graph $G = (V, E)$, $V = V(G)$ denotes the *vertex set* of G and $E = E(G)$ denotes the *edge set* of G. Edges of G are denoted as uv, which means that u and v are *adjacent* in G. For a vertex u of G, $N_G(u)$ denotes the *neighbourhood* of u in G, which is the set of vertices of G that are adjacent to u. For a vertex pair u, v of G, u is a *true twin* of v if $N_G(u) \cup \{u\} = N_G(v) \cup \{v\}$. Note that a vertex can have several true twins. A graph H is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For a set $S \subseteq V(G)$, the subgraph of G *induced* by S, denoted as $G[S]$, has vertex set S and contains exactly the edges uv of G where $u, v \in S$. For a vertex x of G, $G-x$ denotes that subgraph of G that is induced by $V(G) \setminus \{x\}$. We also say that

$G-x$ is obtained from G by *deleting* x. The *disjoint union* of two graphs G and H is $(V(G) \cup V(H), E(G) \cup E(H))$. When applying the disjoint union operation, we always assume that the two involved graphs have disjoint vertex sets.

Clique-width is defined based on a set of operations. Let $k \geq 1$ be an integer. A *k-labelled graph*, *k-graph* for short, is a graph each of whose vertices is labelled with an integer from the set $\{1, \ldots, k\}$. We define four sets of operations on k-labelled graphs:

– $i(u)$ creates a k-graph on vertex set $\{u\}$ where $i \in \{1, \ldots, k\}$ and u has label i
– $\eta_{i,j}(G)$ adds edges between all vertices with label i and all vertices with label j of G where G is a k-graph, $i, j \in \{1, \ldots, k\}$ and $i \neq j$
– $\rho_{i \to j}(G)$ changes all labels i into label j in G where G is a k-graph and $i, j \in \{1, \ldots, k\}$
– $G \oplus H$ is the disjoint union of G and H where G and H are k-graphs.

A *k-expression* is a properly formed expression using the four types of operations. We say that a graph G *has* a k-expression if there exists a k-expression α such that G is equal to the graph defined by α without the labels. The *clique-width* of a graph G, denoted by $\mathrm{cwd}(G)$, is the smallest integer k such that G has a k-expression. An example for a 3-expression for an induced path on five vertices, $(a_1, a_2, a_3, a_4, a_5)$, is this:

$$\rho_{3\to 1}\Big(\eta_{2,3}\Big(3(a_3) \oplus \Big((\eta_{1,2}(1(a_1) \oplus 2(a_2))) \oplus (\eta_{1,2}(1(a_5) \oplus 2(a_4)))\Big)\Big)\Big)\,.$$

Linear clique-width is a restriction of clique-width that allows the disjoint union to operate on at most one labelled graph with at least two vertices. An expression that respects the restriction on the disjoint union operation is called a *linear expression*. The *linear clique-width* of a graph G, denoted by $\mathrm{lcwd}(G)$, is the smallest integer k such that G has a linear k-expression. Since linear k-expressions are k-expressions, it immediately follows that $\mathrm{cwd}(G) \leq \mathrm{lcwd}(G)$.

3 Proper Interval Graphs and the Bubble Model

A graph G is called *proper interval graph* if each vertex of G can be assigned a closed interval of the real line such that: (1) no interval is properly contained in another, and (2) two vertices of G are adjacent if and only if the corresponding assigned intervals have a non-empty intersection. The class of proper interval graphs is equal to indifference graphs [20].

The *clique number* of a graph G, denoted as $\omega(G)$, is the largest number of vertices in a clique of G. The *pathwidth* of G is the smallest clique number of an interval graph that contains G as a subgraph. The linear clique-width of a graph G is bounded from above by its pathwidth $\mathrm{pw}(G)$; precisely, $\mathrm{lcwd}(G) \leq \mathrm{pw}(G) + 2$ [8]. Since proper interval graphs are interval graphs, it holds for every proper interval graph G that $\mathrm{pw}(G) = \omega - 1$. Together with the general pathwidth upper bound on the linear clique-width, we obtain for every proper

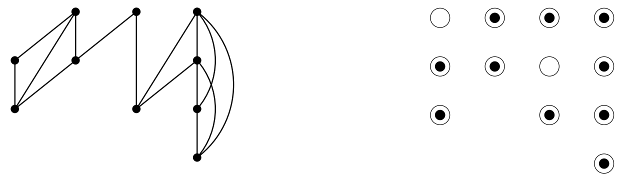


Fig. 1. On the left hand side, find a proper interval graph, and on the right hand side, find a bubble model for the depicted graph. The bubble model contains two empty and ten non-empty bubbles.

interval graph G that $\mathrm{cwd}(G) \leq \mathrm{lcwd}(G) \leq \omega(G) + 1$ [8]. Note that this upper bound can be much larger than the actual linear clique-width. A simple example is a complete graph, that has clique-width and linear clique-width 2 and clique number equal to n.

For the study of the clique-width on proper interval graphs, we use a 2-dimensional model for proper interval graphs. Let G be a graph. A *bubble model* for G is a 2-dimensional structure $\mathcal{B} = \langle B_{i,j} \rangle_{1 \leq j \leq s, 1 \leq i \leq r_j}$ such that the following conditions are satisfied:

- for $1 \leq j \leq k$ and $1 \leq i \leq r_j$, $B_{i,j}$ is a (possibly empty) set of vertices of G
- the sets $B_{1,1}, \ldots, B_{r_k,k}$ are pairwise disjoint and cover $V(G)$
- two vertices u, v of G are adjacent if and only if there are $1 \leq j \leq j' \leq s$ and $1 \leq i \leq r_j$ and $1 \leq i' \leq r_{j'}$ such that $u, v \in B_{i,j} \cup B_{i',j'}$ and (a) $j = j'$ or (b) $j + 1 = j'$ and $i > i'$.

A graph is a proper interval graph if and only if it has a bubble model [10]. Let G be a proper interval graph with bubble model $\mathcal{B} = \langle B_{i,j} \rangle_{1 \leq j \leq s, 1 \leq i \leq r_j}$. A *column* of $\mathcal{B}$ is the collection of all bubbles $B_{i,j}$ for a fixed $1 \leq j \leq s$. The *jth column* of $\mathcal{B}$ is the collection of $B_{1,j}, \ldots, B_{r_j,j}$ and is denoted as $\mathcal{B}_j$. We assume throughout the paper that there is at least one non-empty bubble in the first and last column of $\mathcal{B}$, which means that for $j \in \{1, s\}$ there is $1 \leq i \leq r_j$ such that $B_{i,j} \neq \emptyset$. The *column number* of $\mathcal{B}$, denoted as $\#\mathrm{col}(\mathcal{B})$, is the number of columns of $\mathcal{B}$; in our case, $\#\mathrm{col}(\mathcal{B}) = s$. A simple example of a proper interval graph and a bubble model for it is given in Figure 1. As in the figure, a bubble model may contain empty and non-empty bubbles, and the number of non-empty bubbles in the single columns can be different. As a convention throughout the paper, if the indices of a bubble $B_{i,j}$ exceed the values of $\mathcal{B}$, i.e., if $j < 1$ or $j > s$ or $i < 1$ or $i > r_j$ (for $1 \leq j \leq s$), we assume that $B_{i,j}$ still exists and is empty. The interior of $\mathcal{B}$ is the set of bubbles of $\mathcal{B}$ that are above a non-empty bubble. Formally, the *interior* of $\mathcal{B}$, denoted as $\mathrm{in}(\mathcal{B})$, is the set $\{(i,j) : 1 \leq j \leq s \text{ and } 1 \leq i \text{ and } \exists i' (i \leq i' \text{ and } B_{i',j} \neq \emptyset)\}$. We say that $\mathcal{B}$ is *full* if for every $(i,j) \in \mathrm{in}(\mathcal{B})$, $B_{i,j}$ contains at least one vertex. Proper interval graphs with full bubble models are efficiently recognisable.

Proposition 1. *There is a linear-time algorithm that on input a connected proper interval graph G decides whether G has a full bubble model, and if so, outputs a full bubble model for G where true twins appear in the same bubble.*

4 An Upper Bound on the Linear Clique-Width

We will give an efficient algorithm for computing a linear expression for proper interval graphs with full bubble models. This expression will provide an upper bound on the linear clique-width, and thus the clique-width, of the graph.

Let G be a proper interval graph with bubble model $\mathcal{B} = \langle B_{i,j} \rangle_{1 \le j \le s, 1 \le i \le r_j}$. For $k \ge 0$, a *k-box* in $\mathcal{B}$ is the collection of bubbles $B_{a,b}, \ldots, B_{a+k,b}, B_{a,b+1}, \ldots, B_{a+k,b+k-1}$ for $1 \le b \le s-k+1$ such that $a+k \le r_j$ for all $b \le j \le b+k-1$. Intuitively, a k-box is a rectangle of height $k+1$ and width k that can be placed in $\mathcal{B}$ in the range of $\text{in}(\mathcal{B})$. The pair (a,b) is called the *origin* of the box. It is immediately clear that if (a,b) is the origin of a k-box then also $(1,b)$ is the origin of a k-box. The *box number* of $\mathcal{B}$ is the largest k such that $\mathcal{B}$ has a k-box.

Lemma 1. *Let G be a connected proper interval graph that has a full bubble model. Let $\mathcal{B} = \langle B_{i,j} \rangle_{1 \le j \le s, 1 \le i \le r_j}$ be a full bubble model for G where true twins appear in the same bubble.*

1) *Let κ be the box number of $\mathcal{B}$ plus 1.*
2) *Let $\alpha =_{\text{def}} 0$ if $\mathcal{B}$ has at most one vertex per bubble; otherwise, let $\alpha =_{\text{def}} 1$.*
3) *Let $b \le s$ be smallest such that $\mathcal{B}$ contains a $(\kappa-1)$-box with origin $(1,b)$. If $\mathcal{B}_{b+\kappa-1}$ contains at least two vertices then let $\beta =_{\text{def}} 0$; otherwise, let $\beta =_{\text{def}} -1$.*

Then, $\text{lcwd}(G) \le (\kappa+2) + \alpha + \beta$ *and a linear $(\kappa+2+\alpha+\beta)$-expression for G can be computed in time $\mathcal{O}(n^2)$.*

Proof. We define a linear $(\kappa+2+\alpha+\beta)$-expression for G. Let $\mathscr{H}$ be the set of indices j with $1 \le j \le s$ and $r_j \le \kappa$. Informally, $\mathscr{H}$ represents the set of "short" columns of $\mathcal{B}$; the other columns are called "long". Let $\mathscr{H} = \{j_1, \ldots, j_p\}$, where we assume without loss of generality that $j_1 < \cdots < j_p$. By definition of $\mathscr{H}$, it holds that $r_j \ge \kappa+1$ for every $1 \le j \le s$ with $j \notin \mathscr{H}$. Let $j_0 =_{\text{def}} 0$ and $j_{p+1} =_{\text{def}} s+1$. It follows that $j_{i+1} - j_i \le \kappa$ for all $0 \le i \le p$. Otherwise, $j_{i+1} - j_i \ge \kappa+1$ for some $0 \le i \le p$ implies the existence of a κ-box with origin $(1, j_i+1)$ in $\mathcal{B}$, which contradicts the definition of κ as being larger than the box number of $\mathcal{B}$. So, there are at most $\kappa-1$ long columns between each pair of consecutive short columns in $\mathcal{B}$. For the construction of our expression for G, we partition $\mathcal{B}$ into small parts of long columns, separated by short columns, and process $\mathcal{B}$ from right to left. Due to space restrictions, we concentrate on the case when $\alpha = 0$ and $\beta = 0$.

We construct the linear expression for G inductively. For convenience reasons, we assume that $\mathcal{B}$ also has columns $\mathcal{B}_0$ and $\mathcal{B}_{s+1}$, and $r_0 =_{\text{def}} r_{s+1} =_{\text{def}} 0$. Let $1 \le a \le p+1$. Denote by G_a the subgraph of G that is induced by the vertices in the columns $\mathcal{B}_{j_a}, \ldots, \mathcal{B}_{s+1}$. We assume that we already have a linear expression $\mathscr{E}_a$ for G_a that defines a labelled graph with the following labels: the vertices in the columns $\mathcal{B}_{j_a+1}, \ldots, \mathcal{B}_{s+1}$ have label 1, and the vertices in B_{i,j_a} have label $i+1$ for every $1 \le i \le r_{j_a} \le \kappa$ (remember that $\mathcal{B}_{j_a}$ is a short column). It is important to note that we can assume this particularly for the induction base case of $a = p+1$. For the later arguments, observe the following facts:

– (since $\mathscr{E}_a$ defines a labelled graph) all vertices of G_a have a label
– only labels from the set $\{1, \ldots, \kappa + 1\}$ are used
– vertices with the same label have the same neighbours outside of G_a and no vertex with label 1 has a neighbour outside of G_a.

We show that we can define a linear expression $\mathscr{E}_{a-1}$ for G_{a-1} that uses at most $(\kappa + 2) + \alpha$ labels.

We construct $\mathscr{E}_{a-1}$ in three phases. We partition the vertices in columns $\mathcal{B}_{j_{a-1}+1}, \ldots, \mathcal{B}_{j_a-1}$ into three areas, as indicated by the three different area backgrounds in Figure 2:

– area 1: bubbles $B_{j_a-j+1,j}, \ldots, B_{\kappa,j}$ for all $j_{a-1} < j < j_a$
– area 2: bubbles $B_{\kappa+1,j}, \ldots, B_{r_j,j}$ for all $j_{a-1} < j < j_a$
– area 3: bubbles $B_{1,j}, \ldots, B_{j_a-j,j}$ for all $j_{a-1} < j < j_a$.

We first add the vertices from area 1, then from area 2 and finally from area 3. Let $W =_{\text{def}} j_a - j_{a-1} - 1$, which is the number of columns that are properly between $\mathcal{B}_{j_{a-1}}$ and $\mathcal{B}_{j_a}$. In other words, these are the W consecutive long columns that we will add during this phase. Note that area 3 has the shape of an isosceles triangle, whereas area 1 has the ideal shape as in Figure 2 only in case when W has its maximum possible value, which is $W = \kappa - 1$.

Throughout the following construction, we will use label $\kappa + 2$ for introducing a new vertex; we will refer to it as "creating the vertices in a bubble". For obtaining $\mathscr{E}_{a-1}$ by extending $\mathscr{E}_a$, we consider the vertices in areas 1, 2, 3. This

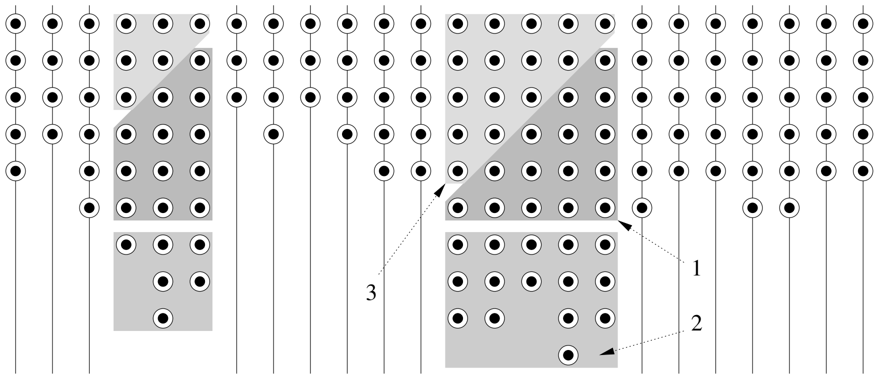


Fig. 2. Depicted is a bubble model for a proper interval graph. The box number is 5, as the largest box has width 5 and height 6. The vertical line segments identify "short columns", that have no vertex in row 7. The other, "long", columns, which have a vertex in row 7, form areas that are partitioned into subareas. There are two areas of long columns in the depicted bubble model. Each of the areas of long columns is partitioned into three subareas, that are identified by the shaded backgrounds and numbered in the right hand side occurrence. The proof of Lemma 1 gives an algorithm for computing a linear expression and treats long columns particularly different from short columns.

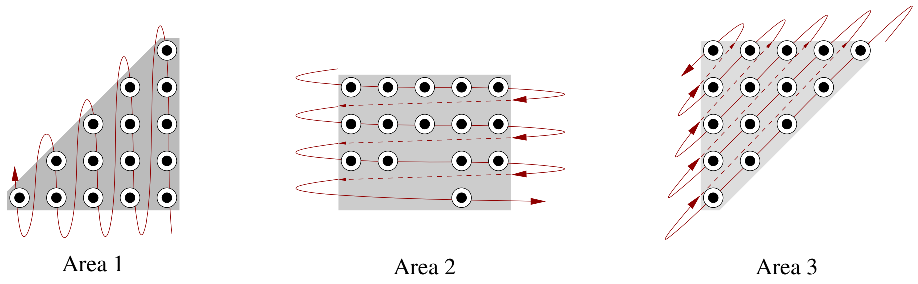


Fig. 3. In the proof of Lemma 1: the three areas in Figure 2 are processed separately and according to the scheme, indicated by the curves

is of particular interest, if $W \geq 1$. Otherwise, the three areas are empty. Due to space restrictions, we concentrate on the case when $W \geq 2$.

We partition the construction for this case into three smaller parts according to the description in Figure 2. The vertices in each area are processed according to special pattern. The three patterns are sketched in Figure 3.

Area 1
This area is marked with number 1 in Figure 2. We process W column intervals of $\mathcal{B}$, starting with the longest one. The first interval consists of the bubbles $B_{2,j_a-1}, \ldots, B_{\kappa,j_a-1}$; we process these bubbles in a bottom-to-top manner. We create the vertices in B_{κ,j_a-1}, then we make all vertices with label $\kappa+2$ adjacent to all vertices with labels $2, \ldots, \kappa$. After this, the vertices in $B_{\kappa-1,j_a}$ and B_{κ,j_a} are not distinguishable anymore. So, we change label κ to label $\kappa+1$, and then, we change label $\kappa+2$ to label κ.

If $\kappa \geq 3$, we proceed with the vertices in $B_{\kappa-1,j_a-1}$. We create the vertices in $B_{\kappa-1,j_a-1}$, then we make the vertices with label $\kappa+2$ adjacent to the vertices with labels $2, \ldots, \kappa$. We change label $\kappa-1$ to label $\kappa+1$, and then, we change label $\kappa+2$ to label $\kappa-1$. This process is continued analogously until the vertices in B_{2,j_a-1} have been processed. Then, it holds that all vertices in $\mathcal{B}_{j_a}$ have label $\kappa+1$, and the vertices in B_{i,j_a-1} have label i for all $2 \leq i \leq \kappa$.

We analogously continue with the next intervals, until all bubbles of area 1 have been processed. At the end of the process, the followings holds for the vertices in area 1: the vertices in column $\mathcal{B}_{j_{a-1}+j}$ have label $\kappa - W + j$ for all $2 \leq j \leq W$, the vertices in bubble $B_{i,j_{a-1}+1}$ have label $i - W + 1$ for all $W+1 \leq i \leq \kappa$, and the vertices in column $\mathcal{B}_{j_a}$ have label $\kappa+1$.

Area 2
This area is marked with number 2 in Figure 2. We process the rows in a top-to-bottom manner and within a row from left to right. The first vertices to process are from bubble $B_{\kappa+1,j_{a-1}+1}$. We create the vertices in $B_{\kappa+1,j_{a-1}+1}$. The vertices with label $\kappa+2$ have to be made adjacent to all already created vertices in columns $\mathcal{B}_{j_{a-1}+1}$ and $\mathcal{B}_{j_{a-1}+2}$, which are exactly the vertices with labels $2, \ldots, \kappa - W + 2$. So, we make all vertices with label $\kappa+2$ adjacent to

all vertices with labels $2, \ldots, \kappa - W + 2$, and then, we change label $\kappa + 2$ to label $\kappa - W + 1$. Note here that the vertices with label $\kappa - W + 1$ are exactly the vertices in bubbles $B_{\kappa, j_{a-1}+1}$ and $B_{\kappa+1, j_{a-1}+1}$, that have no neighbours in column $\mathcal{B}_{j_{a-1}}$.

We continue and create the vertices in $B_{\kappa+1, j_{a-1}+2}$, make the vertices with label $\kappa+2$ adjacent to the vertices with labels $\kappa-W+2$ and $\kappa-W+3$, and then change label $\kappa + 2$ to $\kappa - W + 2$. This process continues until bubble $B_{\kappa+1, j_a-1}$ has been executed.

If there is $r_j > \kappa+1$ for some $j_{a-1} < j < j_a$, we repeat the procedure with the bubbles $B_{\kappa+2, j_{a-1}+1}, \ldots, B_{\kappa+2, j_a-1}$, and so on until all vertices from area 2 have been created. When all rows of area 2 have been processed, all neighbours of the vertices in column $\mathcal{B}_{j_a}$ have been created and made adjacent, so that we can change label $\kappa + 1$ to 1. It holds that after completion of this area 2, the already created vertices in column $\mathcal{B}_{j_{a-1}+j}$ have label $\kappa - W + j$ for all $2 \leq j \leq W$. For $\mathcal{B}_{j_{a-1}+1}$, it holds that the vertices in $B_{i, j_{a-1}+1}$ have label $i - W + 1$ for all $W + 1 \leq i < \kappa$ and the other vertices have label $\kappa - W + 1$. The vertices in columns $\mathcal{B}_{j_a}, \ldots, \mathcal{B}_{j_{p+1}}$ have label 1. It is important to note here that label $\kappa+1$ is not used.

Area 3

This area is marked with number 3 in Figure 2. We process diagonals. Remember that the shape of this area is an isosceles triangle. This also means that there are exactly W diagonals to process. We begin with the longest diagonal and end with the shortest diagonal. Bubbles on diagonals are processed from upper right to lower left. We create the vertices in B_{1, j_a-1}, make all vertices with label $\kappa+2$ adjacent to all vertices with label κ, and change label $\kappa + 2$ to label $\kappa + 1$. Next, we create the vertices in B_{2, j_a-2}, make all vertices with label $\kappa + 2$ adjacent to the already created vertices from $\mathcal{B}_{j_a-2}$ and to the vertices with label $\kappa + 1$, change label $\kappa + 1$ to κ, and then change label $\kappa + 2$ to $\kappa + 1$. We continue this until we completed bubble $B_{W, j_a-W} = B_{W, j_{a-1}+1}$. Note that, similar to the case of area 2, special care has to be taken of the different labels of the vertices in $\mathcal{B}_{j_{a-1}+1}$. All vertices in column $\mathcal{B}_{j_a-1}$ have now been created and made adjacent to all their neighbours. So, we change label κ to label 1. Then, we change label $\kappa + 1$ to label κ.

We continue and repeat the process analogously with the next smaller diagonal until all bubbles from area 3 except for $B_{1, j_{a-1}+1}$ have been processed. The situation now is the following. The vertices from columns $\mathcal{B}_{j_{a-1}+2}, \ldots, \mathcal{B}_{j_{p+1}}$ have label 1. The vertices in bubble $B_{i, j_{a-1}+1}$ have label $\kappa - W + i$ for all $2 \leq i \leq W$, the vertices in bubble $B_{i, j_{a-1}+1}$ have label $i - W + 1$ for all $W + 1 \leq i < \kappa$, and finally, the vertices in bubble $B_{i, j_{a-1}+1}$ have label $\kappa - W + 1$ for all $\kappa \leq i \leq r_{j_{a-1}+1}$. Note again that label $\kappa + 1$ is not used.

We complete area 3 by creating the vertices in $B_{1, j_{a-1}+1}$ and making them adjacent to all other vertices in $\mathcal{B}_{j_{a-1}+1}$. Then, all vertices from $\mathcal{B}_{j_{a-1}+1}$ have been created, and all neighbours of the vertices in $B_{\kappa, j_{a-1}+1}, \ldots, B_{r_{j_{a-1}+1}, j_{a-1}+1}$ have been created and made adjacent. So, the label of these vertices, that are exactly the vertices with label $\kappa - W + 1$, can be changed to label 1. Remember that $\mathcal{B}_{j_{a-1}}$ is a short column. Using the free label $\kappa+1$, we can change the labels of the other vertices in $\mathcal{B}_{j_{a-1}+1}$ so that it holds for all $1 \leq i < \kappa$ that the vertices in $B_{i,j_{a-1}+1}$ have label $i+1$. This completes the construction for the vertices in area 3.

For completing the definition of $\mathscr{E}_{a-1}$, it remains to add the vertices from column $\mathcal{B}_{j_{a-1}}$. This follows the procedure laid out for the vertices in area 1. If $W = 0$ then the vertices in B_{κ,j_a} have no neighbour in $\mathcal{B}_{j_{a-1}}$, and so we change label $\kappa+1$ to label 1. We create the vertices in $B_{\kappa,j_{a-1}}$, make all vertices with label $\kappa+2$ adjacent to all vertices with labels $2,\ldots,\kappa$, change label κ to label 1, and then change label $\kappa+2$ to label $\kappa+1$. We continue with $B_{\kappa-1,j_{a-1}}$ and proceed until the vertices in $B_{1,j_{a-1}}$ have been created and finally received label 2. This completes the definition of $\mathscr{E}_{a-1}$ and satisfies the conditions. By induction, we obtain the claimed result. □

5 The Approximation Result

The previous section has established an upper bound on the linear clique-width of proper interval graphs with full bubble models. The upper bound is dependent on the box number parameter. In this section, we complete this result by giving lower bounds. The upper and lower bounds together will provide an approximation on the clique-width and linear clique-width of proper interval graphs with full bubble models.

An *induced path* is a graph P that admits a vertex sequence $(x_1,\ldots,x_n)$ such that $E(P) = \{x_1x_2,\ldots,x_{n-1}x_n\}$ are the edges of P. For $k \geq 1$, the *kth power* of P is the graph G on vertex set $\{x_1,\ldots,x_n\}$, and two vertices x_i,x_j of G are adjacent if and only if $|i-j| \leq k$. For $k \geq 1$, a *k-path power* is a graph that is the kth power of some induced path. Every k-path power is a proper interval graph. The linear clique-width of k-path powers is completely characterised [11], and there exists a very good lower bound on the clique-width of k-path powers [9]. The following proposition summarises the cases that are important for our results.

Proposition 2 ([11], [9]). *Let G be a k-path power on n vertices, where $k \geq 1$.*

- *If $n = k(k+1)+2$ then $\mathrm{lcwd}(G) = k+2$ and $\mathrm{cwd}(G) \geq k$.*
- *If $n = k(k+1)$ then $\mathrm{lcwd}(G) = k+1$ and $\mathrm{cwd}(G) \geq k$.*

Corollary 1. *Let G be a proper interval graph with full bubble model $\mathcal{B} = \langle B_{i,j}\rangle_{1\leq j\leq s, 1\leq i\leq r_j}$. Let $\mathcal{B}$ contain a k-box with origin $(1,b)$, where $k \geq 1$. Then, $\mathrm{lcwd}(G) \geq k+1$ and $\mathrm{cwd}(G) \geq k$. If $b \leq s-k$ and $r_{b+k} \geq 2$ then $\mathrm{lcwd}(G) \geq k+2$.*

Let G be a graph. If G contains no pair of adjacent vertices then $\mathrm{cwd}(G) \leq \mathrm{lcwd}(G) \leq 1$. Otherwise, $\mathrm{lcwd}(G) \geq \mathrm{cwd}(G) \geq 2$. It is easy to check that $\mathrm{lcwd}(G) \geq \mathrm{cwd}(G) \geq 3$ if G contains an induced path on four vertices as induced subgraph. With these additional and easy bounds on clique-width, we are ready to prove the main result of the paper.

Theorem 1. *There is an $\mathcal{O}(n^2)$-time algorithm that on input a connected proper interval graph G with full bubble model, approximates the clique-width of G within* $\mathrm{cwd}(G)+3$ *and approximates the linear clique-width of G within* $\mathrm{lcwd}(G)+3$. *If G contains no true twins then the algorithm approximates the linear clique-width of G within* $\mathrm{lcwd}(G)+1$.

Proof. Let G be a connected proper interval graph with full bubble model. If u and v are true twins of G then $\mathrm{cwd}(G) = \mathrm{cwd}(G-v)$. With iterated application of this argument, it holds that $\mathrm{cwd}(G) = \mathrm{cwd}(G')$ where G' is a maximal induced subgraph of G without true twins. Let $\mathcal{B} = \langle B_{i,j}\rangle_{1\le j\le s, 1\le i\le r_j}$ be a full bubble model for G where true twins appear in the same bubble. Remember that $\mathcal{B}$ exists according to Proposition 1. Let k be the box number of $\mathcal{B}$. If G is a complete graph then $\mathrm{cwd}(G) \le \mathrm{lcwd}(G) \le 2$. Henceforth, let G not be complete. This particularly means that $s \ge 2$ and $r_1, \ldots, r_{s-1} \ge 2$. Thus, $k \ge 1$, since $\mathcal{B}$ has a 1-box with origin $(1, 1)$. We distinguish between two cases according to the value of k. As the first case, let $k = 1$. If $s = 2$ and $r_s = 1$ then $2 \le \mathrm{cwd}(G) \le \mathrm{lcwd}(G) \le 3$. Otherwise, if $r_2 \ge 2$, G contains an induced path on four vertices as induced subgraph, by choosing a vertex from each of the four bubbles $B_{1,1}, B_{2,1}, B_{1,2}, B_{2,2}$. This means that $\mathrm{lcwd}(G) \ge \mathrm{cwd}(G) \ge 3$. We apply the algorithm of Lemma 1. It holds that $\kappa = 2$ and $\beta \le 0$ and $\alpha \le 1$. This gives $3 \le \mathrm{cwd}(G) \le \mathrm{lcwd}(G) \le (\kappa+2)+\alpha \le \mathrm{cwd}(G)+2$. If G has no true twins then $\alpha = 0$, and thus, $(\kappa + 2) + \alpha \le \mathrm{lcwd}(G) + 1$.

Now, let $k \ge 2$. We first consider linear clique-width. We apply the algorithm of Lemma 1. It holds that $\kappa = k + 1$ and $\beta \le 0$ and $\alpha \le 1$. Hence, $\mathrm{cwd}(G) \le \mathrm{lcwd}(G) \le (k + 3) + \alpha + \beta$. We first consider linear clique-width and the case when G has no true twins; in particular, $\alpha = 0$. Let b be smallest such that $\mathcal{B}$ has a k-box with origin $(1, b)$. Due to definition, $\beta = 0$ if and only if $r_{b+k} \ge 2$, where we assume $r_{s+1} = 0$. Due to Corollary 1, it holds that $\mathrm{lcwd}(G) \ge k + 2 + \beta$. We combine the upper and lower bound results and obtain that $\mathrm{lcwd}(G) \le (k + 3) + \beta \le \mathrm{lcwd}(G) + 1$. If G has true twins then $\alpha = 1$. Due to Corollary 1, $\mathrm{lcwd}(G) \ge k+1$, which shows that $\mathrm{lcwd}(G) \le (k+3)+1 \le \mathrm{lcwd}(G)+3$. In case of clique-width, we consider $\mathrm{cwd}(G')$. We obtain a bubble model $\mathcal{B}'$ for G' by deleting the vertices in $V(G) \setminus V(G')$ from $\mathcal{B}$. Due to the assumption about the true twins being in the same bubble, $\mathcal{B}'$ is a full bubble model for G', and the box number of $\mathcal{B}'$ is k. Due to Corollary 1, $\mathrm{cwd}(G') \ge k$, and applying Lemma 1, where $\kappa = k + 1$, $\beta \le 0$ and $\alpha = 0$, we obtain together with the above lower bound that $\mathrm{cwd}(G) \le k + 3 \le \mathrm{cwd}(G) + 3$. This completes the proof of the theorem. □

6 Final Remarks

We gave an efficient approximation algorithm for computing the clique-width and the linear clique-width of proper interval graphs with a full bubble model. The correctness and quality of the result relies on almost tight lower and upper

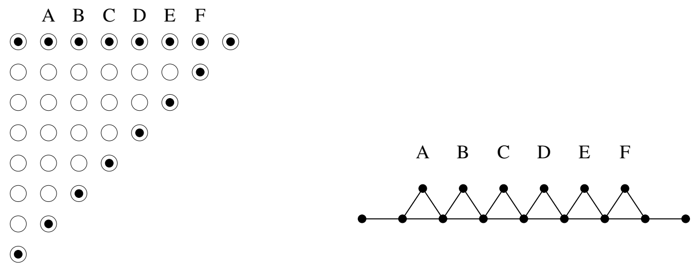


Fig. 4. The left hand side shows a bubble model of the right hand side depicted graph. The graph has clique-width 3 and linear clique-width 3 and the box number of the bubble model is 4.

bounds. The lower bound on the clique-width and linear clique-width is obtained from the results about path powers and the close relationship between the box number and the existence of large path powers as induced subgraphs in these proper interval graphs. In Figure 4, we give an example showing that this close relationship does not extend to arbitrary proper interval graphs.

Can our results help to understand and solve the general case for proper interval graphs? We think that our results are indeed very helpful for the continuation of the work with the aim of computing the clique-width and linear clique-width of proper interval graphs exactly. A closer study of our results shows that the approximation result can be extended to a much larger class of proper interval graphs, namely to all proper interval graphs for which the box number gives a good approximation on the size of an induced subgraph that is a large path power. Another possible extension is by identifying another class of well-structured proper interval graphs, similar to path powers, that can provide a lower bound on the clique-width and linear clique-width of proper interval graphs. Up to now, path powers are the only proper interval graphs for which such lower bound results exist.

Finally, we want to mention that our approximation result gives hope even for an algorithm that computes the clique-width and linear clique-width of proper interval graphs with full bubble models exactly. We believe that our upper bounds for graphs without true twins are tight. So, the main challenge is a matching lower bound. The currently most promising approach is again the identification of small and well-structured induced subgraphs that already require larger clique-width.

Acknowledgements

The authors are grateful to Charis Papadopoulos and Daniel Lokshtanov for fruitful discussions on the subject and valuable hints.

References

1. Brandstädt, A., Dragan, F., Le, H.-O., Mosca, R.: New Graph Classes of Bounded Clique-Width. Theory of Computing Systems 38, 623–645 (2005)
2. Corneil, D.G., Habib, M., Lanlignel, J.-M., Reed, B.A., Rotics, U.: Polynomial Time Recognition of Clique-Width ≤ 3 Graphs (Extended Abstract). In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 126–134. Springer, Heidelberg (2000)
3. Corneil, D.G., Rotics, U.: On the Relationship between Clique-width and Treewidth. SIAM Journal on Computing 34, 825–847 (2005)
4. Courcelle, B., Engelfriet, J., Rozenberg, G.: Handle-rewriting hypergraph grammars. Journal of Computer and System Sciences 46, 218–270 (1993)
5. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. Theory of Computing Systems 33, 125–150 (2000)
6. Courcelle, B., Olariu, S.: Upper bounds to the clique width of graphs. Discrete Applied Mathematics 101, 77–114 (2000)
7. Espelage, W., Gurski, F., Wanke, E.: Deciding clique-width for graphs of bounded tree-width. Journal of Graph Algorithms and Applications 7, 141–180 (2003)
8. Fellows, M.R., Rosamond, F.A., Rotics, U., Szeider, S.: Clique-Width is NP-Complete. SIAM Journal on Discrete Mathematics 23, 909–939 (2009)
9. Golumbic, M.C., Rotics, U.: On the Clique-Width of Some Perfect Graph Classes. International Journal of Foundations of Computer Science 11, 423–443 (2000)
10. Heggernes, P., Meister, D., Papadopoulos, C.: A new representation of proper interval graphs with an application to clique-width. Electronic Notes in Discrete Mathematics 32, 27–34 (2009)
11. Heggernes, P., Meister, D., Papadopoulos, C.: A Complete Characterisation of the Linear Clique-Width of Path Powers. In: Chen, J., Cooper, S.B. (eds.) TAMC 2009. LNCS, vol. 5532, pp. 241–250. Springer, Heidelberg (2009)
12. Hlinený, P., Oum, S.-I.: Finding branch-decompositions and rank-decompositions. SIAM Journal on Computing 38, 1012–1032 (2008)
13. Hlinený, P., Oum, S.-I., Seese, D., Gottlob, G.: Width parameters beyond tree-width and their applications. The Computer Journal 51, 326–362 (2008)
14. Lozin, V.: From tree-width to clique-width: excluding a unit interval graph. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 872–883. Springer, Heidelberg (2008)
15. Lozin, V., Rautenbach, D.: Chordal bipartite graphs of bounded tree- and clique-width. Discrete Mathematics 283, 151–158 (2004)
16. Lozin, V., Rautenbach, D.: On the Band-, Tree-, and Clique-Width of Graphs with Bounded Vertex Degree. SIAM Journal on Discrete Mathematics 18, 195–206 (2004)
17. Müller, H., Urner, R.: On a disparity between relative cliquewidth and relative NLC-width. Discrete Applied Mathematics (to appear)
18. Oum, S.-I.: Approximating rank-width and clique-width quickly. ACM Transactions on Algorithms 5 (2008)
19. Oum, S.-I., Seymour, P.: Approximating clique-width and branch-width. Journal on Combinatorial Theory, Series B 96, 514–528 (2006)
20. Roberts, F.S.: Indifference graphs. In: Harary, F. (ed.) Proof techniques in graph theory, pp. 139–146. Academic Press, New York (1969)

A Note on the Testability of Ramsey's Class

Charles Jordan[⋆] and Thomas Zeugmann[⋆⋆]

Division of Computer Science
Hokkaido University, N-14, W-9, Sapporo 060-0814, Japan
{skip,thomas}@ist.hokudai.ac.jp

Abstract. In property testing, the goal is to distinguish between objects that satisfy some desirable property and objects that are *far* from satisfying it, after examining only a small, random sample of the object in question. Although much of the literature has focused on properties of graphs, very recently several strong results on hypergraphs have appeared. We revisit a logical result obtained by Alon *et al.* [1] in the light of these recent results. The main result is the testability of *all* properties (of relational structures) expressible in sentences of *Ramsey's class.*

Keywords: property testing, logic, Ramsey's class.

1 Introduction

Alon *et al.* [1] proved the testability of all graph properties expressible in sentences of first-order logic where all quantifier alternations are of the type '$\exists\forall$'. This class is the restriction of Ramsey's [12] class to undirected graphs. Fischer [7] extended this (and other results of [1]) to tournaments. However, Ramsey's [12] class is traditionally not restricted to undirected graphs; any (finite) number of predicate symbols with any (finite) arities may appear. It is therefore natural to ask whether one can extend this result to relational structures.

This result of Alon *et al.* [1] has been influential, and has already been extended several times, cf., e.g., [2,8,13]. These extensions have generally focused on an intermediate result: the testability of colorability (and eventually hereditary[1]) properties. In particular, Austin and Tao [4] have recently obtained a strong result: the testability (with one-sided error) of hereditary properties of directed, colored, not necessarily uniform hypergraphs. We return to the logical classification begun by Alon *et al.* [1] and show that this result and generalizations of the remaining parts of the proof in Alon *et al.* [1] combine to give our desired result, the testability of *all* properties expressible in *Ramsey's class.*

2 Preliminaries

Instead of restricting our attention to graphs, we focus on properties of relational structures. We begin by defining vocabularies of such structures.

[⋆] Supported by a Grant-in-Aid for JSPS Fellows under Grant No. 2100195209.

[⋆⋆] Supported by MEXT Grant-in-Aid for Scientific Research on Priority Areas under Grant No. 21013001.

[1] A hereditary property is one that is closed under the taking of induced substructures.

© Springer-Verlag Berlin Heidelberg 2010

Definition 1. *A* vocabulary τ *is a tuple of distinct predicate symbols* $R_i^{a_i}$ *with associated arities* a_i*, i.e.,* $\tau := (R_1^{a_1}, \ldots, R_s^{a_s})$.

The vocabulary (unique up to renaming) of directed graphs is $\tau_G := (E^2)$.

Definition 2. *A* structure A *with vocabulary* τ *is an* $(s+1)$*-tuple,*

$$A := (U, \mathcal{R}_1^A, \ldots, \mathcal{R}_s^A)\,,$$

where U *is a finite universe, and* $\mathcal{R}_i^A$ *is an* a_i*-ary predicate corresponding to the symbol* $R_i^{a_i}$ *of* τ*, i.e.,* $\mathcal{R}_i^A \subseteq U^{a_i}$.

The natural numbers are denoted by $\mathbb{N} := \{0, 1, \ldots\}$. For any set U we write $|U|$ to denote the cardinality of U and we generally identify U with the set $\{0, \ldots, |U|-1\}$. The *size* of A is denoted by $\#(A)$ and defined as $\#(A) := |U|$. Let $STRUC^n(\tau)$ be the set of structures with vocabulary τ and size n, and $STRUC(\tau) := \bigcup_{n\in\mathbb{N}} STRUC^n(\tau)$ be the finite structures with vocabulary τ.

A *property* P with vocabulary τ is any subset of $STRUC(\tau)$. We say that a structure A *has* P if $A \in P$. In property testing, we are interested in distinguishing between structures that have some property and those that are *far* from having the property, and so we require a distance measure. Jordan and Zeugmann [9] introduced several possible distances and considered the relationship between the resulting notions of testability. We are proving a positive result, and so it suffices to use only the most restricted variant considered there.

We begin by noting that relations have *subrelations*, for example monadic loops in a binary predicate. In property testing, it can be useful and is more restrictive (see Jordan and Zeugmann [9]) to consider these subrelations as separate relations when defining the distance between structures. We first define the syntactic notion of *subtype* before proceeding to subrelations.

Definition 3. *A* subtype S *of a predicate symbol* $R_i^{a_i}$ *is any partition of the set* $\{1, \ldots, a_i\}$.

For example, graphs have a single, binary predicate symbol E^2 which has two subtypes: $\{\{1,2\}\}$ and $\{\{1\},\{2\}\}$, corresponding to loops and non-loops respectively. Let $SUB(R_i^{a_i})$ denote the set of subtypes of predicate symbol $R_i^{a_i}$.

Definition 4. *Let* $A \in STRUC(\tau)$ *be a structure with vocabulary* τ *and universe* U*, and let* S *be a subtype of predicate symbol* $R_i^{a_i} \in \tau$*. We define* $s^U(S)$*, the* tuples that belong to S*, as the set* $(x_1, \ldots, x_{a_i}) \in U^{a_i}$ *satisfying the following condition. For every* $1 \le j, k \le a_i$*,* $x_j = x_k$ *iff* j *and* k *are contained in the same element of* S*. The* subrelation $s^A(S)$ *of* A *corresponding to* S *is* $s^A(S) := s^U(S) \cap \mathcal{R}_i^A$.

Returning to our example of graphs, the sets of loops and non-loops are the subrelations of $\mathcal{E}$ corresponding to the subtypes $\{\{1,2\}\}$ and $\{\{1\},\{2\}\}$ of E^2, respectively. We denote the symmetric difference of sets U and V by $U \triangle V$,

$$U \triangle V := (U \backslash V) \cup (V \backslash U)\,.$$

Definition 5. *Let $A, B \in STRUC^n(\tau)$ be structures with vocabulary τ and size n. The* distance *between A and B is*

$$\operatorname{mrdist}(A,B) := \max_{R_i^{a_i} \in \tau} \max_{S \in SUB(R_i^{a_i})} \frac{|s^A(S) \triangle s^B(S)|}{n^{|S|}}.$$

The distance between structures is the fraction of assignments that differ in the most different subtype. The distance between structures generalizes to distance from properties in the usual way.

Definition 6. *Let $A \in STRUC^n(\tau)$ be a structure with vocabulary τ and size n, and let $P \subseteq STRUC(\tau)$ be a property with vocabulary τ. The* distance *between A and P is*

$$\operatorname{mrdist}(A,P) := \min_{B \in P \cap STRUC^n(\tau)} \operatorname{mrdist}(A,B).$$

If $P \cap STRUC^n(\tau)$ is empty, we let the distance be infinite.

We are now able to define property testing itself.

Definition 7. *Let $P \subseteq STRUC(\tau)$ be a property with vocabulary τ. An (ε, q)-*tester *T for P is a probabilistic algorithm which satisfies the following conditions, when given ε and access to an oracle that answers queries for the universe size $\#(A)$ and for the assignment of any tuple $(x_1, \ldots, x_{a_i}) \in \mathcal{R}_i^A$:*

1. *If $A \in P$, then T accepts with probability at least $2/3$.*
2. *If $\operatorname{mrdist}(A,P) \geq \varepsilon$, then T rejects with probability at least $2/3$.*
3. *T makes at most q queries.*

In property testing, we are particularly interested in properties that can be tested with a number of queries depending only on ε.

Definition 8. *A property P is* testable *if there exists a function $c(\varepsilon)$ such that for every $\varepsilon > 0$, there exists an $(\varepsilon, c(\varepsilon))$-tester for P.*

This is a non-uniform definition because we do not require the testers to be, e.g., computable given ε. There exist properties that are non-uniformly testable but not uniformly testable (see, e.g., Alon and Shapira [3]). Our results hold in either case[2] (i.e., one can replace all occurrences of "testable" below by "uniformly testable" and maintain correctness) and so we will not distinguish between them.

We also require logical definitions. These definitions are standard and we review them quickly. See Enderton [6] for an introduction to logic and Börger *et al.* [5] for background on classification problems.

The first-order language of τ is the closure of the atomic formulae $x_i = x_j$ and $R_i^{a_i}(x_1, \ldots, x_{a_i})$ for variable symbols x_k under Boolean connectives $\wedge$, $\vee$ and $\neg$ and first-order quantifiers $\forall$ and $\exists$. We do not allow ordering or arithmetic. These sentences are interpreted in the usual way and so, for a structure $A \in STRUC(\tau)$

[2] In the uniform case, we must restrict Lemma 1 to decidable properties. All properties considered here are clearly decidable.

and sentence φ of vocabulary τ, we can decide whether $A \models \varphi$. Logical sentences define properties; if φ is a sentence with vocabulary τ, then it defines property $P_\varphi := \{A \mid A \in STRUC(\tau), A \models \varphi\}$.

Our logic does not contain arithmetic or ordering, and so all properties expressible in it are closed under isomorphisms. We formalize this as follows.

Definition 9. *Let $A, B \in STRUC^n(\tau)$ be two structures with universe U and vocabulary τ. We say that A and B are* isomorphic *if there exists a bijection $f\colon U \to U$ such that for all $1 \leq i \leq s$ and $x_1, \ldots, x_{a_i} \in U$, we have*

$$(x_1, \ldots, x_{a_i}) \in \mathcal{R}_i^A \iff (f(x_1), \ldots, f(x_{a_i})) \in \mathcal{R}_i^B \, .$$

Definition 10. *Let P be a property with vocabulary τ. We say that P is* closed under isomorphisms *if for all isomorphic A, B, A has P iff B has P.*

Our goal is a classification of the syntactic subclasses of first-order logic according to their testability. These subclasses are traditionally formulated as prefix-vocabulary fragments. Here we are only interested in Ramsey's class, and so we omit more general definitions, see, e.g., Börger *et al.* [5] for details.

Ramsey's class is denoted $[\exists^*\forall^*, all]_=$. This is the set of sentences of first-order predicate logic in prenex normal form, where all existential quantifiers precede all universal quantifiers. Function symbols do not appear, but any number of predicate symbols of any arity may appear, as may the special atomic predicate $=$. Ramsey's class has a number of nice algorithmic properties. For example, Ramsey [12] showed the satisfiability problem for this class is decidable, and Lewis [11] showed it to be NEXPTIME-complete. Kolaitis and Vardi [10] proved a 0-1 law holds for existential *second*-order sentences, if the first-order part is in Ramsey's class. The class that Alon *et al.* [1] proved testable is essentially the restriction of Ramsey's class to graphs, denoted $[\exists^*\forall^*, (0,1)]_=$.

3 Testability of Ramsey's Class

We show that all properties expressible in $[\exists^*\forall^*, all]_=$ are testable. The proof follows that of Alon *et al.* [1]. First, we show that their notion of *indistinguishability* preserves testability after generalizing to relational structures. Then, we prove that all sentences in our class define properties that are indistinguishable from instances of a generalized colorability problem. Finally, we show that all such problems are hereditary and therefore testable in the setting defined by Austin and Tao [4]. This implies testability under our definitions, giving the following.

Theorem 1. *All sentences in $[\exists^*\forall^*, all]_=$ are testable.*

Alon *et al.* [1] introduced the concept of indistinguishability and showed that it preserves testability of graph properties. We begin by extending their notion to relational properties.

Definition 11. *Let $P_1, P_2 \subseteq STRUC(\tau)$ be properties with vocabulary τ that are closed under isomorphisms. We say that P_1 and P_2 are* indistinguishable *if for every $\varepsilon > 0$ there exists an $N := N(\varepsilon) \in \mathbb{N}$ such that the following holds for all $n > N$. For every $A \in STRUC^n(\tau)$, if A has property P_1, then* $\mathrm{mrdist}(A, P_2) < \varepsilon$ *and if A has P_2, then* $\mathrm{mrdist}(A, P_1) < \varepsilon$.

The importance of indistinguishability is that it preserves testability.

Lemma 1. *Let $P_1, P_2 \subseteq STRUC(\tau)$ be indistinguishable properties with vocabulary τ. Property P_1 is testable iff P_2 is testable.*

The proof of Lemma 1 is a simple extension of the proof by Alon *et al.* [1] and is omitted due to space constraints. Next, we will show that all sentences in $[\exists^*\forall^*, all]_=$ define properties that are indistinguishable from instances of a generalized colorability problem. We begin by defining the colorability problem.

For any fixed set F of structures with vocabulary τ, some positive number of colors c, and functions that assign a color between 1 and c to each element of each structure in F, we define the F-colorability problem as follows. A structure $A \in STRUC(\tau)$ is F-colorable if there exists some (not necessarily proper) c-coloring of A such that A does not contain any induced substructures isomorphic to a member of F. We let P_F be the set of structures that are F-colorable.

For example, we can consider the case of graphs and let F contain c copies of K_2. We enumerate these copies in some fashion from 1 to c, and for copy i, color both vertices with i. The resulting problem is of course the usual (k- or equivalently) c-colorability. The following is a straightforward generalization of the proof by Alon *et al.* [1].

Lemma 2. *Let φ be any first-order sentence in the class $[\exists^*\forall^*, all]_=$. There exists an instance of the F-colorability problem that is indistinguishable from P_φ, the property defined by φ.*

Proof. Let $\varepsilon > 0$ be arbitrary and $\varphi := \exists x_1 \ldots \exists x_t \forall y_1 \ldots \forall y_u : \psi$ be any first-order formula with quantifier-free ψ and vocabulary τ. We note, as did Alon *et al.* [1], that we can restrict our attention to formulas ψ where it is sufficient to consider only cases where the variables are bound to distinct elements. This is because, given any ψ', we can construct a ψ satisfying this restriction that is equivalent on structures with at least $t + u$ elements, and the smaller structures do not matter in the context of indistinguishability.

Let $P = \{A \mid A \in STRUC(\tau), A \models \varphi\}$ be the property defined by φ. We now define an instance of F-colorability that we will show to be indistinguishable from P. We denote our c colors by the elements of

$$\{(0,0)\} \cup \{(a,b) \mid 1 \le a \le \pi_1, 1 \le b \le \pi_2, a, b \in \mathbb{N}\}\,.$$

Here, π_1 is the number of distinct structures of vocabulary τ with exactly t elements, $\pi_1 := 2^{\sum_{1\le i\le s} t^{a_i}}$. Similarly, we denote by π_2 the number of ways it is possible to "connect" or "add" a single element to some existing, fixed t-element structure of vocabulary τ, i.e., $\pi_2 := 2^{\sum_{1\le i\le s}\sum_{1\le j\le a_i-1}\binom{a_i}{j} t^{a_i-j}}$. We will use fixed enumerations of these π_1 structures with t elements and π_2 ways of connecting an additional element to a fixed t element structure.

We impose on the coloring of the structure the following restrictions. Each can be expressed by prohibiting finite sets of colored induced substructures.

(1) The color $(0,0)$ may be used at most t times. Therefore, we prohibit all $(t+1)$-element structures that are colored completely with $(0,0)$.
(2) The graph must be colored using only $\{(0,0)\} \cup \{(a,b) \mid 1 \leq b \leq \pi_2\}$ for some fixed $a \in \{1,\ldots,\pi_1\}$. Therefore, we prohibit all two-element structures colored $((a,b),(a',b'))$ with $a \neq a'$.
(3) We now consider some fixed coloring of a u-element structure V, whose universe we identify with $\{v_1,\ldots,v_u\}$. We assume that this coloring satisfies the previous restriction and that color $(0,0)$ does not appear. We must decide whether to prohibit this structure. In order to do so, we first take the fixed a guaranteed by the previous restriction, and consider the t-element structure E, whose universe we identify with $\{e_1,\ldots,e_t\}$, that is the a^{th} structure in our enumeration of t element structures. We connect each v_i to E in the following way. If v_i is colored (a,b), we use the b^{th} way of connecting an additional element to a t-element structure in our enumeration. We denote the resulting $(t+u)$-element structure as M and allow (do not prohibit) M iff M is a model of ψ when we replace x_i with u_i and y_j with v_j.

We now show that the resulting F-colorability problem is indistinguishable from P. Assume that we are given an $A \models \varphi$. Color the t vertices existentially bound to the x_i with $(0,0)$. Then, we can color all remaining vertices v_i with (a,b), where a corresponds to the substructure induced by $\{x_1,\ldots,x_t\}$ in our enumeration of t-element structures, and b corresponds to the connection between v_i and $\{x_1,\ldots,x_t\}$. It is easy to see that this coloring satisfies the restrictions of our F-colorability problem. We have not made any modifications to the structure and so $\mathrm{mrdist}(A, P_F) = 0$.

Next, we assume that we are given a structure with a coloring that satisfies our restrictions. We will show that we can obtain a model of φ by making only a small number of modifications. First, if there are less than t elements colored $(0,0)$, we arbitrarily choose additional elements to color $(0,0)$ so that there are exactly t such elements. We will denote these t elements with $\{e_1,\ldots,e_t\}$. Restriction (2) guarantees that all colors which are not $(0,0)$ share the same first component. Let a be this shared component. We make the structure induced by $\{e_1,\ldots,e_t\}$ identical to the a^{th} structure in our enumeration of t-element structures, requiring at most $\sum_{1\leq i\leq s} t^{a_i} = O(1)$ modifications. Next, for each element v_i that is colored (a,b) with $a,b \neq 0$, we modify the connections between v_i and $\{e_1,\ldots,e_t\}$ in order to make these connections identical to the b^{th} way of making such connections in our enumeration. This requires at most

$$(n-t)\sum_{1\leq i\leq r}\sum_{1\leq j\leq a_i-1}\left[\binom{a_i}{j}t^{a_i-j}\right] = O(n)$$

additional modifications, all of which are to non-monadic subrelations. Binding x_i to e_i, the resulting structure is a model of φ. We made at most $O(1)$ modifications to monadic subrelations and $O(n)$ modifications to non-monadic subrelations, and so $\mathrm{mrdist}(A, P) \leq \max\{O(1)/n, O(n)/\Omega(n^2)\} = o(1) < \varepsilon$, where the inequality holds for sufficiently large n.

Therefore, all such properties P are indistinguishable from instances of F-colorability. $\square$

A *hereditary property* of relational structures is one which is closed under taking induced substructures. F-colorability is clearly a hereditary property; if A is F-colorable, then so are its induced substructures. However, the definitions of Austin and Tao [4] are significantly different from ours and so we explicitly reduce the following translation in our setting to their result.

Theorem 2 (Translation of Austin and Tao [4]). *Let P be a hereditary property of relational structures which is closed under isomorphisms. Then, property P is testable with one-sided error.*

Before reducing Theorem 2 to its statement in [4], we first briefly introduce their definitions. All of the definitions in Subsection 3.1 are from Austin and Tao [4], although we omit definitions which are not necessary for our purposes.

3.1 Framework of Austin and Tao [4]

We begin by introducing their analogue of vocabularies: *finite palettes.*

Definition 12. *A* finite palette *K is a sequence $K := (K_j)_{j=0}^{\infty}$ of finite sets, of which all but finitely-many are singletons. The singletons are called* points *and denoted* pt. *A point is called* trailing *if it occurs after all non-points.*

We will write $K = (K_0, \ldots, K_k)$, omitting trailing points and call k the *order* of K. We use the elements of K_j to *color* the j-ary edges in hypergraphs.

Definition 13. *A* vertex set *V is any set which is at most countable. If V, W are vertex sets, then a* morphism *f from W to V is any injective map $f \colon W \to V$ and the set of such morphisms is denoted* $\mathrm{Inj}(W, V)$. *For $N \in \mathbb{N}$, we denote the set $\{1, \ldots, N\}$ by $[N]$.*

Of course, $[N]$ is a vertex set. Our structures are finite so we are mostly interested in *finite* vertex sets. Next, we define the analogue of relational structures.

Definition 14. *Let V be a vertex set and K be a finite palette. A K-*colored hypergraph *G on V is a sequence $G := (G)_{j=0}^{\infty}$, where each $G_j \colon \mathrm{Inj}([j], V) \to K_j$ is a function. Let $K^{(V)}$ be the set of K-colored hypergraphs on V.*

Only finitely many of the K_j are not points, and so only finitely many G_j are non-trivial. The G_j assign colors from K_j to the morphisms in $\mathrm{Inj}([j], V)$. In our relational setting, this set of morphisms corresponds to the set of j-ary tuples $(x_1, \ldots, x_j)$ with pairwise distinct components.

Before defining hereditary K-properties, we need one last technical definition.

Definition 15. *Let V, W be vertex sets and $f \in \operatorname{Inj}(W, V)$ be a morphism from W to V. The* pullback map $K^{(f)}\colon K^{(V)} \to K^{(W)}$ *is*

$$\left(K^{(f)}(G)\right)_j (g) := G_j(f \circ g)\,,$$

for all $G = (G_j)_{j=0}^{\infty} \in K^{(V)}$, $j \geq 0$ and $g \in \operatorname{Inj}([j], W))$. If $W \subseteq V$ and $f \in \operatorname{Inj}(W, V)$ is the identity map on W, we abbreviate

$$G \downharpoonleft_W := K^{(f)}\,.$$

Abusing notation, the pullback map $K^{(f)}$ maps K-colored hypergraphs on V to those on W, by assigning the color of $f \circ g$ to g, for all tuples g. Note that $G \downharpoonleft_W$ is equivalent to the induced subhypergraph on W. For notational clarity, we reserve P for properties of relational structures and use $\mathcal{P}$ to denote properties of hypergraphs.

Definition 16. *Let $K = (K_j)_{j=0}^{\infty}$ be a finite palette. A* hereditary K-property *$\mathcal{P}$ is an assignment $\mathcal{P}\colon V \mapsto \mathcal{P}^{(V)}$ of a collection $\mathcal{P}^{(V)} \subseteq K^{(V)}$ of K-colored hypergraphs for every finite vertex set V such that*

$$K^{(f)}(\mathcal{P}^{(V)}) \subseteq \mathcal{P}^{(W)}$$

for every morphism $f \in \operatorname{Inj}(W, V)$ between finite vertex sets.

Finally, we state the definition of (one-sided error) testability used by Austin and Tao [4]. Here, for a vertex set V and $c \in \mathbb{N}$, we write $\binom{V}{c} := \{V' \mid V' \subseteq V, |V'| = c\}$ to denote the set of subsets of V with exactly c elements.

Definition 17. *Let K be a finite palette with order $k \geq 0$ and $\mathcal{P}$ be a hereditary K-property. Property $\mathcal{P}$ is* testable with one-sided error *if for every $\varepsilon > 0$, there exists $N \geq 1$ and $\delta > 0$ satisfying the following. For all vertex sets V with $|V| \geq N$, if $G \in K^{(V)}$ satisfies*

$$\frac{1}{\left|\binom{V}{N}\right|} \left|\left\{W \mid W \in \binom{V}{N}, G \downharpoonleft_W \in \mathcal{P}^{(W)}\right\}\right| \geq 1 - \delta\,, \tag{1}$$

then there exists a $G' \in \mathcal{P}^{(V)}$ satisfying

$$\frac{1}{\left|\binom{V}{k}\right|} \left|\left\{W \mid W \in \binom{V}{k}, G \downharpoonleft_W \neq G' \downharpoonleft_W\right\}\right| \leq \varepsilon\,. \tag{2}$$

To see that this is a variant of testability, it is easiest to consider the contrapositive. If there is a G' satisfying (2), then G is not ε-far from $\mathcal{P}$, using the implicit distance measure based on the fraction of differing induced subhypergraphs of size k. If there is no such G' (i.e., G is ε-far from $\mathcal{P}$) and $\mathcal{P}$ is testable, then (1) must not hold. That is, there are *many* induced subhypergraphs of size N that do not have $\mathcal{P}$. The definition is for hereditary $\mathcal{P}$, and so if G has $\mathcal{P}$, then so do all induced subhypergraphs. This allows the construction of testers.

Finally, we can state one of the main results of Austin and Tao [4].

Theorem 3 (Austin and Tao [4]). *Let K be a finite palette and let $\mathcal{P}$ be a hereditary K-property. Then, $\mathcal{P}$ is testable with one-sided error.*

In the following subsection we will map our vocabularies, structures and properties to this setting. We will then show that hereditary properties in our setting correspond to hereditary properties (in the sense of Definition 16) here, and that testability in the sense of Definition 17 implies testability of the original relational properties. That is, we explicitly reduce Theorem 2 to Theorem 3.

3.2 Reducing Theorem 2 to Theorem 3

We begin by mapping vocabulary $\tau = \{R_1^{a_1}, \ldots, R_s^{a_s}\}$ to a finite palette $K_\tau = (K_i)_{i=0}^{\infty}$. We use the color of a "tuple" to represent the set of assignments on it. The difference between the set of j-ary tuples over a finite universe U and $\mathrm{Inj}([j], U)$ is that the latter does not permit repeated components. If $S \in SUB(R_i^{a_i})$ has $|S| < a_i$, then the corresponding subrelation consists of tuples with repeated components. We treat such S as relations with arity $|S|$ and no repeated components. Here, $\mathfrak{S}(n, k)$ is the Stirling number of the second kind.

For $a \geq 1$, let $P_a := \{R_i^{a_i} \mid R_i^{a_i} \in \tau, a_i = a\}$ be the set of predicate symbols with arity a. We now define palette K. Let $K_0 :=$ pt and $K_i := \left[2^{\sum_{j\geq i} |P_j| \mathfrak{S}(j,i)}\right]$. There are finitely-many predicate symbols and so only finitely-many $K_i \neq$ pt.

Let $S_a := \{S_a^i \mid S_a^i \in SUB(R_i^{a_i}), |S_a^i| = a, 1 \leq i \leq s\}$ be the set of subtypes with cardinality a for all $a \geq 1$. Now, $2^{|S_a|} = |K_a|$ and we have exactly enough colors to encode the set of assignments of the a-ary subtypes on a-ary tuples.

We will now define a map h from relational structures A on universe U to hypergraphs $G_A \in K^{(U)}$. For any $S_a^i \in S_a$, there is a bijection

$$r(S_a^i) \colon s^U(S_a^i) \to \{(x_1, \ldots, x_a) \mid x_i \in U, x_i \neq x_j \text{ for } i \neq j\}$$

from $s^U(S_a^i)$ to the a-ary tuples without duplicate components, formed by removing the duplicate components. That is, $r(S_a^i)$ maps $(x_1, \ldots, x_{a_i})$ to $(x_{i_1}, \ldots, x_{i_a})$ where $1 \leq i_1 < i_2 < \ldots < i_a \leq a_i$. We can now define $G_A = h(A)$.

For $j > 0$, we define $G_j \colon \mathrm{Inj}([j], U) \to K_j$ as follows. Assign to $f \in \mathrm{Inj}([j], U)$ the color encoding the set of assignments of the subtypes S_j on $(f(1), \ldots, f(j))$, using the inverses $(r(S_j^i))^{-1}$ to get assignments for subtypes of high-arity relations. For $j = 0$, $\mathrm{Inj}([j], U) = \emptyset$ and $K_0 =$ pt and we can use a trivial map.

Of course, we extend the map to properties in the obvious way. If P is a property of relational structures, we let $\mathcal{P}^{(U)} := \{h(A) \mid A \in P\}$. Formally, we define $\mathcal{P}(U) := \mathcal{P}^{(U)}$, but there is a small technical point. We have identified finite universes with subsets of the naturals, allowing us to call $STRUC(\tau)$ a *set*. However, Definition 13 allows a vertex set to be *any* finite set and Definition 16 requires hereditary hypergraph properties to be closed under bijections between vertex sets. To remedy this, for each finite vertex set W, we fix a[3] bijection $g^W : W \to \{0, \ldots, |W| - 1\}$. We then define $\mathcal{P} := h(P)$ formally as

[3] Our properties are closed under isomorphisms, so any fixed bijection is acceptable.

$$\mathcal{P}(W) := \begin{cases} \mathcal{P}^{(W)}, & \text{if } W = \{0,1,\ldots,|W|-1\}; \\ K^{\left(g^W\right)}\left(\mathcal{P}^{(\{0,\ldots,|W|-1\})}\right), & \text{otherwise.} \end{cases}$$

Hereditary relational properties are mapped to hereditary hypergraph properties, which are testable in the sense of Definition 17 by Theorem 3.

Lemma 3. *If P is a hereditary property of relational structures, then $h(P)$ is a hereditary property of hypergraphs.*

Proof. Let P be a hereditary property of relational structures with vocabulary τ. Assume that $\mathcal{P} := h(P)$ does not satisfy Definition 16. Then, there exist finite vertex sets V and W, and a morphism $f' \in \operatorname{Inj}(W,V)$ such that

$$K^{(f)}(\mathcal{P}^{(V)}) \not\subseteq \mathcal{P}^{(W)} . \tag{3}$$

Since f' exists, $\operatorname{Inj}(W,V)$ cannot be the empty set and so $|V| \geq |W|$. Let $U_V := \{0,\ldots,|V|-1\}$ and $U_W := \{0,\ldots,|W|-1\}$. By the definition of $\mathcal{P}$, we can fix bijections $g^V : V \to U_V$ and $g^W : W \to U_W$ such that $\mathcal{P}^{(V)} = K^{\left(g^V\right)}\left(\mathcal{P}^{(U_V)}\right)$ and $\mathcal{P}^{(W)} = K^{\left(g^W\right)}\left(\mathcal{P}^{(U_W)}\right)$. By the definition of $\mathcal{P} = h(P)$, this implies

$$K^{(f)}\left(K^{\left(g^V\right)}\left(\mathcal{P}^{(U_V)}\right)\right) \not\subseteq K^{\left(g^W\right)}\left(\mathcal{P}^{(U_W)}\right) .$$

Bijections are invertible, and so this implies

$$K^{\left(g^V \circ f \circ \left(g^W\right)^{-1}\right)}\left(\mathcal{P}^{(U_V)}\right) \not\subseteq \mathcal{P}^{(U_W)} .$$

Rename $f' := g^V \circ f \circ \left(g^W\right)^{-1}$ and note $f' \in \operatorname{Inj}(U_W, U_V)$. Let $A' \in \mathcal{P}^{(U_V)}$ be such that $K^{(f')}(A') \notin \mathcal{P}^{(U_W)}$.

We defined $\mathcal{P}$ as $h(P)$ for a hereditary property P of relational structures. Property P is closed under isomorphisms, and so there is an $A := h^{-1}(A') \in P \cap STRUC^{|U_V|}(\tau)$ such that the $|U_W|$-element substructure induced by $\{a \mid a = f'(u) \text{ for some } u \in U_w\}$ does not have P. This contradicts the hereditariness of P and so $\mathcal{P}$ must be hereditary in the sense of Definition 16. □

We mapped hereditary relational properties to hereditary hypergraph properties, which are testable by Theorem 3. We will show this implies testability of the original properties.

Definition 18. *Let $A, B \in STRUC^n(\tau)$ be structures with vocabulary τ and universe $U := \{0,\ldots,n-1\}$ of size n, $k := \max_i a_i$ be the maximum arity of the predicate symbols, and $h\colon STRUC^n(\tau) \to K^{(U)}$ be the map defined above. The* h-distance *between A and B is*

$$\operatorname{hdist}(A,B) := \frac{1}{\left|\binom{U}{k}\right|}\left|\left\{W \mid W \in \binom{U}{k}, h(A)\downharpoonright_W \neq h(B)\downharpoonright_W\right\}\right| .$$

We now relate the two distances with the following simple lemma.

Lemma 4. *Let $A, B \in STRUC^n(\tau)$ be relational structures with vocabulary τ and size n. Then,* $\text{hdist}(A, B) \geq \text{mrdist}(A, B)$.

Proof. Assume that $\text{mrdist}(A, B) = \varepsilon$. Then, there exists a predicate symbol $R_i^{a_i} \in \tau$ and subtype $S \in SUB(R_i^{a_i})$ such that $\left|s^A(S) \triangle s^B(S)\right| / n^{|S|} = \varepsilon$. Let $k := \max_i a_i$ and let the universe of both structures be $U_n := \{0, \ldots, n-1\}$.

Consider a random permutation of the universe (i.e., a bijection $r\colon U_n \to U_n$) chosen uniformly from the set of such permutations. The probability that the substructures induced on $\{r(0), \ldots, r(k-1)\}$ differ in A and B is $\text{hdist}(A, B)$. The probability that the first $|S|$ elements, i.e. $\{r(0), \ldots, r(|S|-1)\}$, differ in $s^A(S)$ and $s^B(S)$ is ε and so $\text{hdist}(A, B) \geq \varepsilon$. □

Equality is obtained when $|S| = k$. It is possible to show that the two distances differ by at most a constant factor, and so the corresponding notions of testability are essentially equivalent. However, Lemma 4 suffices for our purposes.

Lemma 5. *Let $P \subseteq STRUC(\tau)$ be a property of relational structures which is mapped by h to a property of hypergraphs that is testable with one-sided error. Then, P is testable with one-sided error.*

Proof. Let $\mathcal{P} := h(P)$ be the hypergraph property which P is mapped to. We will show that the following is an ε-tester for P with one-sided error. Let $N \geq 1$, $\delta > 0$ be the constants of Definition 17 for ε. Assume that we are testing a structure $A \in STRUC^n(\tau)$ and recall that $U = \{0, \ldots, n-1\}$.

1. If $\#(A) \leq N$, query the entire structure and decide exactly whether $A \in P$.
2. Otherwise, repeat the following $q(\delta)$ times.
   (a) Uniformly select N elements and query the induced substructure.
   (b) If it has P, continue. Otherwise, reject.
3. Accept if all of the induced substructures had P.

If $A \in P$, then all induced substructures have P because P is hereditary and the tester accepts with probability 1. Next, assume $\text{mrdist}(A, P) > \varepsilon$. We use Definition 17 to show the tester will find a witness for $A \notin P$ with probability at least 2/3. By Lemma 4, $\text{hdist}(A, P) \geq \text{mrdist}(A, P) > \varepsilon$. We assumed $h(P)$ is hereditary, and so (by Theorem 3) it is testable in the sense of Definition 17. The probability that a uniformly chosen N-element substructure does not have P is at least δ. We use $q(\delta)$ to amplify the success probability from δ to 2/3. □

This completes the proof of Theorem 1. All properties expressible in Ramsey's class are indistinguishable from instances of F-colorability. Indistinguishability preserves testability and so it sufficed to show that these instances are testable. All instances of F-colorability are hereditary relational properties, which are testable by Theorem 2, which we reduced to the statement by Austin and Tao [4].

4 Conclusion

We have revisited the positive result obtained by Alon *et al.* [1] in the light of a strong result obtained recently by Austin and Tao [4]. We have shown that this allows us to extend the proof that properties expressible in Ramsey's class are testable, from undirected, loop-free graphs to arbitrary relational structures.

A more direct proof for the testability of Ramsey's class would be interesting, especially if it results in better query complexity. It would also be interesting to consider the testability of additional prefix classes and connections with other classifications (such as, e.g., that for the finite model property).

References

1. Alon, N., Fischer, E., Krivelevich, M., Szegedy, M.: Efficient testing of large graphs. Combinatorica 20(4), 451–476 (2000)
2. Alon, N., Shapira, A.: A characterization of the (natural) graph properties testable with one-sided error. SIAM J. Comput. 37(6), 1703–1727 (2008)
3. Alon, N., Shapira, A.: A separation theorem in property testing. Combinatorica 28(3), 261–281 (2008)
4. Austin, T., Tao, T.: On the testability and repair of hereditary hypergraph properties. Available at arXiv.org as arXiv:0801.2179v2 (2009) (preprint)
5. Börger, E., Grädel, E., Gurevich, Y.: The Classical Decision Problem. Springer, Heidelberg (1997)
6. Enderton, H.B.: A Mathematical Introduction to Logic, 2nd edn. Academic Press, London (2000)
7. Fischer, E.: Testing graphs for colorability properties. In: Proceedings, 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 873–882 (2001)
8. Fischer, E.: Testing graphs for colorability properties. Random Struct. Algorithms 26(3), 289–309 (2005)
9. Jordan, C., Zeugmann, T.: Relational properties expressible with one universal quantifier are testable. In: Watanabe, O., Zeugmann, T. (eds.) SAGA 2009. LNCS, vol. 5792, pp. 141–155. Springer, Heidelberg (2009)
10. Kolaitis, P., Vardi, M.: The decision problem for the probabilities of higher-order properties. In: STOC 1987: Proceedings of the 19th Annual ACM Symposium on Theory of Computing, pp. 425–435. ACM, New York (1987)
11. Lewis, H.R.: Complexity results for classes of quantificational formulas. J. Comput. Syst. Sci. 21(3), 317–353 (1980)
12. Ramsey, F.P.: On a problem of formal logic. Proc. London Math. Soc. 30(2), 264–286 (1930)
13. Rödl, V., Schacht, M.: Property testing in hypergraphs and the removal lemma. In: STOC 2007: Proceedings of the 39th Annual ACM Symposium on Theory of Computing, pp. 488–495. ACM, New York (2007)

Deterministic Polynomial-Time Algorithms for Designing Short DNA Words

Ming-Yang Kao[1], Henry C.M. Leung[2], He Sun[3,4,⋆], and Yong Zhang[2]

[1] Department of Electrical Engineering and Computer Science
Northwestern Univeristy, USA
[2] Department of Computer Science, The University of Hong Kong, China
[3] Max-Planck-Institut für Informatik, Germany
[4] Fudan University, China

Abstract. Designing short DNA words is a problem of constructing n DNA strings (words) with the minimum length such that the Hamming distance between each pair is at least k and the words satisfy a set of extra constraints. This problem has applications in DNA computing, DNA self-assembly, and DNA arrays. Previous works include those that extended results from coding theory to obtain bounds on code size for biologically motivated constraints and those that applied heuristic local searches, genetic algorithms, and randomized algorithms. In particular, Kao, Sanghi and Schweller developed polynomial-time randomized algorithms to construct n DNA words of length $9 \cdot \max\{\log n, k\}$ satisfying a sets of constraints with high probability. In this paper, we give deterministic polynomial-time algorithms to construct DNA words based on expander codes, Ramanujan graphs, and derandomization techniques. Our algorithms can construct n DNA words of length $\max\{3\log n, 4k\}$ or $2.1\log n + 6.28k$ satisfying the same sets of constraints as the words constructed by the algorithms of Kao et al. We have also extended these algorithms to construct words that satisfy a larger set of constraints for which the algorithms of Kao et al. do not work.

Keywords: DNA computating, DNA words design, coding theory, expander codes, Ramanujan graphs, derandomization.

1 Introduction

Building on the work of Kao, Sanghi, and Schweller [17], this paper considers the problem of designing sets (codes) of DNA strings (words) of close to optimal length that satisfy certain combinatorial constraints. Many applications depend on the scalable design of such words. For instance, DNA words can be used to store information at the molecular level [8], to act as molecular bar codes for identifying molecules in complex libraries [8, 9, 21], or to implement DNA arrays [5]. For DNA computing, inputs to computational problems are encoded into DNA strands to perform computing via complementary binding [1, 27]. For DNA self-assembly, Wang tile self-assembly systems are implemented by encoding glues of Wang tiles into DNA strands [26–28].

⋆ Corresponding author.

© Springer-Verlag Berlin Heidelberg 2010

A set of DNA words chosen for such applications must meet certain combinatorial constraints. First of all, hybridization must not occur between any two different words in the set, or even between a word and the reverse complement of any other word in the set. For such requirements, Marathe et al. [18] proposed the *basic Hamming constraint* (C_1), the *reverse complement Hamming constraint* (C_2), and the *self-complementary constraint* (C_3). Kao et al. [17] further considered certain more restricting *shifting* versions of these constraints (C_4, C_5, C_6) which require them to hold between all alignments of any pair of words [7].

Kao et al. [17] also considered three constraints unrelated to Hamming distance. The *consecutive base constraint* (C_8) limits the length of any run of identical bases in any given word. Long runs of identical bases are undesirable because they can cause hybridization errors [6, 7, 23]. The *GC content constraint* (C_7) requires that a fixed percentage of the bases in any given word are either G or C. This constraint gives the strings similar thermodynamic properties [23–25]. The *free energy constraint* (C_9) requires that the difference in the free energies of any two words is bounded by a small constant. This constraint helps ensure that the words in the set have similar melting temperatures [7, 18].

Furthermore, it is desirable for the length ℓ of each word to be as small as possible. The motivation for minimizing ℓ is evident in part becasue it is more difficult to synthesize longer DNA strands. Also, longer DNA strands require more DNA to be used for the respective application.

There has been much previous work in the design of DNA words [7, 8, 11–14, 16, 18, 21, 24, 25]. Most of the existing work in this area was based on heuristics, genetic algorithms, or stochastic local searches that did not provide analytical performance guarantees. Notable exceptions include the work of Marathe et al. [18] that extended results from coding theory to obtain bounds on code size for biologically motivated constraints. Also, Kao et al. [17] formulated an optimization problem that takes as input a desired number of words n and produces n words of length ℓ that satisfy a specified set of constraints, while minimizing the length ℓ. They introduced randomized algorithms that run in polynomial time to construct words whose length is within a constant factor of the optimal word length. However, with a non-negligible probability, the constructed words do not satisfy the given constraint. The results of Kao et al. [17] are summarized in Table 1 for comparison with ours.

In this paper, we present two deterministic algorithms for constructing words of length within a constant factor of the optimal word length. Both algorithms run in polynomial time and can construct words that satisfy more constraints than the work of Kao et al. [17] can. Between these two algorithms, the first algorithm constructs shorter words when the Hamming distance k required is large and the second algorithm constructs shorter codewords when the number of words n is large. The first algorithm uses explicitly constructible Ramanujan graphs and expander-based linear codes. Though there is a bulk of research on expander-based codes such as [15, 22], by focusing on the coding efficiency and our specific problem of short DNA words design, we obtain linear codes and expander-based codes with better parameters than available in the literature.

Our second algorithm derandomizates the randomized algorithms of Kao et al. [17]. Depending on the values of k and n, different parameters of derandomization can be applied to minimize the length of words. Our derandomization techniques are general, flexible and potentially applicable to other classes of codes with the following properties. (1) The words have polynomially large densities. (2) The words are uniformly distributed in some manner. (3) The words have polynomial-time computable densities. As shown in Table 1, both of our algorithms can be used to construct words shorter than those constructed by the randomized algorithms of Kao et al. [17].

Table 1. Comparison of word lengths. Here, ℓ is the length of words and k is the maximum of the parameters associated with different constraints. The definitions of the constraints are given in Section 2. The parameter ϵ tends to 0 when n is large enough. Also, $c_1 = 2 + \delta, \delta \in \mathbb{R}^+$ and c_2 is determined according to c_1 as stated in the equation $c_2 = c_1/2(\log c_1/\log(2(c_1 - 2)) + 2.5 - 1/\ln 2)$. For example, for $\delta = 0.1$, $\ell = 2.1 \log n + 6.28k$ and for $\delta = 1$, $\ell = 3 \log n + 4.76k$.

Codes	Randomized Algorithms [16]	Expander-Based Algorithms (this paper)	Derandomization-Based Algorithms (this paper)
$\mathcal{C}_{1,4}$		$\ell^\star = \max\{(3+\epsilon)\log n, (4+\epsilon)k\}$	$\ell^\star = c_1 \log n + c_2 k$
$\mathcal{C}_{1\sim6}$	$\ell = 9\max\{\log n, k\}$	$\ell = \ell^\star + k$	
$\mathcal{C}_{1\sim7}$	$\ell = 10\max\{\log n, k\}$	$\ell = \ell^\star + 2k$	
$\mathcal{C}_{1\sim3,7,8}$	$\ell = 10\frac{d+1}{d}\max\{\log n, k\}$	$\ell = \frac{d}{d-1}(\ell^\star + 2k)$	
$\mathcal{C}_{1\sim8}$		$\ell = \ell^\star + 2k$ for $\frac{1}{d+1} \le \gamma \le \frac{d}{d+1}$ $\ell = \frac{d}{d-1}\ell^\star + \frac{d}{2}k + 2$ $\ell = \ell^\star + \frac{k}{2} + \sqrt{2k\ell^\star} + 2$ when $d \ge \sqrt{2\ell^\star/k} + 1$	
$\mathcal{C}_{1\sim6,9}$	$\ell = 27\max\{\log n, k\}$ when $\sigma \ge 4D + \Gamma_{\max}$	$\ell = 3\ell^\star + 2k$ when $\sigma \ge 4D + \Gamma_{\max}$	

The remainder of this paper is organized as follows. Section 2 gives some basic notations, the nine constraints of short DNA words, and some necessary background for designing our algorithms. Section 3 and 4 detail how to design a short DNA code $\mathcal{C}_{1,4}$ satisfying C_1 and C_4, using two different approaches. In Section 5, we detail how to construct short DNA words under a series of constraints.

2 Preliminaries

Let $X = x_1 \cdots x_\ell$ be a word where x_i belongs to some alphabet Π. For any word X, let the *reverse* of X, denoted X^R, is the word $x_\ell x_{\ell-1} \cdots x_1$, and the *complement* of X is $x_1^c \cdots x_\ell^c$, where for the binary alphabet $\Pi_B = \{0, 1\}$, $0^c = 1$ and $1^c = 0$, and for the DNA alphabet $\Pi_D = \{\mathrm{A, C, G, T}\}$, $\mathrm{A}^c = \mathrm{T}, \mathrm{C}^c = \mathrm{G}, \mathrm{G}^c = \mathrm{C}$, and $\mathrm{T}^c = \mathrm{A}$. The Hamming distance between two words X and Y, denoted by $H(X, Y)$, is the number of positions where X differs from Y.

Next we review the nine constraints as defined in [17]. $\mathcal{W}$ is defined as a codeword set.

1. **Basic Hamming Constraint $C_1(k_1)$:** For any distinct words $Y, X \in \mathcal{W}$, $H(Y, X) \geq k_1$. This constraint limits non-specific hybridization between the Watson-Crick complement of some word Y with a distinct word X.
2. **Reverse Complementary Constraint $C_2(k_2)$:** For any distinct words $Y, X \in \mathcal{W}$, $H\left(Y, X^{RC}\right) \geq k_2$. This constraint limits hybridization between a word and the reverse of another word.
3. **Self Complementary Constraint $C_3(k_3)$:** For any word Y, $H\left(Y, Y^{RC}\right) \geq k_3$. This constraint prevents a word from hybridizing with itself.
4. **Shifting Hamming Constraint $C_4(k_4)$:** For any two distinct words $Y, X \in \mathcal{W}$, $H\left(Y[1 \cdots i], X[(\ell - i + 1) \cdots \ell]\right) \geq k_4 - (\ell - i)$ for all $\ell \geq i \geq \ell - k_4$.
5. **Shifting Reverse Complementary Constraint $C_5(k_5)$:** For any two distinct words $Y, X \in \mathcal{W}$,
$$H\left(Y[1 \cdots i], X[1 \cdots i]^{RC}\right) \geq k_5 - (\ell - i) \text{ for all } i; \text{ and}$$
$$H\left(Y[(\ell - i + 1) \cdots \ell], X[(\ell - i + 1) \cdots \ell]^{RC}\right) \geq k_5 - (\ell - i) \text{ for all } i.$$
6. **Shifting Self Complementary Constraint $C_6(k_6)$:** For any word $Y \in \mathcal{W}$,
$$H(Y[1 \cdots i], Y[1 \cdots i]^{RC}) \geq k_6 - (\ell - i) \text{ for all } i; \text{ and}$$
$$H\left(Y[(\ell - i + 1) \cdots \ell], Y[(\ell - i + 1) \cdots \ell]^{RC}\right) \geq k_6 - (\ell - i) \text{ for all } i.$$
7. **GC Content Constraint $C_7(\gamma)$:** γ percentage of bases in any word $Y \in \mathcal{W}$ are either G or C. The GC content affects the thermodynamic properties of a word. Therefore, having the same ratio of GC content for all the words ensures similar thermodynamic characteristics.
8. **Consecutive Base Constraint $C_8(d)$:** No word has more than d consecutive bases for $d \geq 2$. In some applications, consecutive occurrences (also known as runs) of the same base increase annealing errors.
9. **Free Energy Constraint $C_9(\sigma)$:** For any two distinct words $Y, X \in \mathcal{W}$, $\mathrm{FE}(Y) - \mathrm{FE}(X) \leq \sigma$ where $\mathrm{FE}(W)$ denotes the free energy of a word. This constraint ensures that the words in the set have similar melting temperatures which allows multiple DNA strands to hybridize simultaneously.

The *Hamming distance of code* $\mathcal{C}$ is defined as $\min_{X \neq Y, X, Y \in \mathcal{C}} H(X, Y)$. We call a code $\mathcal{C}$ that maps k bits to n bits with Hamming distance d as an (n, k, d)-code. The *rate* of code $\mathcal{C}$ is defined as k/n.

A binary (n, k, d)-code $\mathcal{C}$ is called *linear* if $0^n \in \mathcal{C}$ and for all x and y in $\mathcal{C}$, $x \oplus y$, where $\oplus$ refers to the bitwise xor operation. Observe that the Hamming distance of $\mathcal{C}$ equals the smallest weight, i.e., the number of nonzero entries of a nonzero word.

For a d-regular graph G, let M be the normalized adjacency matrix of G. Let $\lambda_1 \geq \cdots \geq \lambda_n$ be the spectrum of graph G. Since G is a regular graph, $\lambda_1 = 1$ is associated with eigenvector $(1/n, \cdots, 1/n)$. A graph G is said to have *spectral expansion* λ if and only if $\lambda_2 \leq \lambda$.

Intuitively, an expander graph is sparse and high-connected; *i.e.*, every subset of vertices has relatively many neighbors, and to disconnect a large part of the graph, one has to cut many edges. Alon et al. [3] were the first to formalize the relationship of this combinatorial property and the spectral expansion of graphs.

Lemma 1 (Alon et al. [3]). *Let $G = (V, E)$ be a d-regular graph with the second largest eigenvalue λ of the normalized adjacency matrix. Let $n = |V|$. Then for any subset S and T of V, $\left|e(S,T) - \frac{d|S||T|}{n}\right| \leq \lambda d\sqrt{|S||T|}$, where $e(S,T)$ is the number of edges in G with one endpoint in S and the other endpoint in T.*

Note that $d|S||T|/n$ is the expectation of the number of edges between S and T in a random d-regular graph. A small spectral expansion λ implies a good combinatorial expansion. Alon et al. [3] also proved a lower bound of spectral expansion for any d regular expanders; a graph whose spectral expansion approximately equal to this lower bound is called a *Ramanujan graph*.

Theorem 2 (Alon-Boppana, [2]). *Any infinite family of d-regular graphs has spectral expansion (as $N \to \infty$) at least $2 \cdot \frac{\sqrt{d-1}}{d} - o(1)$.*

Definition 1 (Ramanujan Graph). *A d-regular graph is said to be a Ramanujan graph if $\lambda_2 \leq 2\frac{\sqrt{d-1}}{d}$.*

Note that $\lambda = 1$ for any bipartite graph. For such graphs, the operator *double cover* defined below can be used to generate a bipartite graph from any regular graph while maintaining the combinatorial expansion of the original graph.

For a given d-regular graph $G = (V, E)$ with $V = \{v_1, \cdots, v_n\}$, the *double cover* of G is a bipartite graph $H = (L \cup R, E^\star)$ with left vertex set $L = \{\ell_1, \cdots, \ell_n\}$ and right vertex set $R = \{r_1, \cdots, r_n\}$, such that $(\ell_i, r_j) \in E^\star$ if and only if $(v_i, v_j) \in E$. It is straightforward to see that if H is a double cover of a d-regular expander G, then H is also d-regular.

3 Expander-Based Algorithms

At a high level, our construction makes use of an initial linear code, a family of explicitly constructible Ramanujan graphs, and several combinatorial methods. First of all, we design a code satisfying the basic Hamming constraint $C_1(k_1)$. We begin our construction by analyzing certain Ramanujan graphs.

For each prime number $n, n > 3$, we construct the graph $G_n = (V, E)$ as follows: $V = \mathbb{Z}_n$, and for every vertex $x \in V$, x is connected to $x + 1$, $x - 1$ and x^{-1}, where 0^{-1} is defined as 0 and all the operations described above are modulo n.

The following theorem, originally conjectured by Selberg, was proven and further improved by Rudnick and Sarnak.

Theorem 3 ([20]). *Let G be the graph constructed above with the second largest eigenvalue λ of the normalized adjacency matrix. Then $\lambda \leq \frac{3}{4}$.*

Corollary 1. *There exists a family of explicitly constructible 3-regular Ramanujan graphs.*

Our construction also makes use of several results about the density distribution of primes, as stated in Lemmas 4 and 5. Let p_i be the i-th prime number and $p_1 = 2$.

Lemma 4 **([19]).** *For $n \in \mathbb{N}$, $p_n \geq n(\ln n + \ln\ln n - 1)$.*

Lemma 5 **([4]).** *If $n \geq 6$, then $p_n \leq n(\ln n + \ln\ln n)$.*

Lemma 6. *For any prime $k, k \geq 41 = p_{13}$, there exists at least one prime in the interval $[k, 1.56 \cdot k]$.*

We use a linear $[3,2,2]$-code $\mathcal{C}_0$, consisting of $\mathcal{C}_0(00)=000, \mathcal{C}_0(01)=110, \mathcal{C}_0(10)=101$, and $\mathcal{C}_0(11)=011$ to construct DNA words satisfying basic Hamming constraint $C_1(k_1)$. Each desired codeword in $\{0,1\}^{3n}$ can be considered as an assignment of the edges of graph G_n's double cover. From this way, each binary string $\boldsymbol{x} \in \{0,1\}^{3n}$ is a codeword of $\mathcal{C}$ if and only if the assignment with respect to every vertex v_i, *i.e.*, the subsequence $x_i x_{n+i} x_{2n+i}$, is a codeword in $\mathcal{C}_0$. Note that each edge label must satisfy constraints imposed by both its left and right endpoints. The description of our first algorithm for DNA words is detailed as follows.

Algorithm 1. Expander-Based Algorithm

```
1: $\mathcal{C} = \emptyset$.
2: Let $G = (V, E_1)$ be a 3-regular Ramanujan graphs with vertex size $|V| = n$.
3: Let $H = (L \cup R, E_2)$ be the double cover of $G$.
4: for every binary string $\boldsymbol{x} := x_1 \cdots x_{3n} \in \{0,1\}^{3n}$ do
5:    if $x_i x_{n+i} x_{2n+i} \in \mathcal{C}_0$, for all $1 \leq i \leq n$ then
6:       $\mathcal{C} = \mathcal{C} \cup \{\boldsymbol{x}\}$
7:    end if
8: end for
```

Theorem 7. *Let G be a 3-regular Ramanujan graph with n vertices and $\mathcal{C}_0$ be a linear $[3,2,2]$-code. Then there exists a family of linear $\left[3n, n, \frac{3n}{4}\right]$-code $\mathcal{C}$ for every prime number n.*

Proof. Given a graph G, let $H = (L \cup R, E)$ be the double cover of G. Without loss of generality, let $\mathcal{C} \subseteq \{0,1\}^{3n}$.

From the description of Algorithm 1, it is obvious that for each vertex $v_i \in L$, the corresponding codeword consists of the i-th, $(n+i)$-th and $(2n+i)$-th's positions of $\boldsymbol{x}$. Thus, a sequence $\boldsymbol{x} \in \mathcal{C}$ if and only if x_i, x_{n+i} and x_{2n+i} are the same, *i.e.*, $x_i x_{n+i} x_{2n+i}$ corresponds to a valid code in $\mathcal{C}_0$ for vertex $v_i \in L \cup R, 1 \leq i \leq n$. Therefore the number of constraints in $\mathcal{C}$ is $4n = 6n(1 - \frac{1}{3})$, and $\mathcal{C}$'s rate is $\frac{1}{3}$. For a codeword $\boldsymbol{x}$, define $X = \{e | x_e = 1\}$ and let S and T be the sets of left and right vertices to which edges in X are incident. Therefore the degree of X with respect to the vertices in S and T is 3, and $|X| \geq \frac{3}{2}\big(|S| + |T|\big)$.

On the other hand, by Lemma 1 we obtain $|X| \leq e(S,T) \leq \frac{3}{n}|S||T| + 3\lambda\sqrt{|S||T|}$. Combining the inequalities above, we obtain $\frac{3}{2}\big(|S|+|T|\big) \leq \frac{3}{4n}\big(|S|+|T|\big)^2 + \frac{3\lambda}{2}\big(|S|+|T|\big)$. Since $\lambda < \frac{3}{4}$, we have $|S|+|T| \geq \frac{n}{2}$.

Combining the Equations above, we have $|X| \geq \frac{3n}{4}$ and $\mathcal{C}$'s distance is at least $\frac{3n}{4}$. □

From Theorem 7, we can construct a family of linear $\left[3n, n, \frac{3n}{4}\right]$-code $\mathcal{C}$ where n is a prime number. For the general case, we use Theorem 6 to choose a prime number which is the nearest prime to the given codeword length.

Corollary 2. *For the given codeword length* $\log n$*, there exists a deterministic algorithm to output a* $\left[3\log n, \log n, \frac{3}{4}\log n\right]$*-code* $\mathcal{C}$ *in* $O(n\log n)$ *time and* $O(\log n)$ *space.*

Then construct the desired codes for the general $n, k \in \mathbb{N}$. From the construction of $\mathcal{C}$, observe that we only need to consider the case $k > \frac{3}{4}\log n$. In this case, let $k = f(n) \cdot \log n$, and the resulting codes can be obtained by concatenating sequences of length $3\log n$ $\frac{4}{3}f(n)$ times. It is not difficult to show that the constraint C_1 can be satisfied, and the total length is $4k$. This observation is formalized in Theorem 8 below.

Theorem 8. *Given n and k, Algorithm 1 outputs n sequences of DNA words with length* $\max\{3\log n, 4k\}$ *satisfying constraints C_1 in time $O(n\log n)$ and space $O(\log n)$.*

Now we turn to the problem of constructing a code satisfying constraints $C_1(k_1)$ and $C_4(k_4)$. For any sequence $\boldsymbol{x} = x_1x_2\cdots x_\ell$, define $\boldsymbol{x}^1 = x_2x_3\cdots x_\ell x_1, \cdots,$ $\boldsymbol{x}^i = x_{i+1}\cdots x_\ell x_1 \cdots x_i$. From $\mathcal{C}_1$'s construction, if $\boldsymbol{x} \in \mathcal{C}_1$, then $\boldsymbol{x}^i \in \mathcal{C}_1$ for $i \in \{1, \cdots, \ell\}$. Thus $H(\boldsymbol{x}^1[1\cdots\ell-1], \boldsymbol{x}[2\cdots\ell]) = 0$, which implies that $\mathcal{C}_1$ does not satisfy constraint C_4. So we seek for algorithms to construct the desired codes $\mathcal{C}_{1,4}$.

Theorem 9. *There exists an algorithm to construct a code $\mathcal{C}_{1,4}$ satisfying $C_1(k_1)$ and $C_4(k_4)$ with codeword length* $\ell = \max\{(3+\epsilon)\log n, (4+\epsilon)k\}$*, where* $k = \max\{k_1, k_4\}$ *and ϵ tends to 0 when n is big enough.*

Proof. For any code $\mathcal{C}_1$ and $\boldsymbol{x} \in \mathcal{C}_1$, delete $x^1, \cdots, x^{\ell-1}$ from $\mathcal{C}_1$, and the resulting codewords in C_1 constitute our desired $\mathcal{C}_{1,4}$.

For any two sequences $\boldsymbol{x}$ and $\boldsymbol{y}$ satisfying $\boldsymbol{x} \neq \boldsymbol{y}^i$, $H\left(\boldsymbol{x}[1\cdots i], \boldsymbol{y}[(\ell-i+1)\cdots\ell]\right)$ $\geq k - (\ell - i)$ for all i, so the resulting code satisfies constraint C_1 and C_4.

Therefore, we can construct a code of length $\ell = \max\{3\log n, 4k_1, 4k_4\}$ satisfying constraints C_1 and C_4, such that the number of valid codes is $\frac{n}{\log n}$. Thus, $\mathcal{C}_{1,4}$ satisfies C_1 and C_4 and the codeword length is $\ell = \max\{(3+o(1))\log n, (4+o(1))k\}$, where $k = \max\{k_1, k_4\}$. $\square$

4 Derandomization-Based Algorithm

In this section, we derandomize a random algorithm similar as in [17] to construct n DNA words of length $(c_1\log n + c_2k)$ which satisfy $C_1(k_1)$ and $C_4(k_4)$. The basic idea is generating the DNA words with the occurrence probabilities of 0 and 1 are equal with the goal that every pair of DNA words satisfy $C_1(k_1)$ and $C_4(k_4)$. By setting $\ell = c_1\log n + c_2k$, we can prove such a set of DNA words exists (Theorems 11 and 12). By applying a derandomization process, we can construct the DNA words by Algorithm 2 in $O\left(\max\{(k\ell^\star)^2, n^2k\ell^\star\}\right)$ time, where $\ell^\star$ is the length of codewords satisfying $\mathcal{C}_1$ and $\mathcal{C}_4$.

Definition 2. *An $n \times \ell$ binary matrix M is a k-distance matrix if for any two distinct rows r_α and r_β of M, the Hamming distance between $r_\alpha[1 \cdots i]$ and $r_\beta[\ell - i + 1 \cdots \ell]$ is at least $k - (\ell - i)$ for all positive integer i.*

Definition 3. *For any two rows r_α and r_β of binary matrix M, $\mathrm{HD}(M, \alpha, \beta, i)$ is defined as the Hamming distance between $r_\alpha[1 \cdots i]$ and $r_\beta[\ell - i + 1 \cdots \ell]$.*

Lemma 10. *An $n \times \ell$ k-distance matrix M can be converted into a code $\mathcal{C}_{1,4}$ satisfying $C_1(k_1)$ and $C_4(k_4)$ with length ℓ.*

Proof. Each of the n length-ℓ rows of M is an n length-ℓ codeword. Since the Hamming distance between rows $r_\alpha[1 \cdots i]$ and $r_\beta[\ell - i + 1 \cdots \ell]$ is at least $k-(\ell-i)$ for any two rows r_α and r_β, the codewords satisfy $C_4(k_4)$. By considering $i = \ell$, the codewords also satisfy $C_1(k_1)$.

Theorem 11. *There is an $n \times \ell$ k-distance matrix for any ℓ satisfying $\ell - k \log e - k \log \frac{\ell}{k} - 2 \log n + 2 \log k > 0$ and $\ell \geq 2k$.*

Proof. Assume we generate an $n \times \ell$ random binary matrix M, *i.e.*, the probability that the occurrence of 0 and 1 in each entry is 1/2 each. Under the condition $\ell \geq 2k$, the probability that M is a k-distance matrix is at least $1 - \binom{n}{2} \sum_{d\,=\,0}^{k-1} \binom{\ell}{d} 2^{-\ell} - 2\binom{n}{2} \sum_{i\,=\,\ell-k+1}^{\ell-1} \sum_{d\,=\,0}^{k-(\ell-i)-1} \binom{i}{d} 2^{-i} \geq 1 - n^2k^2 \left(\frac{e\ell}{k}\right)^k 2^{-\ell}$. Thus such a matrix exists if $1 - n^2k^2\left(\frac{e\ell}{k}\right)^k 2^{-\ell} > 0$, which is equivalent to

$$\ell - k \log e - k \log \frac{\ell}{k} - 2 \log n - 2 \log k > 0 \tag{1}$$

Therefore there exists an $n \times \ell$ k-distance matrix when ℓ satisfies Inequality (1).

Theorem 12. *By setting $c_1 = 2 + \delta$ and $c_2 = \frac{c_1}{2}\left[\log\left(\frac{c_1}{\ln 2 \cdot (c_1 - 2)}\right) + 2.5 - \frac{1}{\log 2}\right]$, $\ell^\star = c_1 \log n + c_2 k$ satisfies Inequality (1) for any real number δ.*

Based on Theorems 11 and 12, there exists an $n \times \ell^\star, \ell^\star = c_1 \log n + c_2 k$, k-distance matrix where $c_1 = 2 + \delta$ and $c_2 = \frac{c_1}{2}\left[\log\left(\frac{c_1}{\ln 2 \cdot (c_1 - 2)}\right) + 2.5 - \frac{1}{\ln 2}\right]$ for any positive real δ. For example, when $\delta = 1$, $\ell^\star = 3 \log n + 4.76k$; when $\delta = 0.1$, $\ell^\star = 2.1 \log n + 6.28k$.

Definition 4. *Let $M = \left(m_{i,j}\right)_{n \times \ell^\star}$ be a 0-1 random matrix[1]. $\mathbb{E}(M|A)$ is defined as the expected number of pairs of rows r_α and r_β in M with $\mathrm{HD}(M, \alpha, \beta, i) \geq k - (\ell^\star - i)$ and $\mathrm{HD}(M, \beta, \alpha, i) \geq k - (\ell^\star - i)$ for all positive integer i under the condition that some entries of M are assigned according to set of assignment A.*

[1] In Matrix theory, a random matrix is a matrix of given type and size whose entries consist of random numbers from some specific distribution.

Algorithm 2. Derandomization-Based Algorithm

```
1: Let $M = (m_{i,j})_{n \times \ell^\star}$ be a matrix.
2: $A = \phi$
3: for $c = 1$ to $\ell^\star$ do
4:    for $r = 1$ to $n$ do
5:       if $\mathbb{E}(M|A \cup \{(m_{r,c} = 0)\}) > \mathbb{E}(M|A \cup \{(m_{r,c} = 1)\})$ then
6:          $A := A \cup \{(m_{r,c} = 0\})$
7:       else
8:          $A := A \cup \{(m_{r,c} = 1)\}$
9:       end if
10:   end for
11: end for
12: return the assignment $A$
```

Lemma 13. *For any state in Algorithm 2,* $\mathrm{E}(M|A) > \binom{n}{2} - 1$*.*

Proof. We prove this lemma by induction with the size of A. In the initiation step, as

$$\mathbb{E}(M|\phi) = \binom{n}{2} \cdot \left[1 - \sum_{i=0}^{k-1} \binom{\ell^\star}{i} 2^{-\ell^\star} - 2 \sum_{i=\ell^\star-k+1}^{\ell^\star-1} \sum_{d=0}^{k-(\ell^\star-i)-1} \binom{i}{d} 2^{-i} \right],$$

we have $\mathrm{E}(M|\phi) > \binom{n}{2} - 1$. Consider the assignment of the entry $m_{r,c}$. Since

$$\begin{aligned}\mathbb{E}(M|A) &= \frac{1}{2} \cdot \mathbb{E}(M|A \cup (m_{r,c} = 0)) + \frac{1}{2} \cdot \mathbb{E}(M|A \cup (m_{r,c} = 1)) \\ &< \max\{\mathbb{E}(M|A \cup (m_{r,c} = 0)), \mathbb{E}(M|A \cup (m_{r,c} = 1))\}\end{aligned}$$

we have $\mathbb{E}(M|A) \geq \mathbb{E}(M|\phi) > \binom{n}{2} - 1$ for any state in Algorithm 2. □

Corollary 3. *Algorithm 2 outputs an* $n \times \ell^\star$ *k-distance matrix in time* $O\left(k\ell^\star(k\ell^\star + n^2)\right)$*.*

5 Generalizations

In this section, we generalize Algorithms 1 and 2 to design algorithms that can construct short DNA words for various subsets of the constraints $C_1, \cdots, C_9$. Proofs for the Thoerem can be found in the Appendix.

Theorem 14. *We can construct a length-*$(\ell^\star + k)$ *code* $\mathcal{C}_{1\sim6}$ *with the DNA alphabet satisfying* $C_1(k_1)$, $C_2(k_2)$, $C_3(k_3)$, $C_4(k_4)$, $C_5(k_5)$ *and* $C_6(k_6)$ *from a length-*$\ell^\star$ *code* $\mathcal{C}_{1,4}$ *with the binary alphabet, where* $k = \max\{k_1, k_2, \cdots, k_6\}$*.*

Theorem 15. *We can construct a code* $\mathcal{C}_{1\sim7}$ *satisfying* $C_1(k_1)$, $C_2(k_2)$, $C_3(k_3)$, $C_4(k_4)$, $C_5(k_5)$, $C_6(k_6)$ *and* $C_7(k_7)$ *with codeword length* $\ell = \ell^\star + 2k$ *from a length-*$\ell^\star$ *code* $\mathcal{C}_{1,4}$ *with the binary alphabet, where* $k = \max\{k_1, k_2, \cdots, k_7\}$*.*

Theorem 16. *We can construct sequences of DNA words to satisfy constraints $C_1(k_1)$, $C_2(k_2)$, $C_3(k_3)$, $C_7(k_7)$ and $C_8(k_8)$ with codeword length $\ell = \frac{d}{d-1}(\ell^\star + 2k)$ from a length-$\ell^\star$ code $\mathcal{C}_{1,4}$ with the binary alphabet, where $k = \max\{k_1, k_2, k_3, k_7, k_8\}$.*

Theorem 17. *We can construct a length-$(\ell^\star + 2k)$ code $\mathcal{C}_{1\sim 8}$ with the DNA alphabet satisfying $C_1(k_1)$, $C_2(k_2)$, $C_3(k_3)$, $C_4(k_4)$, $C_5(k_5)$, $C_6(k_6)$, $C_7(k_7)$ and $C_8(k_8)$ from a length-$\ell^\star$ code $\mathcal{C}_{1,4}$ with the binary alphabet when the GC content γ is in the range $\frac{1}{d+1} \le \gamma \le \frac{d}{d+1}$ where d is the parameter for the Consecutive Base Constraint C_8 and $k = \max_{1\le i\le 8}\{k_i\}$.*

Theorem 18. *We can construct a length-$\left(\frac{d}{d-1}\ell^\star + \frac{d}{2}k + 2\right)$ code $\mathcal{C}_{1\sim 8}$ with DNA alphabet satisfying $C_1(k_1)$, $C_2(k_2)$, $C_3(k_3)$, $C_4(k_4)$, $C_5(k_5)$, $C_6(k_6)$, $C_7(k_7)$ and $C_8(k_8)$ from a length-$\ell^\star$ code $\mathcal{C}_{1,4}$ with the binary alphabet.*

Theorem 19. *When $d \ge \sqrt{\frac{2\ell^\star}{k}} + 1$, we can construct a code $\mathcal{C}_{1\sim 8}$ of length $\left(\ell^\star + \frac{k}{2} + \sqrt{2k\ell^\star} + 2\right)$ with the DNA alphabet satisfying $C_1(k_1)$, $C_2(k_2)$, $C_3(k_3)$, $C_4(k_4)$, $C_5(k_5)$, $C_6(k_6)$, $C_7(k_7)$ and $C_8(k_8)$ from a code $\mathcal{C}_{1,4}$ of length $\ell^\star$ with the binary alphabet, where $k = \max_{1\le i\le 8}\{k_i\}$.*

Proof. By applying Theorem 18 using $d = \sqrt{2\ell^\star/k} + 1$. □

Next we show how to construct codewords so that the free energy constraint is satisfied. Following the approach of Breslauer et al. [10], the free energy of a DNA word $X = x_1x_2\ldots x_\ell$ can be approximated by the formula $\mathrm{FE}(X) = \text{correction factor} + \sum_{i=1}^{\ell-1}\Gamma_{x_i,x_{i+1}}$, where $\Gamma_{x,y}$ is an integer denoting the pairwise free energy between base x and base y. As in the work of Kao et al. [17], for simplicity, we denote the free energy as simply the sum $\sum_{i=1}^{\ell-1}\Gamma_{x_i,x_{i+1}}$ with respect to a given pairwise energy function Γ. Let $\Gamma_{\max}$ and $\Gamma_{\min}$ be the maximum and the minimum entries in Γ respectively, and $D = \Gamma_{\max} - \Gamma_{\min}$.

Below we present the result to construct DNA words satisfying the free energy constraint $C_9(\sigma)$ for a constant $\sigma = 4D + \Gamma_{\max}$, while simultaneously satisfying constraints C_1, C_2, C_3, C_4, C_5 and C_6.

Theorem 20. *We can construct n DNA words to satisfy constraints $C_1(k_1)$, $C_2(k_2)$, $C_3(k_3)$, $C_4(k_4)$ $C_5(k_5)$, $C_6(k_6)$ and $C_9(k_9)$ with codeword length $\ell = 3\ell^\star + 2k$ from a length-$\ell^\star$ code $\mathcal{C}_{1,4}$ with the binary alphabet, where $k = \max_{1\le i\le 6}\{k_i\}$.*

Our main results described above are described in the following two theorems.

Theorem 21 (Time Complexity of the Expander-Based Algorithm). *The codewords $\mathcal{C}_{1,4}$, $\mathcal{C}_{1\sim 6}$, $\mathcal{C}_{1\sim 7}$, $\mathcal{C}_{1\sim 3,7,8}$ and $\mathcal{C}_{1\sim 8}$ can be deterministically constructed in time $O(n\ell^\star)$.*

Theorem 22 (Time Complexity of the Derandomization-Based Algorithm). *The codewords $\mathcal{C}_{1,4}$, $\mathcal{C}_{1\sim 6}$, $\mathcal{C}_{1\sim 7}$, $\mathcal{C}_{1\sim 3,7,8}$ and $\mathcal{C}_{1\sim 8}$ can be deterministically constructed in time $O\left(k\ell^\star(k\ell^\star + n^2)\right)$.*

Acknowledgement. We wish to thank Francis Y. L. Chin, Robert Schweller, and the anonymous referees for helpful insights and comments.

References

1. Adleman, L.M.: Molecular Computation of Solutions to Combinatorial Problems. Science 266, 1021–1024 (1994)
2. Alon, N.: Eigenvalues and Expanders. Combinatorica 6, 83–96 (1986)
3. Alon, N., Chung, F.R.K.: Explicit Construction of Linear Sized Tolerant Networks. Discrete Mathematics 72, 15–19 (1989)
4. Bach, E., Shallit, J.: Algorithmic Number Theory, vol. 1. MIT Press, Cambridge (1996)
5. Ben-Dor, A., Karp, R., Schwikowski, B., Yakhini, Z.: Universal DNA Tag Systems: A Combinatorial Design Scheme. In: Proceedings of the 4th Annual International Conference on Computational Molecular Biology, pp. 65–75 (2000)
6. Braich, R.S., Johnson, C., Rothemund, P.W.K., Hwang, D., Chelyapov, N.V., Adleman, L.M.: Solution of a Satisfiability Problem on a Gel-Based DNA Computer. In: Condon, A., Rozenberg, G. (eds.) DNA 2000. LNCS, vol. 2054, pp. 27–42. Springer, Heidelberg (2001)
7. Brenneman, A., Condon, A.E.: Strand Design for Bio-Molecular Computation. Theoretical Computer Science 287, 39–58 (2001)
8. Brenner, S.: Methods for Sorting Polynucleotides using Oligonucleotide Tags. US Patent Number 5,604,097 (February 1997)
9. Brenner, S., Lerner, R.A.: Encoded Combinatorial Chemistry. Proceedings of Natianal Academy of Science 89, 5381–5383 (1992)
10. Breslauer, K.J., Frank, R., Blocker, H., Marky, L.A.: Predicting DNA Duplex Stability from the Base Sequence. Proceedings of the National Academy of Sciences 83, 3746–3750 (1986)
11. Deaton, R., Garzon, M., Murphy, R., Franceschetti, D., Stevens, S.: Genetic Search of Reliable Encodings for DNA Based Computation. In: Proceedings of the 1st Annual Conference on Genetic Programming, pp. 9–15 (1996)
12. Frutos, A.G., Liu, Q., Thiel, A.J., Sanner, A.M.W., Condon, A.E., Smith, L.M., Corn, R.M.: Demonstration of a Word Design Strategy for DNA Computing on Surfaces. Nucleic Acids Research 25, 4748–4757 (1997)
13. Gaborit, P., King, O.D.: Linear Constructions for DNA Codes. Theoretical Computer Science 334, 99–113 (2005)
14. Garzon, M., Deaton, R., Neathery, P., Franceschetti, D., Murphy, R.: A New Metric for DNA Computing. In: Proceedings of the 2nd Genetic Programming Conference, pp. 472–278 (1997)
15. Horry, S., Linial, N., Wigderson, A.: Expander Graphs and Their Applications. Bulletin of the American Mathematical Society 43(4), 439–561 (2006)
16. King, O.D.: Bounds for DNA Codes with Constant GC-content. The Electronic Journal of Combinatorics 10, #R33 (2003)
17. Kao, M.Y., Sanghi, M., Chweller, R.: Randomized Fast Design of Short DNA Words. ACM Transactions on Algorithms 5(4), Article 43 (2009)
18. Marathe, A., Condon, A., Corn, R.M.: On Combinatorial DNA Word Design. Journal of Computational Biology 8, 201–219 (2001)
19. Pierre, D.: The kth Prime is Greater Than $k(\ln k + \ln\ln k - 1)$ for $k \geq 2$. Mathematics of Computation 68(225), 411–415 (1999)

20. Selberg, A.: On the Estimation of Fourier Coefficients of Modular Forms. In: Proceedings of Pure Mathematics, vol. III, pp. 1–15. American Mathematical Society, Providence (1965)
21. Shoemaker, D.D., Lashkari, D.A., Morris, D., Mittmann, M., Davis, R.W.: Quantitative Phenotypic Analysis of Yeast Deletion Mutants Using a Highly Parallel Molecular Bar-coding Strategy. Nature 16, 450–456 (1996)
22. Sipser, M., Spielman, D.A.: Expander Codes. IEEE Transactions on Information Theory 42(6), 1710–1722 (1996)
23. Tsaftaris, S.A., Katsaggelos, A.K., Pappas, T.N., Papoutsakis, E.T.: DNA Computing from a Signal Processing Viewpoint. IEEE Signal Processing Magazine 21(5), 100–106 (2004)
24. Tulpan, D.C., Hoos, H.H.: Hybrid Randomised Neighbourhoods Improve Stochastic Local Search for DNA Code Design. In: Xiang, Y., Chaib-draa, B. (eds.) Canadian AI 2003. LNCS (LNAI), vol. 2671, pp. 418–433. Springer, Heidelberg (2003)
25. Tulpan, D.C., Hoos, H.H., Condon, A.: Stochastic Local Search Algorithms for DNA Word Design. In: Hagiya, M., Ohuchi, A. (eds.) DNA 2002. LNCS, vol. 2568, pp. 229–241. Springer, Heidelberg (2003)
26. Wang, H.: Proving theorems by pattern recognition. Bell System Technical Journal 40, 1–42 (1961)
27. Winfree, E.: Algorithmic Self-Assembly of DNA. California Institute of Technology (1998)
28. Winfree, E., Liu, F., Wenzler, L., Seeman, N.: Design and Self-Assembly of Two-Dimensional DNA Crystals. Nature 394, 539–544 (1998)

Hamiltonian Cycles in Subcubic Graphs: What Makes the Problem Difficult*

Nicholas Korpelainen[1,2], Vadim V. Lozin[1,2], and Alexander Tiskin[1,3]

[1] DIMAP, University of Warwick, Coventry CV4 7AL, UK
[2] Mathematics Institute, University of Warwick, Coventry CV4 7AL, UK
[3] Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK

Abstract. We study the computational complexity of the HAMILTONIAN CYCLE problem in the class of graphs of vertex degree at most 3. Our goal is to distinguish boundary properties of graphs that make the problem difficult (NP-complete) in this domain. In the present paper, we discover the first boundary class of graphs for the HAMILTONIAN CYCLE problem in subcubic graphs.

1 Introduction

In a graph, a Hamiltonian cycle is a cycle containing each vertex of the graph exactly once. Determining whether a graph has a Hamiltonian cycle is an NP-complete problem. Moreover, it remains NP-complete even if restricted to subcubic graphs, i.e. graphs of vertex degree at most 3. However, under some further restrictions, the problem may become polynomial-time solvable. A trivial example of this type is the class of graphs of vertex degree at most 2. Our goal is to distinguish boundary graph properties that make the problem difficult in subcubic graphs. In our study, we restrict ourselves to the properties that are hereditary, in the sense that whenever a graph possesses a certain property, the property is inherited by all induced subgraphs of the graph.

Within the family of hereditary properties, we are interested in those that are critical for the HAMILTONIAN CYCLE problem. Speaking informally, these are the minimal properties that make the problem difficult. This notion was recently introduced under the name "boundary classes" with respect to the maximum independent set problem [1], and then was applied to some other graph problems [2,3,7]. The importance of this notion is due to the fact that boundary classes separate difficult instances of a problem from polynomially solvable ones.

In [3], it was observed that there must exist at least five boundary classes of graphs for the HAMILTONIAN CYCLE problem, but none of them has been identified so far. In the present paper, we discover the first boundary class for the problem in question.

The organization of the paper is as follows. In Section 2, we introduce basic notation and concepts, including the definition of boundary graph property.

* Research supported by the Centre for Discrete Mathematics and Its Applications (DIMAP), University of Warwick.

© Springer-Verlag Berlin Heidelberg 2010

In Section 3, we prove an NP-completeness result, which suggests an idea of the structure of a boundary property for the HAMILTONIAN CYCLE problem in subcubic graphs. In Section 4, we provide a structural characterization of the property, and in Section 5 we prove its minimality.

2 Preliminaries

All graphs in this paper are finite, without loops or multiple edges. For a graph G, we denote by $V(G)$ and $E(G)$ the vertex set and the edge set of G, respectively. The neighborhood of a vertex $v \in V(G)$ (i.e. the set of vertices adjacent to v) is denoted $N(v)$. The degree of v is the number of its neighbors. If the degree of each vertex of G is at most 3, we call G a *subcubic* graph. The vertices of degree 3 are called cubic. For a subset of vertices $U \subseteq V(G)$, we denote by $G[U]$ the subgraph of G induced by U, i.e. the subgraph of G with vertex set U, and two vertices being adjacent in $G[U]$ if and only if they are adjacent in G. We say that a graph H is an induced subgraph of G if H is isomorphic to $G[U]$ for some $U \subseteq V(G)$.

A *graph property* (or *class of graphs*) is a set of graphs closed under isomorphism. A property is *hereditary* if with any graph G it contains all induced subgraphs of G. It is known that a graph property is hereditary if and only if it can be characterized in terms of forbidden induced subgraphs. More formally, given a set of graphs M, we say that a graph G is M-free if G does not contain induced subgraphs from the set M. Then a class X of graphs is hereditary if and only if there is a set M such that X is the class of all M-free graphs. If M is a finite set, we call X a *finitely defined class* of graphs. For instance, the class of subcubic graphs is obviously hereditary, and the set of minimal forbidden induced subgraphs for it consists of 11 graphs on five vertices (in each graph, one vertex is dominating and the remaining vertices induce all possible 4-vertex graphs).

A graph property X will be called *HC-tough* if there is no polynomial-time algorithm to solve the problem for graphs in X. If $P \neq NP$, the family of HC-tough properties is non-empty, in which case the problem of characterization of the family of graph classes with polynomial-time solvable HAMILTONIAN CYCLE problem arises. By analogy with the induced subgraph characterization of hereditary classes, we want to characterize this family in terms of minimal properties that do not belong to it. Finding the set of minimal forbidden induced subgraphs for a hereditary class is generally a difficult problem, as the example of perfect graphs shows [5]. Finding the set of minimal classes for a certain family is not only more difficult, but generally is impossible, since the family of graph classes partially ordered by the inclusion relationship is not well-founded. In other words, the difficulty is that an HC-tough class may contain infinitely many HC-tough subclasses.

To overcome this difficulty, we employ the notion of boundary classes (see e.g. [3]), which can be defined as follows. A class of graphs X will be called a *limit class* for the HAMILTONIAN CYCLE problem, if $X = \bigcap_{i=1}^{\infty} X_i$, where $X_1 \supseteq X_2 \supseteq \dots$

is a sequence of HC-tough classes. A minimal limit class will be called *boundary* for the problem in question. The importance of this notion is due to the following theorem [3].

Theorem 1. *The* HAMILTONIAN CYCLE *problem is polynomial-time solvable in a finitely defined class of graphs X if and only if X contains none of the boundary classes for this problem.*

In what follows, we identify the first boundary class of graphs for the HAMILTONIAN CYCLE problem.

3 Approaching a Limit Class

As we mentioned in the introduction, the HAMILTONIAN CYCLE problem is NP-complete for subcubic graphs [6]. Recently, it was shown [3,4] that the problem is NP-complete for graphs of large girth, i.e. graphs without small cycles. In this section, we strengthen both these results. First, we show that the problem is NP-complete in the class of subcubic graphs, in which every cubic vertex has a non-cubic neighbor. Throughout the paper, we denote this class by Γ.

Lemma 1. *The* HAMILTONIAN CYCLE *problem is NP-complete in the class Γ.*

Proof. Plesńik [8] proved that the HAMILTONIAN CYCLE problem is NP-complete in the class of directed graphs, where every vertex has either indegree 1 and outdegree 2, or indegree 2 and outdegree 1. The lemma is proved by a reduction from the HAMILTONIAN CYCLE problem on such graphs, which we call *Plesńik graphs*. Given a Plesńik graph H, we associate with it an undirected graph from Γ as follows. First, we consider all the *prescribed edges* of H, i.e. directed edges $u \to v$, such that either u has outdegree 1, or v has indegree 1 (or both). We replace every such edge by a *prescribed path* $u \to w \to v$, where w is a new node of indegree and outdegree 1. Then, we erase orientation from all edges, and denote the resulting undirected graph by G.

Clearly, $G \in \Gamma$. Assume H has a directed Hamiltonian cycle. Then the corresponding edges of G form a Hamiltonian cycle in G. Conversely, if G has a Hamiltonian cycle, then it must contain all the prescribed paths, and therefore the corresponding Hamiltonian cycle in H respects the orientation of the edges. □

Now we strengthen Lemma 1 as follows. Denote by $Y_{i,j,k}$ the graph represented in Figure 1 and call any graph of this form a tribranch. Also, denote $\mathcal{Y}_p = \{Y_{i,j,k} : i, j, k \leq p\}$ and $\mathcal{C}_p = \{C_k : k \leq p\}$. Finally, let $\mathcal{S}_p$ be the class of $\mathcal{C}_p \cup \mathcal{Y}_p$-free graphs in Γ.

Lemma 2. *For any $p \geq 1$, the* HAMILTONIAN CYCLE *problem is NP-complete in the class $\mathcal{S}_p$.*

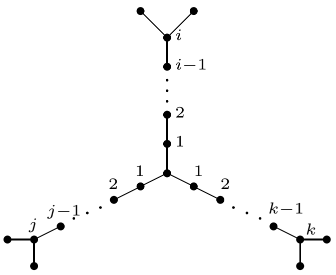


Fig. 1. A tribranch $Y_{i,j,k}$

Proof. We reduce the problem from the class Γ to $\mathcal{C}_p \cup \mathcal{Y}_p$-free graphs in Γ. Let G be a graph in Γ. Obviously, every edge of G incident to a vertex of degree 2 must belong to every Hamiltonian cycle in G (should G have any). Therefore, by subdividing each of such edges with p new vertices we obtain a graph $G' \in \Gamma$ which has a Hamiltonian cycle if and only if G has. It is not difficult to see that G' is $\mathcal{Y}_p$-free. Moreover, G' has no small cycles (i.e. cycles from $\mathcal{C}_p$) containing at least one vertex of degree 2. If G' has a cycle $C \in \mathcal{C}_p$ each vertex of which has degree 3, we apply to an arbitrary vertex a_0 of C the transformation F_p represented in Figure 2, where a_3 denotes a non-cubic neighbor of a_0. Clearly, F_p transforms G' into a new graph in Γ, which has a Hamiltonian cycle if and only if G has. Moreover, this transformation increases the length of C without producing any new cycle from $\mathcal{C}_p$ or any tribranch from $\mathcal{Y}_p$. Repeated applications of this transformation allow us to get rid of all small cycles. Thus, any graph G in Γ can be transformed in polynomial time into a $\mathcal{C}_p \cup \mathcal{Y}_p$-free graph in Γ, which has a Hamiltonian cycle if and only if G has. Together with the NP-completeness of the problem in the class Γ, this proves the lemma. □

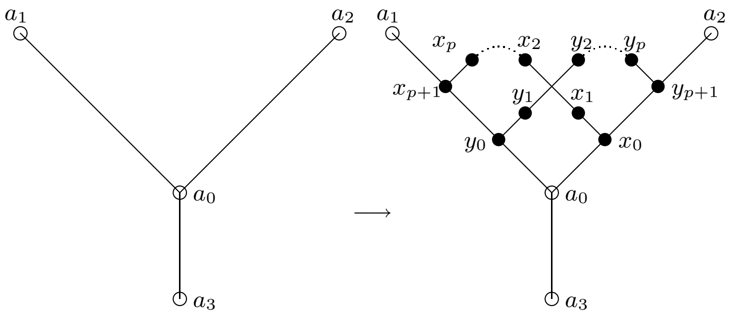


Fig. 2. Transformation F_p

4 Limit Class

The results of the previous section show that $\bigcap_{p \geq 1} \mathcal{S}_p$ is a limit class for the HAMILTONIAN CYCLE PROBLEM. Throughout the paper we will denote this class by $\mathcal{S}$. In the present section, we describe the structure of graphs in the class $\mathcal{S}$. Let us define a *caterpillar with hairs of arbitrary length* to be a subcubic tree in which all cubic vertices belong to a single path. An example of a caterpillar with hairs of arbitrary length is given in Figure 3.

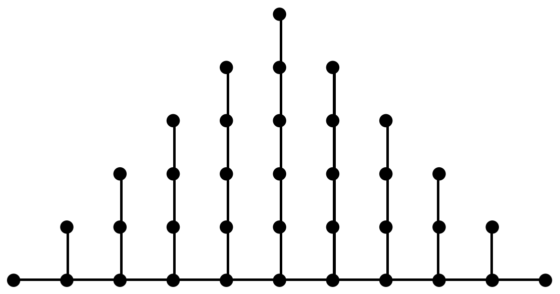


Fig. 3. A caterpillar with hairs of arbitrary length

Lemma 3. *A graph G belongs to the class $\mathcal{S}$ if and only if every connected component of G is a caterpillar with hairs of arbitrary length.*

Proof. If every connected component of G is a caterpillar with hairs of arbitrary length, then G is a subcubic graph without induced cycles or tribranches. Therefore, G belongs to $\mathcal{S}$.

Conversely, let G be a connected component of a graph in $\mathcal{S}$. Then, by definition, G is a subcubic tree without tribranches. If G has at most one cubic vertex, then obviously G is a caterpillar with hairs of arbitrary length. If G has at least two cubic vertices, then let P be an induced path of maximum length connecting two cubic vertices, say v and w. Suppose there is a cubic vertex u that does not belong to P. The path connecting u to P meets P at a vertex different from v and w (since otherwise P would not be maximum). But then a tribranch arises. This contradiction shows that every cubic vertex of G belongs to P, i.e., G is a caterpillar with hairs of arbitrary length. □

In the next section, we will prove that $\mathcal{S}$ is a minimal limit class for the Hamiltonian cycle problem. In the proof we will use the following obvious lemma, where T_d ($d \geq 2$) denotes a caterpillar with a path of length $2d$ (containing all cubic vertices) and $2d-1$ consecutive hairs of lengths $1, 2, \ldots, d-1, d, d-1, \ldots, 2, 1$ (see Figure 3 for the graph T_5).

Lemma 4. *Every graph in $\mathcal{S}$ is an induced subgraph of T_d for some $d \geq 2$.*

5 Minimality of the Limit Class

All results of this section are proved under the assumption that $P \neq NP$. The proof of minimality of the class $\mathcal{S}$ is based on the following lemma.

Lemma 5. *If for every graph G in $\mathcal{S}$, there is a constant $p = p(G)$, such that the* HAMILTONIAN CYCLE *problem can be solved in polynomial time for G-free graphs in $\mathcal{S}_p$, then $\mathcal{S}$ is a minimal limit class for the problem.*

Proof. Assume, by contradiction, that for every graph G in $\mathcal{S}$, there is a constant $p = p(G)$, such that the HAMILTONIAN CYCLE problem can be solved in polynomial time for G-free graphs in $\mathcal{S}_p$, but $\mathcal{S}$ is not a minimal limit class. Let $\mathcal{X}$ be a limit class which is a proper subclass of $\mathcal{S}$. Then there must exist a graph G in $\mathcal{S}$ that does not belong to $\mathcal{X}$. Denote by $p = p(G)$ the constant associated with G, and by $\mathcal{Z}$ the class of G-free graphs in $\mathcal{S}_p$. By our assumption, the problem is solvable in polynomial time in $\mathcal{Z}$.

Clearly $\mathcal{X} \subseteq \mathcal{Z}$. Let us show that $\mathcal{Z}$ also is a limit class for the HAMILTONIAN CYCLE problem. Since $\mathcal{X}$ is a limit class, we have $\mathcal{X} = \bigcap\limits_n \mathcal{X}_n$ for a sequence $\mathcal{X}_1 \supseteq \mathcal{X}_2 \supseteq \ldots$ of HC-tough graph classes. But then the class $\mathcal{Z}_n := \mathcal{X}_n \cup \mathcal{Z}$ is HC-tough for each n, $\mathcal{Z}_k \supseteq \mathcal{Z}_{k+1}$ for each k, and $\mathcal{Z} = \bigcap\limits_n \mathcal{Z}_n$, i.e. $\mathcal{Z}$ is a limit class for the HAMILTONIAN CYCLE problem.

We observe that the class $\mathcal{Z}$ is defined by finitely many forbidden induced subgraphs. Indeed, the set of forbidden subgraphs for this class consists of G, finitely many cycles and tribranches, and the set of forbidden graphs for the class Γ. To characterize the class Γ in terms of forbidden induced subgraphs, we need to exclude 11 graphs containing a dominating vertex of degree 4 (which is equivalent to bounding vertex degree by 3), and finitely many subcubic graphs containing a cubic vertex with three cubic neighbors.

Since the set of forbidden induced subgraphs for the class $\mathcal{Z}$ is finite, there must exist an n such that $\mathcal{Z}_n$ contains none of the forbidden graphs for $\mathcal{Z}$. But then $\mathcal{Z}_n = \mathcal{Z}$, which contradicts the assumption that the problem is polynomial-time solvable in the class $\mathcal{Z}$, while $\mathcal{Z}_n$ is an HC-tough class of graphs. This contradiction proves the lemma. □

Now we apply Lemma 5 to prove the key result of this section.

Lemma 6. *For each graph $T \in \mathcal{S}$, there is a constant p such that the* HAMILTONIAN CYCLE *problem can be solved in polynomial time for T-free graphs in $\mathcal{S}_p$.*

Proof. By Lemma 4, T is an induced subgraph of T_d for some d. We define $p = 3 \times 2^d$, and will prove the lemma for T_d-free graphs in $\mathcal{S}_p$. Obviously, this class contains all T-free graphs in $\mathcal{S}_p$.

Let G be a T_d-free graph in $\mathcal{S}_p$. Without loss of generality, we will assume that G has no vertices of degree 1, since otherwise there is no Hamiltonian cycle in G. Let us call an edge of G *black*, if it belongs to every Hamiltonian cycle in G (should such a cycle exist). Similarly, we will call an edge of H *white*, if it belongs to no Hamiltonian cycle in G. We will show that every vertex of G is incident to at least 2 black edges. This is obviously true for vertices of degree 2. Therefore, let v be a cubic vertex of G.

Denote by H the subgraph of G induced by the set of vertices of distance at most d from v. Since the degree of each vertex of H is at most 3, the number of vertices in H is less than p. Since H belongs to $\mathcal{S}_p$, it cannot contain small cycles and small tribranches (i.e. graphs from the set $\mathcal{C}_p \cup \mathcal{Y}_p$). Moreover, H cannot contain large cycles and large tribranches, because the size of H is too small (less than p). Therefore, H belongs to $\mathcal{S}$, and obviously H is connected. Thus, H is a caterpillar with hairs of arbitrary length. Observe that each leaf u of H is at distance exactly d from v, since otherwise u has degree 1 in G.

Let P be a path in H connecting two leaves and containing all vertices of degree 3. If every vertex of P (except the endpoints) has degree 3, then $H = T_d$, which is impossible because G is T_d-free. Therefore, P must contain a vertex of degree 2. Let v_i be such a vertex closest to v, and let $(v = v_0, v_1, \ldots, v_i)$ be the path connecting v_i to $v = v_0$ (along P). Then the edge $v_i v_{i-1}$ is black, as it is incident to a vertex of degree 2. By the choice of v_i, the vertex v_{i-1} has degree 3, and hence it has a neighbor u that does not belong to P. Therefore, the edge uv_{i-1} is also black, which implies that the edge $v_{i-1}v_{i-2}$ is white. In its turn, this implies that $v_{i-2}v_{i-3}$ is black, and therefore, as before, $v_{i-3}v_{i-4}$ is white. By induction, we conclude that the colors of the edges of the path $(v = v_0, v_1, \ldots, v_i)$ alternate. If the edge v_0v_1 is white, then the other two edges incident to $v = v_0$ are black. If the edge v_0v_1 is black, then the edge connecting v to the vertex outside P is also black.

Thus, we proved that every vertex of G is incident to at least 2 black edges. Now checking whether G has a Hamiltonian cycle is equivalent to checking whether the set of black edges forms a Hamiltonian cycle, which is obviously a polynomially solvable task. □

From Lemmas 5 and 6 we conclude that

Theorem 2. *$\mathcal{S}$ is a boundary class for the* HAMILTONIAN CYCLE *problem.*

References

1. Alekseev, V.E.: On easy and hard hereditary classes of graphs with respect to the independent set problem. Discrete Applied Mathematics 132, 17–26 (2003)
2. Alekseev, V.E., Korobitsyn, D.V., Lozin, V.V.: Boundary classes of graphs for the dominating set problem. Discrete Mathematics 285, 1–6 (2004)
3. Alekseev, V.E., Boliac, R., Korobitsyn, D.V., Lozin, V.V.: NP-hard graph problems and boundary classes of graphs. Theoretical Computer Science 389, 219–236 (2007)
4. Arkin, E.M., Mitchell, J.S.B., Polishchuk, V.: Two New Classes of hamiltonian Graphs (Extended Abstract). Electronic Notes in Discrete Mathematics 29, 565–569 (2007)

5. Chudnovsky, M., Robertson, N., Seymour, P.D., Thomas, R.: The strong perfect graph theorem. Ann. Math. 164, 51–229 (2006)
6. Itai, A., Papadimitriou, C.H., Szwarcfiter, J.L.: Hamilton paths in grid graphs. SIAM J. Computing 11, 676–686 (1982)
7. Lozin, V.V.: Boundary classes of planar graphs. Combinatorics. Probability and Computing 17, 287–295 (2008)
8. Plesńik, J.: The NP-completeness of the Hamiltonial cycle problem in planar digraphs with degree bound two. Information Processing Letters 8, 199–201 (1979)

A Dichotomy for k-Regular Graphs with $\{0, 1\}$-Vertex Assignments and Real Edge Functions

Jin-Yi Cai[1,⋆] and Michael Kowalczyk[2]

[1] Computer Sciences Department, University of Wisconsin at Madison
Madison, WI 53706, USA
jyc@cs.wisc.edu

[2] Department of Mathematics and Computer Science, Northern Michigan University
Marquette, MI 49855, USA
mkowalcz@nmu.edu

Abstract. We prove a complexity dichotomy theorem for a class of Holant Problems over k-regular graphs, for any fixed k. These problems can be viewed as graph homomorphisms from an arbitrary k-regular input graph G to the weighted two vertex graph on $\{0, 1\}$ defined by a symmetric function h. We completely classify the computational complexity of this problem. We show that there are exactly the following alternatives, for any given h. Depending on h, over k-regular graphs: Either (1) the problem is #P-hard even for planar graphs; or (2) the problem is #P-hard for general (non-planar) graphs, but solvable in polynomial time for planar graphs; or (3) the problem is solvable in polynomial time for general graphs. The dependence on h is an explicit criterion. Furthermore, we show that in case (2) the problem is solvable in polynomial time over k-regular planar graphs, by exactly the theory of holographic algorithms using matchgates.

1 Introduction

In this paper we continue the study of a class of Holant Problems [4,5,10]. We aim to extend our previous dichotomy results [10] on 3-regular graphs to all k-regular graphs, for an arbitrary k.

The problems we address in this paper can be described as follows. An input k-regular graph $G = (V, E)$ is given, where every $e \in E$ is labeled with a (symmetric) edge function h. In this paper we will mainly consider real-valued functions. The function h takes 0-1 inputs from its incident nodes and outputs arbitrary real values. Now we consider all possible $\{0, 1\}$-assignments on the vertex set V. The computational problem is to compute the quantity

$$\text{Holant}(G) = \sum_{\sigma:V\to\{0,1\}} \prod_{\{u,v\}\in E} h(\{\sigma(u), \sigma(v)\}). \tag{1}$$

⋆ Supported in part by NSF grant CCF-0830488.

© Springer-Verlag Berlin Heidelberg 2010

In Section 2 we will give a more detailed account of the Holant framework, of which (1) is a special case. Our special case of Holant problems with vertex assignments can be also viewed as graph homomorphisms from an arbitrary k-regular input graph G to the weighted two vertex graph on $\{0, 1\}$ defined by the symmetric function h. In general, Holant Problems are a natural class of counting problems which can encode all counting Constraint Satisfaction Problems (#CSP) and graph homomorphisms [1,2,6,7,8,9]. For instance, if h is the Boolean Or function on two input bits, then problem (1) is Vertex Cover: Holant(G) is the number of vertex covers in a k-regular graph.

Dichotomy theorems (i.e., the problem is either in P or #P-hard, depending on h) have been given in many cases [1,2,6,7,8]. However, a major tool in the hardness proofs of these papers is to construct graph gadgets whose vertices have varying degrees, especially some vertices have arbitrarily high degrees. Therefore if we inquire about the computational complexity of these counting problems on k-regular graphs, these dichotomy theorems are not applicable. More intrinsically, over k-regular graphs, there are in fact new tractable cases computable in P, which would be #P-hard over non-regular graphs.

In the same spirit, holographic algorithms using matchgates [3,15] have produced surprisingly novel P-time algorithms if we restrict to planar graphs. In this paper we will add to the body of evidence that a wide class of counting problems, which are #P-hard on general graphs, become tractable in P on planar graphs, *and they do so precisely due to* holographic algorithms using matchgates.

The inability to use graph gadgets with arbitrarily high degrees makes hardness proofs more difficult. We denote the (symmetric) edge function h by $[x, y, z]$, where $x = h(00), y = h(01) = h(10)$ and $z = h(11)$. Functions will also be called gates or signatures. To tackle the hardness of these counting problems on *regular graphs*, we proved in [4] a dichotomy theorem which implies the case when G is a 3-regular graph and the function h is a 0-1 valued Boolean function. The main proof technique is to extend the method of *interpolation* introduced by Valiant [13,14] which had been further developed by many others [1,7,8,12]. One new ingredient in [4] is an algebraic criterion which ensures that *interpolation* succeeds. However that criterion is Galois theoretic, and to go beyond integer valued functions $h = [x, y, z]$, this approach will be difficult. In [5] the dichotomy theorem is extended to all real-valued functions h over 3-regular graphs, and we have to deal with infinitely many problems, where each tuple $(x, y, z) \in \mathbb{R}^3$ defines a problem. So instead of the Galois theoretic approach we used a computational approach: We devised a large number of recursive gadgets and showed, with the help of symbolic algebra, especially the decidability of semi-algebraic sets, that this collection of gadgets *in aggregate* covered all the hardness cases. The drawback of this proof is that the computational task is enormous (and many additional ideas were needed to carry this out). In particular, it would seem hopeless to extend that approach further by simply adding computational muscle.

The immediate predecessor to the present paper is [10]. In that paper we managed to find a way to drastically reduce the dependence of symbolic computation and extend the theorem of [5] to all complex valued functions h. This was accomplished by a new method of higher dimensional iteration for gadget construction, and by finding a new polynomial expression for Holant(G) for 3-regular graphs G. More precisely, after a normalization we may assume h takes the form $[a, 1, b]$, for $a, b \in \mathbb{C}$. Then for any 3-regular graph G, there exists a polynomial $P(\cdot, \cdot)$ with integer coefficients, such that Holant$(G) = P(X, Y)$, where $X = ab$ and $Y = a^3 + b^3$.

In this paper we extend the dichotomy theorem to all k-regular graphs. We will use the technique of higher dimensional iteration [5], adapting it to the case of k-regular graphs. We will also find a corresponding polynomial expression for Holant(G). The situation with an arbitrary degree k requirement creates at least two additional difficulties. The first is that with infinitely many k, it seems likely that we will need an infinite number of collections of gadgets, one for each k. The statement involving a variable k cannot be stated for a semi-algebraic set. If we follow this strategy, we can hope to prove at best a small number of concretely given constants k; the symbolic computation from the decidability of semi-algebraic sets will soon overwhelm this attempt, as k increases.

The second difficulty is presented by the parity of k. It turns out that for even degree k, the proof cannot be directly extended from degree 3. The technical reason is that it is not possible to construct an $\mathcal{F}$-gate with an odd number of dangling edges in a regular graph of even degree (see Section 2 for definitions). In particular this means that the approach in [10] to construct all unary signatures will not work. It is not possible to construct starter and finisher gadgets as described in [10]. We overcome the first difficulty by fortuitously choosing a universal set of gadget families for all k, and showing that collectively they always work. Here a symbolic substitution $X = ab$ and $Y = a^k + b^k$ is shown to essentially eliminate all symbolic dependence on k. We overcome the second difficulty by changing the strategy of constructing all unary signatures to constructing all binary signatures of a certain kind. This set of binary signatures plays the virtual role of all unary signatures for our purposes. Our main theorem is as follows:

Theorem 1. *Suppose $a, b \in \mathbb{R}$, and let $X = ab$ and $Y = a^k + b^k$ where $k \geq 3$ is an integer. Then the Holant Problem in (1) on k-regular graphs with $h = [a, 1, b]$ is #P-hard except in the following cases, for which the problem is in P.*

1. $X = 1$
2. $X = 0$ *and* $Y = 0$
3. $X = -1$ *and* $Y = 0$
4. $X = -1$, *k is even, and* $Y = \pm 2$

If we restrict the input to planar k-regular graphs, then these four categories are solvable in P, as well as a fifth category $Y^2 = 4X^k$ (equivalently, $a^k = b^k$), and the problem remains #P-hard in all other cases.

We actually prove the above theorem for all $a, b \in \mathbb{C}$ such that $X, Y \in \mathbb{R}$. This is slightly stronger than Theorem 1. Furthermore, our methods are not sensitive to the exact model of computation using complex numbers.

2 Background and Notation

We state the counting framework more formally in terms of bipartite graphs. As we will see, any k-regular graph with vertex assignments is interchangeable with a certain bipartite $(2,k)$-regular graph with edge assignments, but it is often more convenient to work in terms of bipartite graphs. A *signature grid* $\Omega = (G, \mathcal{F}, \pi)$ consists of a labeled graph $G = (V, E)$ where π labels each vertex $v \in V$ with a function $f_v \in \mathcal{F}$. We consider all edge assignments $\xi : E \to \{0,1\}$; f_v takes inputs from its incident edges $E(v)$ at v and outputs values in $\mathbb{C}$. The counting problem on the instance Ω is to compute $\text{Holant}_\Omega = \sum_{\xi:E\to\{0,1\}} \prod_{v\in V} f_v(\xi \mid_{E(v)})$.

Suppose G is a bipartite graph (U, V, E) such that each $u \in U$ has degree 2. Furthermore suppose each $v \in V$ is labeled by an EQUALITY gate $=_k$ where $k = \deg(v)$. Then any non-zero term in Holant_Ω corresponds to a 0-1 assignment $\sigma : V \to \{0,1\}$. In fact, we can merge the two incident edges at $u \in U$ into one edge e_u, and label this edge e_u by the function f_u. This gives an edge-labeled graph (V, E') where $E' = \{e_u : u \in U\}$. For an edge-labeled graph (V, E') where $e \in E'$ has label g_e, $\text{Holant}_\Omega = \sum_{\sigma:V\to\{0,1\}} \prod_{e=(v,w)\in E'} g_e(\sigma(v), \sigma(w))$. If each g_e is the same function g (but assignments $\sigma : V \to [q]$ take values in a finite set $[q]$) this is exactly the H-coloring problem (for undirected graphs g is a symmetric function). In particular, if (U, V, E) is a $(2,k)$-regular bipartite graph, equivalently $G' = (V, E')$ is a k-regular graph, then this is the H-coloring problem restricted to k-regular graphs. In this paper we will discuss k-regular graphs where each g_e is the same symmetric real-valued or complex-valued function. We also remark that for general bipartite graphs (U, V, E), giving EQUALITY (of various arities) to all vertices on one side V defines #CSP as a special case of Holant Problems. But whether EQUALITY of various arities are present has a major impact on complexity, thus Holant Problems are a refinement of #CSP.

A symmetric function $g : \{0,1\}^k \to \mathbb{C}$ can be denoted as $[g_0, g_1, \ldots, g_k]$, where g_i is the value of g on inputs of Hamming weight i. They are also called *signatures*. Frequently we will revert back to the bipartite view: for $(2,k)$-regular bipartite graphs (U, V, E), if every $u \in U$ is labeled $g = [g_0, g_1, g_2]$ and every $v \in V$ is labeled $r = [r_0, r_1, \ldots, r_k]$, then we also use $\#[g_0, g_1, g_2] \mid [r_0, r_1, \ldots, r_k]$ to denote the Holant Problem. The main dichotomy theorem in this paper is about $\#[x, y, z] \mid =_k$, for all $x, y, z \in \mathbb{R}$. If $y = 0$ then this is easily computable in P, so we assume $y \neq 0$. The problem $\#[x, y, z] \mid =_k$ has the same complexity as $\#[x/y, 1, z/y] \mid =_k$, hence we can normalize $[x, y, z]$ so that $y = 1$. We will also denote $\text{Hol}_k(a, b) = \#[a, 1, b] \mid =_k$, or $\text{Pl-Hol}_k(a, b)$ to denote $\#[a, 1, b] \mid =_k$ when restricted to planar graphs as input. More generally, If $\mathcal{G}$ and $\mathcal{R}$ are sets of signatures, and vertices of U (resp. V) are labeled by signatures from $\mathcal{G}$ (resp. $\mathcal{R}$), then we also use $\#\mathcal{G} \mid \mathcal{R}$ to denote the bipartite Holant Problem. Signatures in $\mathcal{G}$ are called *generators* and signatures in $\mathcal{R}$ are called *recognizers*. This notation is particularly convenient when we perform holographic transformations. Throughout this paper, all $(2,k)$-regular bipartite graphs are arranged with generators on the degree 2 side and recognizers on the degree k side.

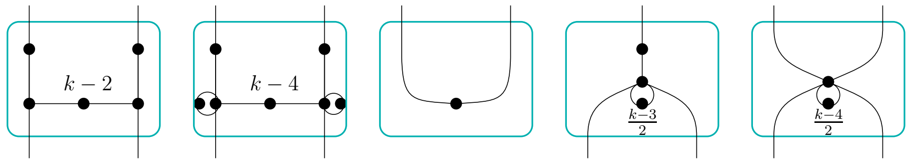
Signatures from $\mathcal{F}$ are available at each vertex as part of an input graph. Instead of a single vertex, we can use graph fragments to generalize this notion.

An $\mathcal{F}$-gate Γ is a pair $(H, \mathcal{F})$, where $H = (V, E, D)$ is a graph with some dangling edges D (Figure 1 contains some examples). Other than these dangling edges, an $\mathcal{F}$-gate is the same as a signature grid. The role of dangling edges is similar to that of external nodes in Valiant's notion [15], however we allow more than one dangling edge for a node. In $H = (V, E, D)$ each node is assigned a function in $\mathcal{F}$ (we do not consider "dangling" leaf nodes at the end of a dangling edge among these), E are the regular edges, and D are the dangling edges. Then we can define a function for this $\mathcal{F}$-gate: $\Gamma(y_1, y_2, \ldots, y_q) = \sum_{(x_1,x_2,\ldots,x_p)\in\{0,1\}^p} H(x_1, x_2, \ldots, x_p, y_1, y_2, \ldots, y_q)$, where $p = |E|$, $q = |D|$, $(y_1, y_2, \ldots, y_q) \in \{0,1\}^q$ denotes an assignment on the dangling edges, and $H(x_1, x_2, \ldots, x_p, y_1, y_2, \ldots, y_q)$ denotes the value of the signature grid on an assignment of all edges, i.e., the product of evaluations at every vertex of H, for $(x_1, x_2, \ldots, x_p, y_1, y_2, \ldots, y_q) \in \{0,1\}^{p+q}$.

We will also call this function the signature of the $\mathcal{F}$-gate Γ. An $\mathcal{F}$-gate can be used in a signature grid as if it is just a single node with the same signature. We note that even for a very simple signature set $\mathcal{F}$, the signatures for all $\mathcal{F}$-gates can be quite complicated and expressive. Matchgate signatures are an example, where $\mathcal{F}$ consists of just the Exact-One function [15].

The dangling edges of an $\mathcal{F}$-gate are considered as input or output variables. Any m-input n-output $\mathcal{F}$-gate can be viewed as a 2^n by 2^m matrix M which transforms arity-m signatures into arity-n signatures (this is true even if m or n are 0). The $\mathcal{F}$-gates in this paper will transform symmetric signatures to symmetric signatures. This implies that there exists an equivalent $n+1$ by $m+1$ matrix $\widetilde{M}$ which operates directly on column vectors written in symmetric signature notation. We will henceforth identify the matrix $\widetilde{M}$ with the $\mathcal{F}$-gate itself. The constructions in this paper are based upon three different types of bipartite $\mathcal{F}$-gates which we call *starter gadgets*, *recursive gadgets*, and *finisher gadgets*. An *arity-r starter gadget* is an $\mathcal{F}$-gate with no input but r output edges. If an $\mathcal{F}$-gate has r input and r output edges then it is called an *arity-r recursive gadget*. Finally, an $\mathcal{F}$-gate is an *arity-r finisher gadget* if it has r input edges 1 output edge. With the exception of gadget F_2 (which has two inputs and two outputs), we consider any dangling edge incident with a generator as an output edge and any dangling edge incident with a recognizer as an input edge; see Figure 1.

Throughout this paper, we denote $X = ab$ and $Y = a^k + b^k$, and we assume that $X, Y \in \mathbb{R}$ and $a, b \in \mathbb{C}$. In all cases our gadgets have signature $[a, 1, b]$



(a) Gadget M_1 (b) Gadget M_2 (c) Gadget S_1 (d) Gadget F_1 (e) Gadget F_2

Fig. 1. Labels indicate the number of pairs of edges in parallel

assigned to the degree 2 vertices and signature $=_k$ assigned to the degree k vertices. We use S_i, M_i, and F_i to denote the (recurrence matrices of the) gadgets displayed in Figures 1 and 2 with the vertex signatures assigned as described.

3 Interpolation Technique

In this section we introduce the interpolation technique we will use in this paper. We start with Lemma 3.2 from [10].

Lemma 1. *Suppose that the following gadgets can be built using complex-valued signatures from a finite generator set $\mathcal{G}$ and a finite recognizer set $\mathcal{R}$.*

1. *A binary starter gadget with nonzero signature $s = [z_0, z_1, z_2]$.*
2. *A binary recursive gadget with nonsingular recurrence matrix M, for which $[z_0, z_1, z_2]^{\mathrm{T}}$ is not a column eigenvector of M^ℓ for any positive integer ℓ.*
3. *Three binary finisher gadgets with rank 2 matrices $F_1, F_2, F_3 \in \mathbb{C}^{2\times 3}$, where the intersection of the row spaces of F_1, F_2, and F_3 is the zero vector.*

Then for any $x, y \in \mathbb{C}$, $\#\mathcal{G} \cup \{[x,y]\} \mid \mathcal{R} \leq_T \#\mathcal{G} \mid \mathcal{R}$.

It will be more convenient to reframe condition 2 in terms of the eigenvalues of M. Assume that we are using a nonsingular recursive gadget M and a starter gadget whose signature s is not orthogonal to any row eigenvector of M. Additionally assume that there exist eigenvalues α and β of M for which $\frac{\alpha}{\beta}$ is not a root of unity. Then we want to show that s is not a column eigenvector of M^ℓ for any positive integer ℓ (note that s is nonzero). Writing out the Jordan Normal Form for M, we have $M^\ell s = T^{-1}D^\ell Ts$, where D^ℓ has the form $\begin{bmatrix} \alpha^\ell & 0 & 0 \\ 0 & \beta^\ell & 0 \\ 0 & * & * \end{bmatrix}$. Let $t = Ts$ and write $t = \begin{bmatrix} c \\ d \\ e \end{bmatrix}$. The first two rows of T are row eigenvectors of M. Then s is not orthogonal to the first two rows of T, hence $c, d \neq 0$. If s were an eigenvector of M^ℓ for some positive integer ℓ, then $T^{-1}D^\ell Ts = M^\ell s = \lambda s$ for some nonzero complex value λ ($\lambda \neq 0$ since M^ℓ is nonsingular), and $D^\ell t = T(\lambda s) = \lambda t$. But then $c\alpha^\ell = \lambda c$ and $d\beta^\ell = \lambda d$, which means $\frac{\alpha^\ell}{\beta^\ell} = 1$, contradicting the fact that $\frac{\alpha}{\beta}$ is not a root of unity.

We satisfy condition 3 by demonstrating a finisher gadget family suitable for all odd values of $k \geq 3$. Note that since no single-output binary finisher gadget exists when k is even, we will have to deal with even k separately (in regular graphs with even degree it is impossible to build an $\mathcal{F}$-gate with an odd number of dangling edges).

Lemma 2. *Suppose $k \geq 3$ is odd, $X \notin \{0,1\}$, and $a^k \neq b^k$. Then F_1, F_1M_1, and $F_1M_1^2$ (see Figures 1(d) and 1(a)) are all rank 2 matrices and their row spaces have trivial intersection.*

Proof. We get $F_1 = \begin{bmatrix} a^{(k-1)/2} & 0 & b^{(k-3)/2} \\ a^{(k-3)/2} & 0 & b^{(k-1)/2} \end{bmatrix}$. Build two more finisher gadgets F_1' and F_1'' using M_1 so that $F_1' = F_1M_1$ and $F_1'' = F_1M_1^2$. Since F_1 and M_1

both have full rank (note $\det(M_1) = X^{k-2}(X-1)^3$ and F_1 has a submatrix with determinant $X^{(k-3)/2}(X-1)$), it follows that F_1' and F_1'' also have full rank. To show that the row spaces of F_1, F_1' and F_1'' have trivial intersection, it suffices to show that the cross products of the row vectors of F_1, F_1', and F_1'' (which we denote by v_1, v_1', and v_1'' respectively) are linearly independent. Let N be the matrix which has as its rows, v_1', v_1', and v_1'' respectively. Then $\det(N) = 4X^{(5k-13)/2}(X-1)^7(a^k - b^k) \neq 0$.

Now consider even $k \geq 4$. The construction is similar to the odd case, with two important differences. Firstly, instead of standard finisher gadgets with a generator output vertex we use $\mathcal{F}$-gates, each with a recognizer output vertex, so for even k we will be interpolating recognizer signatures. Secondly, there are two output edges from each $\mathcal{F}$-gate. It may appear as though this causes a problem with the main construction, but for any such "modified finisher gadget" F_i, there is a unique matrix $F \in \mathbb{C}^{2\times 3}$ such that $F_i = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} F$. Thus, the interpolation technique still applies, but in the case of even k we end up with the reduction $\#\mathcal{G} \mid \mathcal{R} \cup \{[x,0,y]\} \leq_T \#\mathcal{G} \mid \mathcal{R}$ for any $x, y \in \mathbb{C}$. The proof of the following Lemma is similar to Lemma 2, and is omitted.

Lemma 3. *Suppose $k \geq 4$ is even and let F be the unique $\mathbb{C}^{2\times 3}$ matrix such that $F_2 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} F$ (see Figure 1(e)). If $X \notin \{0,1\}$, and $a^k \neq b^k$ then F, FM_1, and FM_1^2 are all rank 2 matrices and the row spaces have trivial intersection.*

We will interpolate generator signatures of the form $[x, y]$ for odd k and recognizer signatures of the form $[x, 0, y]$ for even k. In the case of odd k, connecting an $=_k$ signature vertex to a vertex with signature $[x, y]$ and $k-3$ vertices with signature $[1, 1]$ results in an $\mathcal{F}$-gate with signature $[x, 0, y]$. This means that regardless of which variant of finisher gadget we apply, we can simulate any recognizer signature of the form $[x, 0, y]$. With $[x, 0, y]$ signatures in hand, we can simulate the generator signature $[0, 1, 1]$ directly by Lemma 3.3 from [10]. (Technically, that lemma requires generator signatures of the form $[x, y]$. However, it is easy to verify that recognizer signatures of the form $[x, 0, y]$ suffice with only trivial modifications to the construction). Thus we have a reduction from Vertex Cover on k-regular graphs (i.e. $\#[0,1,1] \mid =_k$), which is known to be #P-hard for all $k \geq 3$ even if the input is restricted to planar graphs (counting Vertex Cover on 3-regular planar graphs is proved to be #P-hard in [16]; this can be generalized to a proof for any $k \geq 3$). This gives us the following result.

Theorem 2. *If the following can be built using generator $[a, 1, b]$ and recognizer $=_k$ where $X \notin \{0,1\}$, $k \geq 3$, and $a^k \neq b^k$, then $\text{Pl-Hol}_k(a, b)$ is #P-hard:*

1. *A planar binary recursive gadget with nonsingular recurrence matrix M which has eigenvalues α and β such that $\frac{\alpha}{\beta}$ is not a root of unity.*
2. *A planar binary starter gadget with signature s which is not orthogonal to any row eigenvector of M.*

4 Classification of Real Valued Signatures on k-Regular Graphs

4.1 Tractable Problems

Once we show that X and Y capture the complexity of the Holant problems we are studying, the problem of proving tractability (or #P-hardness) becomes easier. To extend this idea from [10] to k-regular graphs for all $k \geq 3$, we need to consider a parity issue that arises when k is even and the number of vertices in the graph is odd. It is impossible to have an odd number of vertices when k is odd, and when the number of vertices in the graph is even the Holant is a polynomial in X and Y. However, when the number of vertices is odd (which can only happen for even k), the Holant is no longer a polynomial in X and Y; there is an extra factor $(a^{k/2} + b^{k/2})$. Nevertheless, the following lemma can be proved using an argument similar to Lemma 4.1 in [10].

Lemma 4. *Let G be a k-regular graph with n vertices. If n is even, then there exists a polynomial $P(\cdot, \cdot)$ with two variables and integer coefficients such that for any signature grid Ω having underlying graph G and every edge labeled $[a, 1, b]$, the Holant value is $\text{Holant}_\Omega = P(ab, a^k + b^k)$. If n is odd, then there exists a polynomial $P(\cdot, \cdot)$ as before such that $\text{Holant}_\Omega = (a^{k/2} + b^{k/2})P(ab, a^k + b^k)$.*

Corollary 1. *Let G be any k-regular graph with n vertices, where k is even and n is odd, and let Ω be any signature grid having underlying graph G and every edge labeled $[a, 1, b]$. If $a^{k/2} + b^{k/2} = 0$, then $\text{Holant}_\Omega = 0$.*

If the number of vertices n is odd then k must be even, and we may assume $a^{k/2} + b^{k/2} \neq 0$. Then we can change Ω to Ω' by adding an extra vertex with $k/2$ simple loops. Then Ω' has an even number of vertices. The Holant value of Ω' is $\text{Holant}_{\Omega'} = (a^{k/2} + b^{k/2})\text{Holant}_\Omega$, hence we can compute Holant_Ω from $\text{Holant}_{\Omega'}$. Therefore we will always assume the number of vertices is even from now on. For even n, Lemma 4 says that X and Y capture the essence of (in)tractability for the Holant Problems under consideration. If we find the complexity for any one setting of a and b such that $X = ab$ and $Y = a^k + b^k$, then we have already characterized all settings of a and b that result in the same X and Y. Specifically, given a signature $[a, 1, b]$ as input, one can compute the Holant in terms of any a' and b' for which $a'b' = X$ and $(a')^k + (b')^k = Y$.

Theorem 3. *If any of the following four conditions is true, then $\text{Hol}_k(a, b)$ is solvable in* P*:*

1. $X = 1$
2. $X = 0$ *and* $Y = 0$
3. $X = -1$ *and* $Y = 0$
4. $X = -1$ *and* [$Y = \pm 2$ *if* k *is even, and* $Y = \pm 2\mathfrak{i}$ *if* k *is odd*]

If $Y^2 = 4X^k$ then $\text{Pl-Hol}_k(a, b)$ *is solvable in* P.

Proof. If $X = 1$ then the signature $[a, 1, b]$ is degenerate and the Holant can be computed in polynomial time. If $X = Y = 0$, then $a = b = 0$ and a connectivity argument can be applied to the edges to calculate the Holant. If $X = -1$, then applying a holographic transformation under basis $T = \begin{bmatrix} 1 & 0 \\ 0 & a \end{bmatrix}$, we get $T^{\otimes 2} g = [a, a, -a]^{\mathrm{T}}$ and $r(T^{-1})^{\otimes k} = [1, 0, 0, \ldots, 0, a^{-k}]$, where $r = [1, 0, 0, \ldots, 0, 1]$ is the $=_k$ signature and $g = [a, 1, -a^{-1}]^{\mathrm{T}}$ (note this g corresponds to the assumption $X = -1$). Multiplying the signature $[a, a, -a]$ by a^{-1} does not change the complexity of the problem, so $\#g \mid r$ is equivalent in complexity to $\#[1, 1, -1] \mid [1, 0, 0, \ldots, 0, a^{-k}]$, which is known to be tractable in P if $a^k \in \{1, -1, \mathfrak{i}, -\mathfrak{i}\}$, by families $\mathcal{F}_1$ and $\mathcal{F}_3$ in [2]. If k is even, then $Y = a^k + a^{-k}$, which can be set to -2, 0, or 2 by using any $a \in \mathbb{C}$ such that a^k is -1, $\mathfrak{i}$, or 1 respectively. If k is odd, then $Y = a^k - a^{-k}$, which can be set to $-2\mathfrak{i}$, 0, or $2\mathfrak{i}$ by using any $a \in \mathbb{C}$ such that a^k is $-\mathfrak{i}$, 1, or $\mathfrak{i}$, respectively. Finally, if $Y^2 = 4X^k$, then $a^k = b^k$ and holographic algorithms using matchgates can be applied when the input graph is planar (see [3], Lemmas 4.4 and 4.8).

4.2 Intractable Problems

In this section we show that the remaining problems are #P-hard. This is carried out primarily by applying binary recursive gadgets to Theorem 2 for different real-valued settings of X and Y (note this includes some cases where a or b are complex). Usually when we speak of a gadget in this section, we really mean a member of a family of gadgets; most of the gadgets in Figures 1 and 2 actually define families of gadgets, with a different gadget for each k. We will make use of the following lemma, proved in [10].

Lemma 5. *If all roots of the complex polynomial $x^3 + Bx^2 + Cx + D$ have the same norm, then $C|C|^2 = \overline{B}|B|^2 D$.*

This criterion can be used to study the suitability of binary recursive gadgets for interpolation. Every recursive gadget we use has a recurrence matrix with a characteristic polynomial of the form $x^3 + Bx^2 + Cx + D$ where B, C, and D are polynomials in X and Y with integer coefficients. Since X and Y are real, the condition of Lemma 5 is simply the zero set of the real polynomial $f(X, Y) = C^3 - B^3 D$, and this becomes an important tool in proving #P-hardness. We want to show that for any remaining X and Y, there is a planar binary recursive gadget with a nonsingular recurrence matrix such that the corresponding polynomial $f(X, Y)$ is nonzero. This implies that condition 1 of Theorem 2 is satisfied (condition 2 can be shown separately). However, there is some difficulty in applying this lemma; not only is the degree of f high in a and b for all but the smallest of gadgets, but the exponents in the polynomial f are functions of k. It is not obvious how to obtain suitable binary recursive gadgets for all k. Furthermore, for a family of gadgets indexed by k, if we treat k as a variable, the question can no longer be formulated as one about semi-algebraic sets. Nevertheless, there exists a pair of binary recursive gadget families M_1 and M_2 suitable for handling these difficulties (see Figures 1(a) and 1(b)). It is the *combination* of these two gadget families and the algebraic relationship *between* them, combined with the coordinate change $X = ab$ and $Y = a^k + b^k$, which allows us to eliminate k entirely and to finally derive a general result. When $k \leq 4$ this family of gadget pairs does not work, so we will deal with this case separately, by selecting different binary recursive gadgets for $k = 4$. We also need to find starter gadgets suitable to be used with M_1 and M_2 in the recursive construction, so we state an easy way of identifying such starter gadgets first.

Lemma 6. *Let* $M \in \mathbb{C}^{n\times n}$ *and let* $s \in \mathbb{C}^{n\times 1}$. *If* $\det([s, Ms, \ldots, M^{n-1}s]) \neq 0$ *then* s *is not orthogonal to any row eigenvector of* M.

Proof. Suppose s is orthogonal to a row eigenvector v of M with eigenvalue λ. Then $v[s, Ms, ..., M^{n-1}s] = 0$, since $vM^i s = \lambda^i vs = 0$. Since $v \neq 0$ this is a contradiction.

Lemma 7. *Suppose* $k \geq 5$, $X \notin \{0,1\}$, *and* $a^k \neq b^k$. *If* $X = -1$, *additionally assume that* k *is odd and* $Y \neq 0$. *Then* Pl-Hol$_k(a,b)$ *is* #P*-hard.*

Proof. We show that for every setting of X and Y under consideration, either gadget M_1 or M_2 satisfies Theorem 2 when used with S_1 as a starter gadget. The recurrence matrices of gadgets M_1 and M_2 are as follows.

$$M_1 = \begin{bmatrix} a^k & 2a & b^{k-2} \\ a^{k-1} & ab+1 & b^{k-1} \\ a^{k-2} & 2b & b^k \end{bmatrix}, \quad M_2 = \begin{bmatrix} a^k & 2a^2b & b^{k-2} \\ a^{k-1} & ab(ab+1) & b^{k-1} \\ a^{k-2} & 2ab^2 & b^k \end{bmatrix}$$

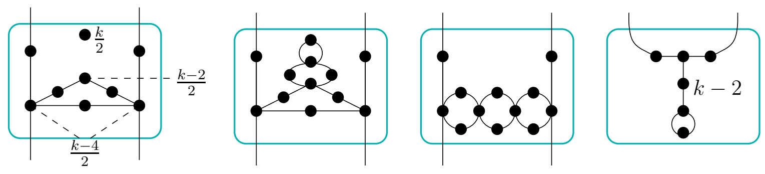
Starter gadget S_1 has signature $[a,1,b]$. Then $\det([S_1, M_1S_1, M_1^2S_1]) = (X-1)^3(b^k - a^k)(X^{k-2} - 1) \neq 0$ and $\det([S_1, M_2S_1, M_2^2S_1]) = X^2(X-1)^3(b^k - a^k)(X^{k-4} - 1) \neq 0$, so S_1 is not orthogonal to any row eigenvector of M_1 or M_2. Let the characteristic polynomials of gadgets M_1 and M_2 be $x^3 + B_1x^2 + C_1x + D_1$ and $x^3 + B_2x^2 + C_2x + D_2$ respectively, and let $Z = X^{k-3}$. Then $\det(M_1) = ZX(X-1)^3 \neq 0$ and $\det(M_2) = ZX^2(X-1)^3 \neq 0$. Now suppose $X \neq -1$. If all eigenvalues of M_i have the same norm, then by Lemma 5, $C_i^3 - B_i^3 D_i = 0$. We claim that this cannot be the case for both gadgets. Otherwise, we factorize polynomials to get

$$C_1^3 - B_1^3 D_1 = (X-1)^3(XZ-1)(ZX(1+X)^2(X^2Z + XZ + 3Y + X + 1) - Y^3)$$
$$C_2^3 - B_2^3 D_2 = X^2(X-1)^3(Z-X)(ZX(1+X)^2(XZ + Z + 3Y + X^2 + X) - Y^3)$$

hence $Y^3 = ZX(1+X)^2(X^2Z + XZ + 3Y + X + 1)$ and $Y^3 = ZX(1+X)^2(XZ + Z + 3Y + X^2 + X)$. But then, by some fortuitous factorization,

$$\begin{aligned} 0 = {} & ZX(1+X)^2(X^2Z + XZ + 3Y + X + 1) \\ & -ZX(1+X)^2(XZ + Z + 3Y + X^2 + X) = (X-1)X(1+X)^3(Z-1)Z \neq 0. \end{aligned}$$

Now suppose $X = -1$, k is odd, and $Y \neq 0$. We will show that gadget M_1 has a pair of eigenvalues with distinct norm. In this case, the characteristic polynomial of M_1 is $x^3 - Yx^2 - 2Yx - 8$, so by Lemma 5, if all roots of the characteristic polynomial have the same norm, then $-8Y^3 = C^3 = B^3D = 8Y^3$, but this implies $Y = 0$.



(a) Gadget M_3 (b) Gadget M_4 (c) Gadget M_5 (d) Gadget S_2

Fig. 2. Labels for gadget M_3 indicate the number of 2-cycles on each recognizer vertex. The label in gadget S_2 indicates the number pairs of edges in parallel.

The following Lemma deals with even $k \geq 4$, $X = -1$ and $Y \in \mathbb{R} - \{-2, 0, 2\}$. Note that when k is even, $X = -1$, and $Y \in \{-2, 0, 2\}$ the problem $\text{Hol}_k(a, b)$ is tractable.

Lemma 8. *Suppose $k \geq 4$ is even, $X = -1$ and $Y \in \mathbb{R} - \{-2, 0, 2\}$. Then $\text{Pl-Hol}_k(a, b)$ is #P-hard.*

Proof. If $X = -1$, then the characteristic polynomial of gadget M_3 is $x^3 + (4 - Y^2)x^2 + 2(4 - Y^2)(2 + (-1)^{k/2}Y)x + 8(4 - Y^2)(2 + (-1)^{k/2}Y)$, so the determinant is nonzero. By Lemma 5, if all roots of the characteristic polynomial have the same norm, then $C^3 = B^3D$ and this amounts to $(2 + (-1)^{k/2}Y)^2 = 4 - Y^2$, but then $4 \pm 4Y + Y^2 = 4 - Y^2$ and $Y(Y \pm 2) = 0$, which is not true. Finally, applying Lemma 6 to M_3 and S_1, we get $\det([S_1, M_3S_1, M_3^2S_1]) = -16Y(Y^2 - 4)^3(2 + (-1)^{k/2}Y)(2(-1)^{k/2} + Y) \neq 0$.

The bulk of the work is now done.

Lemma 9. *Suppose $k \geq 3$, $X \notin \{0, 1\}$, $a^k \neq b^k$, $(X, Y) \neq (-1, 0)$. Then $\text{Pl-Hol}_k(a, b)$ is #P-hard.*

Proof. This result is already known for $k = 3$ (see [10]). If $k \geq 5$ then this is established by Lemmas 7 and 8 (note that if k is even, $X = -1$, and $Y = \pm 2$ then $a^k = b^k$). If $k = 4$, then some symbolic computation shows that gadgets M_1, M_4, M_5, and S_1 together satisfy Theorem 2 provided $X \notin \{0, 1\}$, $a^k \neq b^k$, and $(X, Y) \neq (-1, 0)$ and hence the problem $\text{Pl-Hol}_4(a, b)$ is #P-hard for any such setting of X and Y (we omit the details).

Two cases that remain are $X = 0$ and $a^k = b^k$. Whenever $X = 0$ and $Y \notin \{0, -1\}$, either gadget M_1S_1 or gadget S_2 simulates a signature that is already covered in Lemma 9 so we are done by reduction from that case. Similarly, when $X = 0$ and $Y = -1$ a reduction from the case where $X = 0$ and $Y \notin \{0, -1\}$ can be made using gadget M_1S_2. If $a^k = b^k$ and $X \notin \{-1, 0, 1\}$, then after a suitable holographic reduction we can use gadget S_1 and either M_1 or M_2 (without a finisher gadget) to interpolate all signatures of the form $[x, 1, x]$. Then we can apply the $[1, 1, 1]$ signature to pairs of vertices with the $=_k$ signature to simulate $=_3$ and we are done by a reduction from $\#[a, 1, b] \mid =_3$ (see [10]). Note that planarity does not need to be preserved in this case and we can assume that the number of vertices in the input graph is even. This completes Theorem 1.

References

1. Bulatov, A., Grohe, M.: The complexity of partition functions. Theoretical Computer Science 348(2-3), 148–186 (2005)
2. Cai, J.-Y., Lu, P., Xia, M.: Holant problems and counting CSP. In: STOC 2009: Proceedings of the 41st Annual ACM Symposium on Theory of Computing, pp. 715–724 (2009)
3. Cai, J.-Y., Lu, P.: Holographic algorithms: from art to science. In: STOC 2007: Proceedings of the 39th Annual ACM Symposium on Theory of Computing, pp. 401–410 (2007)
4. Cai, J.-Y., Lu, P., Xia, M.: Holographic algorithms by Fibonacci gates and holographic reductions for hardness. In: FOCS 2008: Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science, pp. 644–653 (2008)
5. Cai, J.-Y., Lu, P., Xia, M.: A computational proof of complexity of some restricted counting problems. In: Chen, J., Cooper, S.B. (eds.) TAMC 2009. LNCS, vol. 5532, pp. 138–149. Springer, Heidelberg (2009)
6. Dyer, M., Goldberg, L.A., Paterson, M.: On counting homomorphisms to directed acyclic graphs. Journal of the ACM 54(6) (2007)
7. Dyer, M., Greenhill, C.: The complexity of counting graph homomorphisms (extended abstract). In: SODA 2000: Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 246–255 (2000)
8. Goldberg, L.A., Grohe, M., Jerrum, M., Thurley, M.: A complexity dichotomy for partition functions with mixed signs. CoRR, abs/0804.1932 (2008)
9. Hell, P., Nešetřil, J.: On the complexity of H-coloring. Journal of Combinatorial Theory, Series B 48(1), 92–110 (1990)
10. Kowalczyk, M., Cai, J.-Y.: Holant problems for regular graphs with complex edge functions. In: STACS 2010: Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science, pp. 525–536 (2010)
11. Tarski, A.: A decision method for elementary algebra and geometry. University of California Press, Berkeley (1951)
12. Vadhan, S.: The complexity of counting in sparse, regular, and planar graphs. SIAM Journal on Computing 31(2), 398–427 (2001)
13. Valiant, L.: The complexity of enumeration and reliability problems. SIAM Journal on Computing 8(3), 410–421 (1979)
14. Valiant, L.: The complexity of computing the permanent. Theoretical Computer Science 8, 189–201 (1979)
15. Valiant, L.: Quantum circuits that can be simulated classically in polynomial time. SIAM Journal on Computing 31(4), 1229–1254 (2002)
16. Xia, M., Zhang, P., Zhao, W.: Computational complexity of counting problems on 3-regular planar graphs. Theoretical Computer Science 384(1), 111–125 (2007)

Graph Sharing Games: Complexity and Connectivity[⋆]

Josef Cibulka[1], Jan Kynčl[2], Viola Mészáros[2,3], Rudolf Stolař[1], and Pavel Valtr[2]

[1] Department of Applied Mathematics,
Charles University, Faculty of Mathematics and Physics,
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
cibulka@kam.mff.cuni.cz, ruda@kam.mff.cuni.cz

[2] Department of Applied Mathematics and
Institute for Theoretical Computer Science,
Charles University, Faculty of Mathematics and Physics,
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
kyncl@kam.mff.cuni.cz

[3] Bolyai Institute, University of Szeged,
Aradi vértanúk tere 1, 6720 Szeged, Hungary
viola@math.u-szeged.hu

Abstract. We study the following combinatorial game played by two players, Alice and Bob. Given a connected graph G with nonnegative weights assigned to its vertices, the players alternately take one vertex of G in each turn. The first turn is Alice's. The vertices are to be taken according to one (or both) of the following two rules: (T) the subgraph of G induced by the taken vertices is connected during the whole game, (R) the subgraph of G induced by the remaining vertices is connected during the whole game. We show that under all the three combinations of rules (T) and (R), for every $\varepsilon > 0$ and for every $k \geq 1$ there is a k-connected graph G for which Bob has a strategy to obtain $(1 - \varepsilon)$ of the total weight of the vertices. This contrasts with the game played on a cycle, where Alice is known to have a strategy to obtain 4/9 of the total weight.

We show that the problem of deciding whether Alice has a winning strategy (i.e., a strategy to obtain more than half of the total weight) is PSPACE-complete if condition (R) or both conditions (T) and (R) are required. We also consider a variation of the game where the first player who violates condition (T) or (R) loses the game. We show that deciding who has the winning strategy is PSPACE-complete.

[⋆] Work on this paper was supported by the project 1M0545 of the Ministry of Education of the Czech Republic. Viola Mészáros was also partially supported by OTKA Grant K76099 and by the grant no. MSM0021620838 of the Ministry of Education of the Czech Republic. Josef Cibulka and Rudolf Stolař were also supported by the Czech Science Foundation under the contract no. 201/09/H057.

© Springer-Verlag Berlin Heidelberg 2010

1 Introduction

Dan Brown devised the following pizza puzzle in 1996 which we formulate using graph notation. The pizza with n slices of not necessarily equal size, can be considered as a cycle C_n with nonnegative weights on the vertices. Two players, Bob and Alice, are sharing it by taking turns alternately. In every turn one vertex is taken. The first turn is Alice's. Afterwards, a player can take a vertex only if the subgraph induced by the remaining vertices is connected. Dan Brown asked if Alice can always obtain at least half of the sum of the weights. Peter Winkler and others solved this puzzle by constructing graphs where Bob had a strategy to get even 5/9 of the pizza. Later Peter Winkler posed the pizza problem asking how much of the pizza can Alice gain. He conjectured that the bound 5/9 is tight. This problem has been solved in the affirmative by showing that Alice can always get 4/9 of the pizza [1,2].

Here we investigate a generalized setting where the cycle C_n is replaced with an arbitrary connected graph G. Consider the following two conditions:

(T) the subgraph of G induced by the taken vertices is connected during the whole game,
(R) the subgraph of G induced by the remaining vertices is connected during the whole game.

Observe that the two conditions are equivalent if G is a cycle. The generalized game is called *game T*, *game R*, or *game TR* if we require condition (T), condition (R), or both conditions (T) and (R), respectively. In games T and R, there are graphs where Bob can ensure (almost) the whole weight to himself; see Fig. 1 for examples where the graph G is a tree. The same results hold even if G is restricted to be k-connected (for any $k \geq 2$), in any of the three variants of the game. See Fig. 2 for examples of such 2-connected graphs.

Theorem 1. *For games T, R and TR, for every $\varepsilon > 0$ and for every $k \geq 1$ there is a k-connected graph G for which Bob has a strategy to obtain $(1-\varepsilon)$ of the total weight of the vertices.*

Micek and Walczak [3] also studied, independently of us, generalizations of the pizza game. They investigated how the parity of the number of vertices affects

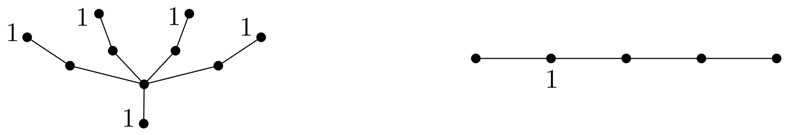


Fig. 1. Left, game T: a tree where Alice gets at most one vertex of weight 1. Right, game R: a path of even length, Bob gets the only vertex of positive weight. Vertices with no label have weight 0.

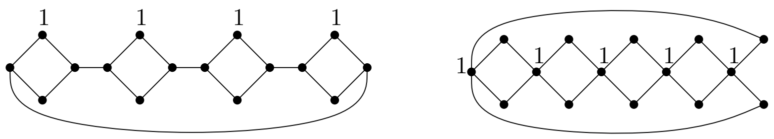


Fig. 2. Examples of 2-connected graphs where Alice gets at most one vertex of weight 1. Left: game T, or game TR. Right: game R; the number of vertices of degree 4 (and of weight 1) must be odd. Vertices with no label have weight 0.

Alice's chances to gain a positive fraction of the total weight in games T and R, particularly when the game is played on a tree. They proved that Alice can gain at least 1/4 of the total weight in game R on a tree with an even number of vertices and in game T on a tree with an odd number of vertices. For the opposite parities they constructed examples of trees (such as those in Figure 1) where Bob has a strategy to get almost all the weight.

Note that in games T and R, regardless of the strategies of the players, there is always an available vertex for the player on turn until all vertices are taken as the graph G is connected. For game TR, there is a sequence of turns to take all the vertices of G if and only if each cut vertex of G separates the graph into precisely two components and every 2-connected component of G contains at most two cut vertices. Game TR will end with all vertices taken for any sequence of turns if and only if each cut vertex of G separates the graph into two components and the set of all cut vertices in G induces a path.

As game TR does not always end with all vertices taken, it is natural to consider the following variation of the game, which we call a *canonical game TR*. Given a graph G, Alice and Bob take turns alternately. In each turn a player takes one vertex of G. The first player who is forced to violate condition (T) or (R) loses the game. We show the following complexity result.

Theorem 2. *It is PSPACE-complete to decide who has the winning strategy in the canonical game TR.*

We also consider the complexity of determining the winning strategy (i.e., gaining more than half of the total weight) for the original three types of the game. We show the following.

Theorem 3. *It is PSPACE-complete to decide who has the winning strategy in game R and TR.*

However, we are unable to determine the complexity of deciding the winner for game T. The reason might be the following difference in the "nature" of games T and R. Once a vertex becomes available in game T, it remains available until the end of the game. The same holds for game R played on a tree. In games R and TR on general graphs, however, a vertex may become available and unavailable several times during the game.

Problem 4. *What is the complexity of deciding who has the winning strategy in game T? What is the complexity of deciding who has the winning strategy in game R and in game T when the input graph G is a tree?*

In this extended abstract many proofs are omitted due to space limitations.

2 Proof of Theorem 1

In games T and TR, Bob can choose the following k-connected graph with an even number of vertices (for any given $k \geq 2$): Take a large even cycle and replace each vertex in it by a $2\lceil k/4 \rceil$-clique and each edge by a complete bipartite graph $K_{2\lceil k/4 \rceil, 2\lceil k/4 \rceil}$. Assign weight 1 to one vertex in every other $2\lceil k/4 \rceil$-clique, and weight 0 to all the other vertices of the graph. It is easy to see that Bob can make sure that Alice takes at most one vertex of weight 1.

In game R, Bob can choose the following k-connected (bipartite) graph G with an odd number of vertices. The vertex set is a disjoint union of sets X, Y and Z, where Y is an m-element set for some large $m \geq k+2$, Z is the set of all k-element subsets of Y and X is chosen to be a set of at least $|Y| + |Z| + 2$ elements so that the total number of vertices is odd. The edge set of G consists of all edges between X and Y and all edges that connect a vertex $z \in Z$ with each of the k vertices $y \in Y$ such that $y \in z$. Each vertex from Y has weight 1, all the other vertices have weight 0.

Claim 5. *Bob can force Alice to get at most $\lfloor k/2 \rfloor$ vertices of weight* 1.

As the weight of the graph may be arbitrarily large, this completes the proof of Theorem 1. Note that the graphs for games T and TR in the proof have an even number of vertices and the graphs for game R an odd number of vertices. Micek and Walczak (personal communication) asked whether also k-connected graphs with opposite parities satisfying the conditions of Theorem 1 exist. We find examples of such graphs for all three variants of the game.

For game T and game TR and for every $k \geq 1$, we can construct a $(2k+1)$-connected graph $H_{n,k}$ with an odd number of vertices starting from the graph H_n described by Micek and Walczak [3, Example 2.2], replacing each vertex of weight 0 by $2k+1$ vertices of weight 0 forming a clique, and replacing each original edge by a complete bipartite graph. The graph H_n consists of vertices $a_1, a_2, \ldots, a_n$ of weight 1, vertices $b_1, b_2, \ldots, b_n$ of weight 0, and a vertex c_S of weight 0 for every non-empty subset $S \subseteq \{1, 2, \ldots, n\}$. Each a_i is joined by an edge to b_i and each c_S is joined to all b_i such that $i \in S$. For the graph $H_{n,k}$ Bob has a strategy to take all but one vertex of weight 1, analogous to the strategy for H_n [3].

For game R, for every $k \geq 1$ we construct a k-connected weighted graph $G'_{n,k}$ with $n + \binom{n}{k}$ vertices (we may assume that $n = 2^m$ for some positive integer m so that the total number of vertices is even). The construction generalizes the graph G'_n [3, Example 5.2] consisting of a clique of n vertices of weight 1, with a leaf of weight 0 attached to each vertex of the clique. The graph $G'_{n,k}$ consists

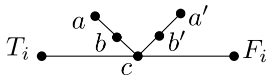


Fig. 3. Variable gadget for x_i where T_i and F_i represent the two possible values, TRUE and FALSE, of the variable x_i

of a clique on n vertices $a_1, a_2, \ldots, a_n$ of weight 1 and a vertex b_S of weight 0 for each k-element subset $S \subseteq \{1, 2, \ldots, n\}$. The vertex b_S is connected by an edge to all k vertices a_i such that $i \in S$.

Claim 6. *Bob can force Alice to get at most* $k+1$ *vertices of weight* 1.

3 Proof of Theorem 2

We proceed by polynomial reduction from the standard PSPACE-complete problem TQBF (also called QBF). An instance of the TQBF problem is a fully quantified boolean formula starting and ending with the existential quantifier:

$$\Phi = \exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \exists x_n \varphi(x_1, x_2, \ldots, x_n).$$

The question is whether Φ is true. Without loss of generality we may assume that φ in the previous expression is a 3-SAT formula.

As the game always ends after polynomially many turns, one can search through all possible game states and determine who has the winning strategy in PSPACE. To show that the problem is also PSPACE-hard, it suffices to prove that for every formula Φ there is a graph G_Φ constructible in polynomial time such that Φ is true if and only if Alice has a strategy to win on G_Φ in the canonical game TR.

3.1 The Construction of G_Φ

Let n be the number of variables in φ. Note that n is odd as φ starts and ends with the existential quantifier. First we introduce the V-gadget that will be used many times in the construction of G_Φ. The V-gadget is a path P of length four. The middle vertex c of P is distinguished because c will be identified with other vertices during the construction. For every variable of φ we build a variable gadget. For the variable x_i the gadget will consist of a path P_i of length two between the vertices T_i and F_i that represent the two possible values of x_i, and we attach a V-gadget to the middle vertex of P_i, see Figure 3. The variable gadgets are connected by edges T_iT_{i+1}, T_iF_{i+1}, F_iT_{i+1} and F_iF_{i+1} in G_Φ for all $i < n$, see Figure 4.

For every clause c_l of φ we introduce a new vertex C_l in G_Φ. The vertex C_l is connected to the three vertices in G_Φ corresponding to the literals of the clause c_l. In case the literal x_i stands with negation in c_l, the vertex C_l is connected to T_i in G_Φ, otherwise C_l is connected to F_i. We attach a V-gadget to C_l. Further

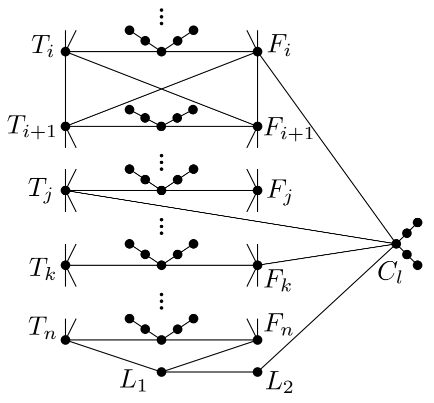


Fig. 4. Variable gadgets and the clause $c_l = (x_i, \neg x_j, x_k)$

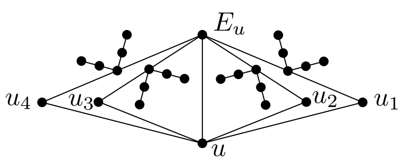


Fig. 5. Order enforcer for u where u_1, u_2, u_3, u_4 are the indicated neighbors of u and E_u is the newly added vertex

we add two special vertices L_1 and L_2 to G_Φ and edges T_nL_1, F_nL_1, L_1L_2 and L_2C_l for each C_l, see Figure 4.

The *order enforcer* for the vertex u is a gadget that prevents Alice to start at u. This we will prove later after the description of the construction. The *special neighbors* of a vertex are the neighbors of the vertex among T_i, F_i and L_i for all i. The order enforcer for the vertex u connects u to a newly added vertex E_u, adds a path S_i of length two, avoiding u, between E_u and each special neighbor of u, and adds a V-gadget to the middle vertices of each S_i, see Figure 5. We attach an order enforcer simultaneously for each T_i, F_i and L_i except of T_1 and F_1.

3.2 The Game

In the following we make some easy observations. Their proofs are omitted from this extended abstract.

Observation 7. *In a game where Alice wins, no vertex of the V-gadget can be taken.*

As a straightforward consequence of Observation 7 we get that no vertex C_l corresponding to some clause c_l can be taken from G_Φ.

Observation 8. *For every i, only one of T_i and F_i can be taken.*

Observation 9. *Let u be a vertex in G_Φ to which an order enforcer is attached. If in the first turn Alice takes u or E_u, then she loses the game.*

Observation 10. *If some neighbor of u is taken, E_u cannot be taken anymore.*

The game must proceed as follows. As a consequence of Observations 7, 8 and 9 Alice will take T_1 or F_1 in the first turn. By observations 8 and 10, the only possible choices in the ith turn for the player on turn are T_i and F_i for $i \leq n$. In the $(n+1)$st turn Bob has to take L_1 to obey the rules of the game TR. Similarly, Alice has to take L_2 in the $(n+2)$nd turn, or she has no turn and loses the game. If Alice cannot take L_2, it means that some vertex C_l corresponding to a clause c_l would get disconnected from the part of G_Φ that contains the remaining vertices T_i and F_i. This occurs if and only if previously all three vertices corresponding to the literals of c_l were taken. If after taking L_2 the subgraph induced by the remaining vertices of G_Φ is connected, then there is no further vertex to take as it would necessarily disconnect G_Φ.

For $i \leq n$, let x_i be TRUE if T_i was taken by one of the players and FALSE if F_i was taken. It follows that the formula φ is satisfied if and only if Alice can take L_2 at the end of the game, that is, if and only if she wins the game. As Alice's turns in G_Φ correspond to the variables with the existential quantifier in Φ and Bob's turns in G_Φ to the variables with the universal quantifier, it follows that Alice wins if and only if Φ is a true formula.

Obviously the construction of G_Φ was carried out in polynomial time. This completes the proof of Theorem 2.

4 Proof of Theorem 3

As in the proof of Theorem 2, we show a polynomial reduction from the TQBF problem. Without loss of generality we may assume that the considered formula Φ is of the form $\Phi = \exists x_1 \forall x_2 \exists x_3 \forall x_4 \dots \exists x_n \varphi(x_1, x_2, \dots, x_n)$, where φ is a NAE-3-SAT formula. That is, each clause of φ has three literals and it is satisfied if and only if at least one of the three literals is evaluated as TRUE and at least one as FALSE.

4.1 The Construction of G_Φ

Let m be the number of clauses in φ. We construct a 3-connected weighted graph G_Φ of size $O(m+n)$ in the following way (see Figure 6). For each variable x_i we take a path P_i with 4 vertices. The end vertices of P_i are labeled x_i and $\neg x_i$. If a variable x_i occurs in the j-th clause of φ, we add a vertex x_i^j connected by an edge to x_i and a vertex $\neg x_i^j$ connected by an edge to $\neg x_i$. The vertices x_i^j and $\neg x_i^j$ are connected by clause gadgets depicted in Figure 7. We add a set Z of $50(m+n)+1$ vertices that are connected to all vertices x_i and $\neg x_i$. Finally, we add two vertices a and b connected to all other vertices (including the edge between a and b). Observe that the constructed graph G_Φ is 3-connected and has an odd number of vertices.

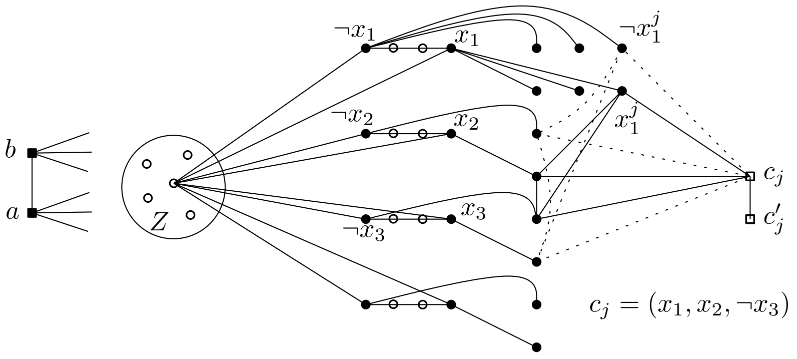


Fig. 6. An illustration of a part of the graph G_Φ

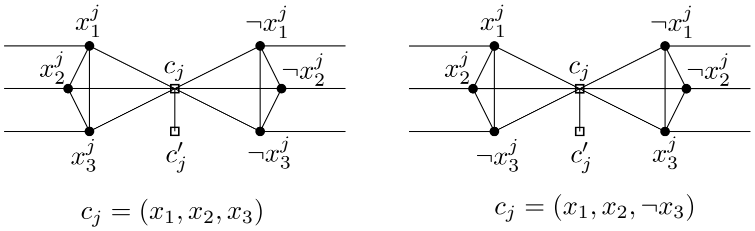


Fig. 7. Examples of NAE-3-SAT clause gadgets

The weights $w : V(G_\Phi) \to [0, \infty)$ are set as follows:

$$\begin{aligned} w(x_i) = w(\neg x_i) &= 9^{n+1-i}, \\ w(x_i^j) = w(\neg x_i^j) &= 10/999^j, \\ w(c_j) = w(c'_j) &= 11/999^j, \\ w(b) &= 9^{n+2}, \\ w(a) &= 9^{n+2} + 2 \cdot 9^{n-1} + 2 \cdot 9^{n-3} + \cdots + 2 \cdot 9^2 + 1/999^{m+1}. \end{aligned}$$

All other vertices (that is, the inner points on the paths P_i and the vertices of Z) have weight 0. The $2n + 8m + 2$ vertices of positive weight can be partitioned into the following groups: $V_0 = \{a, b\}, V_1 = \{x_1, \neg x_1\}, \ldots, V_n = \{x_n, \neg x_n\}$, and the m groups $V_{n+1}, \ldots, V_{n+m}$, each V_{n+j} consisting of 8 vertices of the j-th clause gadget. Note that the weights are chosen so that each vertex in a group V_k has larger weight than the sum of weights of all the vertices from groups $V_l, l > k$. Let W denote the total weight of all vertices in G_Φ.

We show that Alice has a winning strategy in game TR played on G_Φ if and only if Φ is true.

4.2 Starting the Game

We start with the following easy observations.

Observation 11. *Once one of the vertices a or b is taken, game TR reduces to game R as condition (T) is always satisfied further on.* □

We will call the graph induced by the remaining vertices in some position in the game briefly the *remaining graph.*

Observation 12. *If in some position in game R played on G_Φ the remaining graph has a cut vertex v, then Bob has a strategy to get v.*

Corollary 13. *If Bob is on turn in game R, he has a strategy to get any of the remaining vertices.* □

Now in a sequence of lemmas we show that the players do not have much freedom in taking the vertices, if they want to play optimally.

Lemma 14. *If both players play optimally, then Alice's first turn is on a and Bob's first turn is on b.*

4.3 Variable and Clause Gadgets

The variable gadget for the variable x_i is the path P_i connecting vertices x_i and $\neg x_i$, with only the end vertices connected to Z.

Lemma 15. *If both players play optimally, then for each $i = 1, 2, \ldots, n$, in the $(i+2)$-nd turn of the game either x_i or $\neg x_i$ is taken.*

Let $w_1, w_2, \ldots, w_n$ be the sequence of the remaining vertices from the groups $V_1, V_2, \ldots, V_n$. The vertices w_i determine a truth assignment σ of the variables: $\sigma(x_i) = \text{TRUE}$ if $w_i = x_i$ and $\sigma(x_i) = \text{FALSE}$ if $w_i = \neg x_i$. See Figure 8.

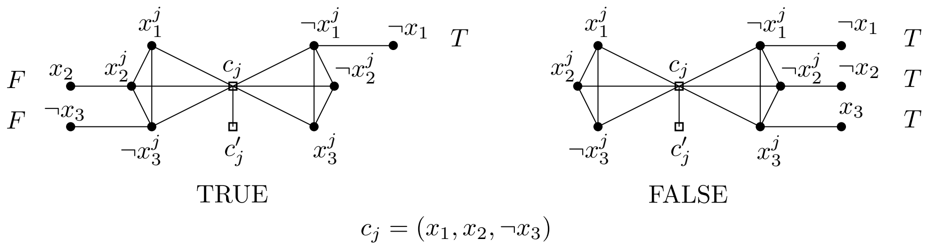


Fig. 8. A clause gadget after the players chose the evaluation σ of the variables. Left: $\sigma(x_1) = \text{TRUE}, \sigma(x_2) = \text{FALSE}, \sigma(x_3) = \text{TRUE}$, the clause c_j is evaluated as TRUE. Right: $\sigma(x_1) = \text{TRUE}, \sigma(x_2) = \text{TRUE}, \sigma(x_3) = \text{FALSE}$, the clause c_j is evaluated as FALSE.

The subgraph induced by the group V_{n+j} acts as a clause gadget for the j-th clause. Call the two vertices of the group V_{n+j} of weight $11/999^j$ *heavy* and the six vertices of weight $10/999^j$ *light*.

Lemma 16. *Suppose the players are restricted to play on the subgraph of G_Φ induced by V_{n+j} with Bob being on turn, with all vertices $Z \cup \{w_1, w_2, \ldots, w_n\}$ still remaining. If both players play optimally, then Alice takes one heavy and three light vertices if the j-th clause is satisfied by σ, otherwise she takes four light vertices. That is, Alice takes exactly half of the weight of the group V_{n+j} (namely $41/999^j$) if the j-th clause of φ is satisfied by σ and $40/999^j$ otherwise..*

Lemma 17. *If both players play optimally, then during the $8m$ (or $8m+1$) turns following the first $n+2$ turns of the game, the players take sequentially vertices of the groups $V_{n+1}, V_{n+2}, \ldots, V_{n+m}$, with the following possible exceptions: there are indices $1 \le j_1 < k_1 \le j_2 < k_2 \le \cdots \le j_l \le m$ such that Bob takes one vertex of V_{n+k_p} instead of a vertex of V_{n+j_p}, $p = 1, 2, \ldots, l-1$, and he takes a vertex outside $V_{n+1} \cup V_{n+2} \cup \cdots \cup V_{n+j_l}$ instead of a vertex of V_{n+j_l}.*

Moreover, Alice takes one heavy and three light vertices from each group V_{n+j} corresponding to a satisfied clause and four light vertices otherwise..

Bob has a strategy to get all the vertices w_i after all vertices from clause gadgets are taken: 1) take an available vertex w_i; 2) if 1) cannot be applied, take any vertex except the vertices that lie on a path P_i and neighbor an available vertex w_i. The weights of the two vertices a, b are set in such a way that if all clauses of φ are satisfied by σ, then by Lemma 17 Alice gets slightly more than half of the total weight W, and less than half if at least one clause is not satisfied.

The existence of a winning strategy for Alice on the graph G_Φ is thus naturally reformulated as the quantified boolean formula Φ, where the existential quantifiers correspond to Alice's turns and the universal quantifiers to Bob's turns on the variable gadgets.

References

1. Cibulka, J., Kynčl, J., Mészáros, V., Stolař, R., Valtr, P.: Solution of Peter Winkler's Pizza Problem. In: Fete of Combinatorics. Springer, New York (to appear); Extended abstract in: Fiala, J., Kratochvíl, J., Miller, M. (eds.): IWOCA 2009. LNCS, vol. 5874, pp. 356–368. Springer, Heidelberg (2009)
2. Knauer, K., Micek, P., Ueckerdt, T.: How to eat 4/9 of a pizza (submitted)
3. Micek, P., Walczak, B.: Parity in graph sharing games (manuscript)

A Visual Model of Computation

Ian Mackie

LIX, CNRS UMR 7161, École Polytechnique, 91128 Palaiseau Cedex, France

Abstract. Many new models of computation have emerged over the last years to meet the demands of modelling new features or new concepts in programming languages. Rarely do we find a model that becomes a programming paradigm. The aim of this paper is to show how a fine-grained model of computation can be used directly as a programming language. Moreover, the model can also exhibit visually properties of algorithms, such as space and time usage.

1 Introduction

Lafont [10] put forward interaction nets as a model of computation with a number of novel features. Unlike models such as Turing machines, cellular automata, λ-calculus or combinators, interaction nets model the computation process with constant time operations, and the model allows for parallelism (many steps can take place at the same time). The model therefore is an interesting one if we are interested in cost models of computation, and also taking advantage of possible parallelism.

The aim of this paper is to show through a number of examples that this model of computation can be used directly as a programming language. This idea is not new: the λ-calculus for instance has often been documented as a programming language with example programs showing how to represent numbers, simple data types and operations over this data. However, unlike the λ-calculus, interaction nets do not require sophisticated encoding of simple data types, and the main novelty of the model is that algorithms can be expressed in simple and efficient ways. By simple we mean that it resembles the algorithm we have in mind, and by efficient we mean that the computation steps correspond to what is required to compute the answer (the algorithm) rather than manipulating data structures that are part of the representation of the algorithm. Indeed, interaction nets can be seen as modelling internal representations of data, and as such is not weighed down by textual syntax.

Interaction nets are a particular kind of rewriting system (cf. term rewriting systems [9,2]). They were put forward as a generalisation of proof nets which are a graphical representation of linear logic proofs [4]. The theory and language have now developed into a powerful programming paradigm [6]. In this paper we put forward a variant of this paradigm as a formalism for programming with models. We present a number of example algorithms directly in the model to demonstrate that the approach is appropriate for algorithm design in addition to providing an aid to debug programs. The examples demonstrate also that the

© Springer-Verlag Berlin Heidelberg 2010

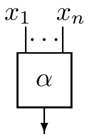
approach is appropriate for learning and understanding algorithms, as well as visualising a number of properties.

Related works. Lafont demonstrated that interaction nets are a graphical programming language when they were introduced [10]. In particular, programming examples involving lists were given (for instance an append operation). Numbers were also used, using the constructors zero and successor. Other people, notably Lippi [11] investigated further the programming paradigm, but still based on the original definition of interaction nets. He gave a sorting algorithm and an implementation of the Towers of Hanoi. The current paper develops programming further with an extended version of interaction nets which includes built-in data types such as numbers. This approach allows for a more economic representation of data, and thus allows for easier comparison with other programming languages.

Structure. The rest of this paper is organised as follows. In the next section we briefly recall the rewriting system of interaction nets, generalised to include data types. In Section 3 we illustrate the main ideas of the paper through two sorting algorithms. Section 4 looks at Dijkstra's partition algorithm, before giving quicksort and selection sort as the final examples. Section 5 is a discussion about the direction of this work and perspectives. We conclude in Section 6.

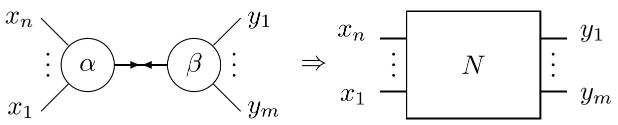
2 Interaction Nets

Here we define the rewrite system, which is a generalisation of interaction nets found in the literature [10]. We have a set Σ of names of the nodes in our diagrams. Each node has an arity *ar* that determines the number of *auxiliary ports* that the node has. If $ar(\alpha) = n$ for $\alpha \in \Sigma$, then α has $n + 1$ *ports:* n auxiliary ports and a distinguished one called the *principal port*.



Nodes are drawn variably as circles, triangles or squares. The position of the ports on the nodes is not important, but the order is. They optionally have an attribute, which is a value of base type: integers and booleans. We write the attribute in brackets after the name: e.g. $c(2)$ is a node called c which holds the value 2. A *net* built on Σ is an undirected graph with nodes at the vertices. The edges of the net connect nodes together at the ports such that there is only one edge at every port. A port which is not connected is called a *free port*.

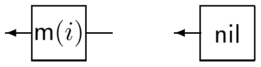
Two nodes $(\alpha, \beta) \in \Sigma \times \Sigma$ connected on their principal ports form an *active pair*, which is a reducible expression (redex). A rule $((\alpha, \beta) \Rightarrow N)$ replaces the pair (α, β) by the net N. All the free ports are preserved during reduction, and there is at most one rule for each pair of nodes. The following diagram illustrates the idea, where N is any net built from Σ.



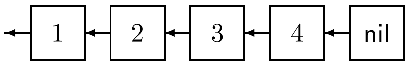
If either (or both) of the nodes are holding a value, then we can use these values to give different right-hand sides of the rule by labelling the arrow with a condition. The condition can be built out of the usual boolean operators ($<$, $>$, $=$, $!=$, etc.). However, the conditions must be all disjoint (there cannot be two rules which can be applied). Each alternative must of course give a net satisfying the property given above for the case without attributes. The most powerful property that this system has is that it is one-step confluent: the order of rewriting is not important, and all sequences of rewrites are of the same length and equal (in fact they are permutations). This has practical aspects: the diagrammatic transformations can be applied in any order, or even in parallel, to give the correct answer. One of the consequences of this is that once a program is written, the strategy of the algorithm is fixed up to permutation (i.e. the efficiency of the algorithm is fixed).

3 Insertion Sort

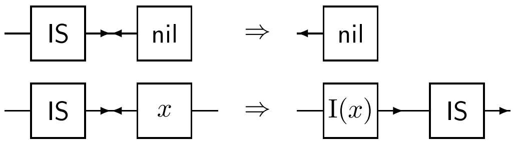
Insertion sort is one of the first sorting algorithms that we learn, and has been well studied in all programming languages. We first explain how to represent data using interaction nets. We can represent a memory location, containing an integer i, simply as a node holding the value. This can then be used to give the representation a list of nodes, with the addition of a nil node.

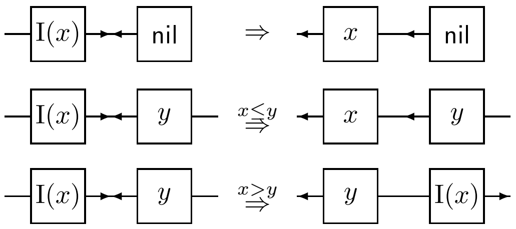


In the diagram above, the node m has one principal port that will be used to interact with it, and one auxiliary port to connect to the remaining elements of the list. The nil node just has one principal port, and no auxiliary ports. To simplify the diagrams, we often just write the contents of the node and omit the name when no confusion will arise. For example, here is a list of 4 elements:

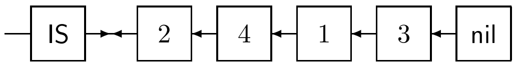


Note that this representation of a list will only allow rewrites through the first element, but just like in other languages, other variants can be implemented. We can now implement insertion sort over this data structure, then we compare with other paradigms.





The above five rules fully describe the algorithm: there is no other computational machinery required. The example net below can be used to illustrate the dynamics, which we leave for the reader.



This algorithm can be written in Prolog, Haskell and C for example. See Figure 1.

```
insertion_sort(Xs, Ys):-
  insertion_sort_1(Xs, [], Ys).

insertion_sort_1([], Ys, Ys).
insertion_sort_1([X|Xs], Ys0, Ys):-
  insert(Ys0, X, Ys1),
  insertion_sort_1(Xs, Ys1, Ys).

insert([Y|Ys], X, [Y|Zs]):-
  Y < X, !, insert(Ys, X, Zs).
insert(Ys, X, [X|Ys]).
```

Alternatively, in Haskell:

```
insert e [] = [e]
insert e (x:xs)
  | e < x     = e : (x:xs)
  | otherwise = x : (insert e xs)
```

```
iSort lst = iSort' lst [] where
  iSort' [] lst = lst
  iSort' (x:xs) lst =
    iSort' xs (insert x lst)
```

And finally in C:

```
void iSort(int a[], int len){
 int i, j, v;
 for(i = 1; i < len; i++) {
  v = a[i];
  for(j=i-1; j>=0 && a[j]>v; j--)
   a[j + 1] = a[j];
  a[j + 1] = v;
 }
}
```

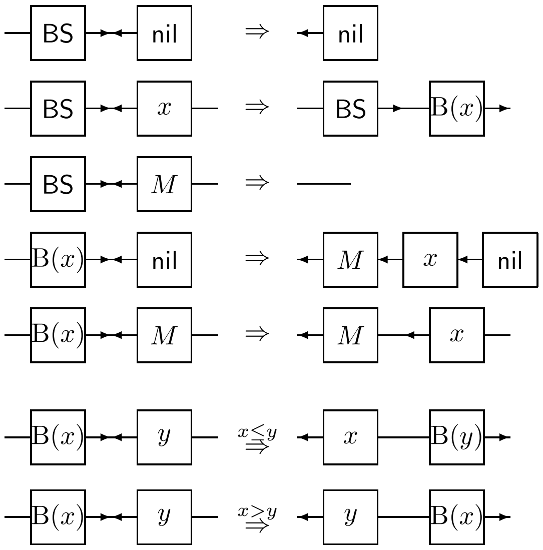
Fig. 1. Insertion sort

Our focus here is to draw out the advantages of the visual model, but of course, each programming paradigm has its own advantages. The visual model is graphical, so removed much of the syntax of the other languages. The data structures are also different (lists, arrays, etc.) which make a fair comparison difficult. However, there are much more significant differences that we want to harness here.

– Insertion sort can be implemented in-place. It is not clear if any of the programming languages have this property. However, the interaction net

implementation does implement the algorithm in-place, and we can see this by observing the rules: each rule preserves the data structure size, and since the program and data are in the same formalism this is an in-place algorithm.
- The careful reader might have noticed that there is a choice of application of the rules in the example given. However, the order of application does not matter, as this system of rewriting is deterministic (all reductions lead to the same answer). However, the algorithm is in-place no matter what order we take, which is very strong property.

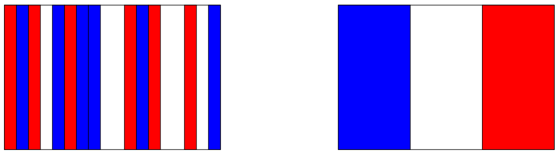
We show a second sorting algorithm to illustrate another feature of the formalism. Bubble sort can be encoded by the following rules, where the node M is a marker representing the part of the list which has already been sorted.



Comparing algorithms in any programming language is usually a study of complexity of the underlying algorithms. Here, however, we can compare algorithms in the model quite directly, and see relationships and efficiency differences directly in the rules. Bubble sort, without the rules for the marker M, can be related very directly to insertion sort: the algorithmic difference is seen in the second rules of each algorithm—insertion sort inserts an element into a sorted list, whereas bubble sort inserts an elements into a list that has yet to be sorted. These connections and differences are hard to see in traditional languages, but are clear in this model.

4 Partition and Sorting

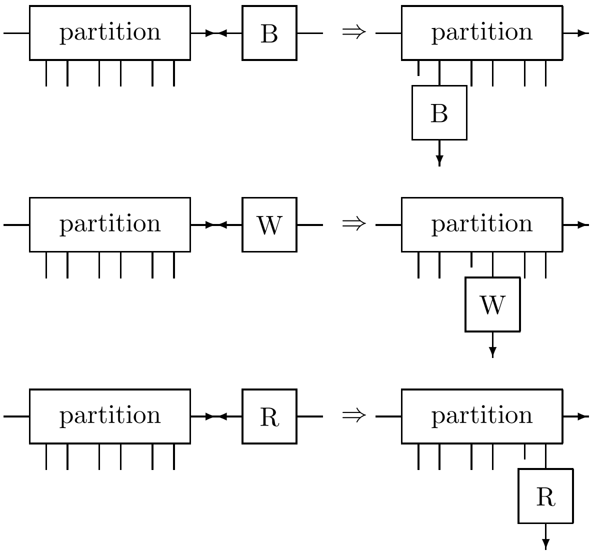
Dijkstra's partition algorithm, known as the Dutch national flag algorithm, is a 3-way partition algorithm of complexity $O(n)$. Here we recast the algorithm using the French national flag. The idea is to sort the flag from strips, as shown on the left below to give the flag on the right:



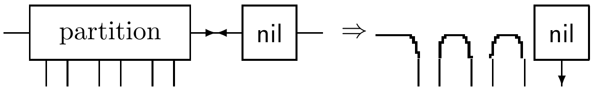
This is a popular algorithm that benefits from animation. Usually, implemented using an array, and several pointers to allow the array to be partitioned. Below we give a solution in an imperative language for arrays:

```
static void restoreFlag() {
  int white = 0;
  int grey = 0;
  int red = flag.length;
  while (grey < red) {
    switch (flag[grey]) {
    case blueColor :
      swap(white,grey);
      white++; grey++; break;
    case whiteColor :
      grey++; break;
    case redColor :
      swap(grey,red-1);
      red--; break;
    }
  }
}
```

It is clear that this is a linear algorithm (just one while-loop is used), but it is much less clear that it works. Typically, we would write this program using assertions and while-loop invariants to ascertain the correctness of the algorithm, or at least to have any idea of what is happening (something different happens for each colour). Using the visual model we can reproduce the algorithm with the same ideas, and the same complexity, but with an additional advantage that we perform essentially the same operation for each case. The first three rules deal with the three different colours, and partitions them:

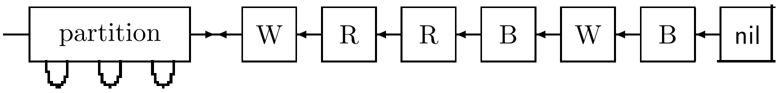


The final rule deals with the end of the list, which just concatenates the stored elements to produce one list. Note in particular that one constant time rewrite rule concatenates three lists.

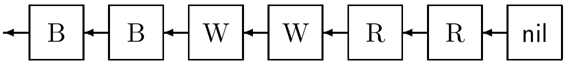


This algorithm is clearly linear—each element is examined once. It is also interesting to note that we can extend this algorithm to any number of partitions and still maintain a linear algorithm (which is not the case with arrays). Comparing with the imperative solution, we do not need all the variables to partition the array, as we created lists for each partition, so the program is simpler to follow. However, it is also in-place by observing the rules.

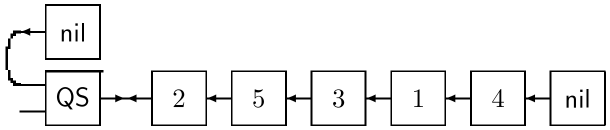
To use this program, we need to begin with a correct application of the partition node. The following diagram shows how the program is used with a list of six elements:



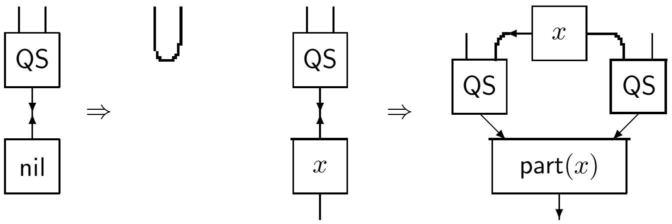
After seven rewrite steps (one for each element of the list, plus one for nil), we get the following net in normal form, which is the correctly partitioned list:



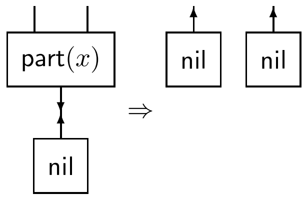
Quicksort. The partition algorithm above can be simplified for use in the quicksort algorithm, where only two partitions are needed. One way in which we can implement this algorithm in this model is given by the following starting configuration, where we show a list of elements to be sorted.

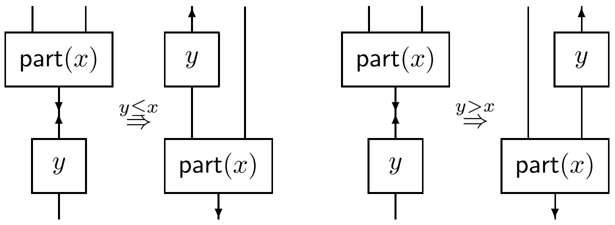


The QS node has two auxiliary ports: one for the result of sorting the list, and a second which is used like an accumulating parameter. This additional port is linked to a nil node, and the following two rules give the main structure of the algorithm:



The first rule sorts the empty lists to give the accumulated list, and the second rule does all the work in quicksort: partition the list, and append the sub-lists to create a sorted list. The structure of the second rule brings out the visual aspect of the language, in that this rule is describing the algorithm. The remaining rules that we need to complete quicksort are given below, which partition a list to generate the two lists required.




part(x)
y
$\overset{y \leq x}{\Rightarrow}$
y
part(x)
part(x)
y
$\overset{y > x}{\Rightarrow}$
y
part(x)

We end this section with one final sorting algorithm: selection sort. The rules are given in Figure 2, where the node SS is used to start the sorting. It is included here to demonstrate the ideas that can be used to implement a swap of elements that are not adjacent in the list.

5 Discussion

Interaction nets are a model of computation, and as an alternative to general graph rewriting, they have a number of striking features. In particular, in graph rewriting, locating (by graph matching) a reduction step is an expensive operation, as is finding the next redex. In interaction nets these problems are overcome. Matching only involves two nodes of the graph (as specified in the definition of a rewrite rule), and maintaining a list of redexes, updated by looking locally after each rewrite, is a simple addition to any implementation. Just like term rewriting systems, interaction nets are user defined. Because we can write systems which correspond to rewrite systems we can see them as specification languages. But, because we must also explain all the low-level details (such as copying and erasing) then we can see them as a low-level operational semantics/implementation language. There are interesting aspects of interaction nets for parallel evaluation.

The aim of this paper is to demonstrate that a model of computation can be used directly as a programming language. The application of the model does not lead to inefficient encodings of data, as is the case with λ-calculus for instance. Indeed, we have demonstrated that the resulting programs are as efficient, if not

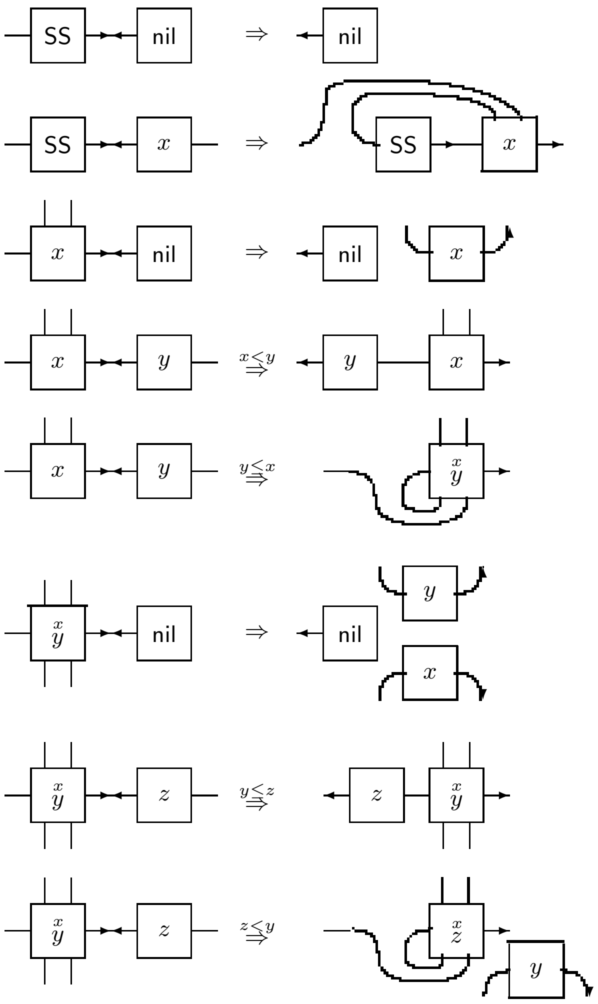


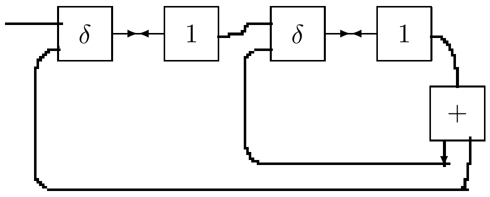
Fig. 2. Selection sort

more efficient, than current programming languages. The further advantages are that properties of algorithms are visible from directly from the program. Specifically, the local property that each rule preserves the size of the data structure tells us that globally this algorithm is implemented in-place. This kind of property is very hard to obtain from other models of computation, and requires sophisticated static analyses for programs.

The development of a number of case studies does not prove the usefulness of this model, but it does give a significant step towards building a framework where we can program with this paradigm and in a more efficient way than in traditional languages. There are a few directions that are currently being investigated.

1. A programming language for interaction nets has been developed, together with a compiler for the language to enable some of these ideas to be tested. The language is textual at the time of writing, and graphical tools to provide a visual programming environment are not yet available. Once we have completed the visual environment, we can demonstrate better the principles outlined in this paper, and compare with other visual programming languages, such as [13,8,1].
2. A second theme of work is to investigate translating (compiling) other programming languages into interaction net. This will allow properties of the underlying algorithm to be visualised in the model, but will also serve as a framework where programs can be transformed (optimised) to make better use of the model. The current state of this technology is a number of hand-coded examples that illustrate the feasibility of the approach, but much implementation work remains.
3. Finally, related to the previous point, translating other models of computation directly into interaction nets provides an alternative approach. This has been very successful for models such as the λ-calculus [5,12], where significantly more efficient implementations have been developed. Using this approach, any language modelled on the λ-calculus can be implemented through the translation.

   This approach is not limited to the λ-calculus. For instance term rewriting systems have been studied through interaction nets [3]. Other models such as process networks [7] can also be represented. An example of this is generating the Fibonacci sequence as a stream of numbers is the following net:



We leave the details of the rules to the interested reader.

6 Conclusions

Interaction nets are a model of computation that capture computation particularly well and are visually appealing. They are equivalent to Turing Machines and the λ-calculus, but they offer computation steps that are known, constant-time operations, and are also local so that no two rewrite rules overlap.

The purpose of this paper is to demonstrate through several examples programs, that programming directly in this model of computation is not only possible, but leads to implementations of algorithms that are simpler than many programming languages, and more efficient. Moreover, the model allows certain properties, such as time and space use to be seen visually in the rules. Visual editors are now being developed to compare with textual implementations of interaction nets to identify the best way forward with this programming paradigm.

The point that interaction nets can be implemented efficiently, and algorithms can be represented efficiently in interaction nets makes this formalism an interesting model of computation for further investigation as a programming language.

References

1. Dami, L., Vallet, D.: Higher-order functional composition in visual form. Technical report (1996)
2. Dershowitz, N., Jouannaud, J.-P.: Rewrite Systems. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science: Formal Methods and Semantics, vol. B, North-Holland, Amsterdam (1989)
3. Fernández, M., Mackie, I.: Interaction nets and term rewriting systems. Theoretical Computer Science 190(1), 3–39 (1998)
4. Girard, J.-Y.: Linear Logic. Theoretical Computer Science 50(1), 1–102 (1987)
5. Gonthier, G., Abadi, M., Lévy, J.-J.: The geometry of optimal lambda reduction. In: Proceedings of the 19th ACM Symposium on Principles of Programming Languages (POPL 1992), January 1992, pp. 15–26. ACM Press, New York (1992)
6. Hassan, A., Mackie, I., Sato, S.: Interaction nets: programming language design and implementation. In: Proceedings of the seventh international workshop on Graph Transformation and Visual Modeling Techniques (March 2008)
7. Kahn, G.: The semantics of a simple language for parallel programming. Information Processing 74 (1974)
8. Kelso, J.: A Visual Programming Environment for Functional Languages. PhD thesis, Murdoch University (2002)
9. Klop, J.-W.: Term rewriting systems. In: Abramsky, S., Gabbay, D.M., Maibaum, T.S. (eds.) Handbook of Logic in Computer Science, vol. 2, pp. 1–116. Oxford University Press, Oxford (1992)
10. Lafont, Y.: Interaction nets. In: Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL 1990), January 1990, pp. 95–108. ACM Press, New York (1990)
11. Lippi, S.: Programming with interaction nets. Technical report, Logic and Interaction school (2002)
12. Mackie, I.: Efficient λ-evaluation with interaction nets. In: van Oostrom, V. (ed.) RTA 2004. LNCS, vol. 3091, pp. 155–169. Springer, Heidelberg (2004)
13. Reekie, H.J.: Realtime Signal Processing – Dataflow, Visual, and Functional Programming. PhD thesis, University of Technology at Sydney (1995)

An Automata-Theoretic Characterization of the Chomsky-Hierarchy

Benedek Nagy

Department of Computer Science, Faculty of Informatics
University of Debrecen, Debrecen, Hungary
nbenedek@inf.unideb.hu

Abstract. In this paper a new family of stateless (non-deterministic) pushdown automata are used to accept languages of the Chomsky hierarchy. Having only a stack with at most 1 symbol the regular languages can be recognized. The usual pushdown automata accept the context-free languages. The extended version which uses additional half-translucent shadow symbols accept the context-sensitive languages. Finally, allowing a kind of λ-transitions the automata accept any recursively enumerable languages.

1 Introduction

The theory of formal languages and automata is one of the most important and most developed fields of theoretical computer science. Some decades ago it was said by Aho, Salomaa etc. that formal language theory is "the flower of theoretical computer science". The Chomsky hierarchy of languages, the generative grammars and the corresponding classes of automata are well-known. The regular languages are recognized by (deterministic or non-deterministic) finite automata. The context-free languages are accepted by (non-deterministic) pushdown automata. The automata used to accept the context-sensitive languages are the linear bounded Turing machines. Every recursively enumerable language is accepted by a Turing machine ([12]).

In this paper variations of the pushdown automata are used. It is known, that its stateless non-deterministic version accepts exactly the context-free languages. We will consider a translation of the finite automata to stateless pushdown automata (and back) using at most 1 symbol in their pushdown stack. To go beyond context-free, extended versions of pushdown automata will be presented. There are several known extension of pushdown automata: In two-way pushdown automata the reading head on the input tape can move in both directions [3]. In the stack automata it is allowed to scan inside of the stack in read only mode [4]. Higher level pushdown automata [6] have stacks of stacks. Deep pushdown automata may make expansions deeper in the pushdown stack [7] etc. None of these devices (having only 1 stack) fits to the context-sensitive languages, therefore our work fills this gap. There are several characterizations

© Springer-Verlag Berlin Heidelberg 2010

of context-sensitive languages known: They are exactly the frontiers of recognizable picture languages [5]. They coincide with the languages generated by one-dimensional cellular automata [1]. Moreover it is proved that the rational graphs trace context-sensitive languages ([8]) by linear bounded Turing machines. The class of recursively enumerable languages has several characterizations as well. In this paper our approach is closely related to the usual derivation-trees for context-free grammars. We use the context-sensitive and recursively enumerable grammars as extensions/generalisations of the context-free grammar. In this paper, a new extension of the pushdown automata will be shown, which accepts exactly the recursively enumerable and with a kind of restriction, the context-sensitive languages. The work of the pushdown automata is based on the left-most derivation in context-free grammars. Generally the pure concept of derivation trees does not work for context-sensitive grammars. Therefore a generalised concept is used working for grammars in Penttonen normal form including additional context-edges to signal context-sensitive dependencies. We introduce a new normal form, the so-called context normal form with special (context) non-terminals. The left-most derivations cannot be defined in the usual (sentential form) sense for non context-free grammars without loosing the generative power ([12]). Opposite to this fact the derivation graphs can be obtained by a left-most construction. In several applications, such as in parsing technologies, the existence of left-most derivation in context-free grammars is crucial. Our work is also important in this point of view, since our left-most construction can effectively be used in non context-free case, e.g., this left-most construction and the context normal form of the formal grammars give the basis of the work of our shadow-pushdown automata.

2 Preliminaries

In this section we recall some basic definitions, notations and facts about the Chomsky hierarchy. For more details see [4,12].

A grammar is a quadruple $G = (N, T, S, H)$, where N, T are the non-terminal and terminal alphabets, with $N \cap T = \emptyset$; they are finite sets. $S \in N$ is a special symbol, called initial letter. H is a finite set of pairs, where a pair is written in the form $v \rightarrow w$ with $v \in (N \cup T)^*N(N \cup T)^*$ and $w \in (N \cup T)^*$. H is the set of derivation rules. A sequence of letters $v \in (N \cup T)^*$ is called a string, while we refer $u \in T^*$ as a (terminal) word. Let $v, w \in (N \cup T)^*$. Then $v \Rightarrow w$ is a direct derivation if and only if there exist $v_1, v_2, v', w' \in (N \cup T)^*$ such that $v = v_1v'v_2$, $w = v_1w'v_2$ and $v' \rightarrow w' \in H$. The derivation $v \Rightarrow^* u$ is the reflexive and transitive closure of the direct derivation. The sentential forms are those strings which can be derived from S. The language generated by a grammar G is the set of all terminal words that can be derived from the initial letter: $L(G) = \{w | S \Rightarrow^* w \wedge w \in T^*\}$. Two grammars are equivalent if they generate the same language modulo λ (where λ denotes the empty word). In this paper we mostly deal with λ-free languages and do not care about if $\lambda \in L$ or not.

A grammar is type 3, or regular, if each derivation rule has one of the following forms: $A \rightarrow a$, $A \rightarrow aB$, $A \rightarrow \lambda$; where $A, B \in N$ and $a \in T$. (In this paper the capitals $A, B, C, ..$ and S usually refer for non-terminals, while lower case letters $a, b, c, ..$ refer for terminal symbols.) A language is regular if there is a type 3 grammar that generates it. Every regular language can be generated by only rules of type $A \to aB$ and $A \to \lambda$. We will refer to such grammars as regular grammars in normal form.

A grammar is context-free (type 2) if each of its rules has the form $A \to v$ with $A \in N$ and $v \in (N \cup T)^*$. A language is context-free if it can be generated by a context-free grammar. For each context-free grammar there is an equivalent grammar having rules of forms $A \to BC, A \to a$ (Chomsky normal form).

A grammar is context-sensitive (type 1) if each of its derivation rules has the following form $v_1 A v_2 \to v_1 w v_2$, with $v_1, v_2 \in (N \cup T)^*$, $A \in N$ and $w \in (N \cup T)^* \setminus \{\lambda\}$. A language is context-sensitive if it can be generated by a context-sensitive grammar. Now we recall the Penttonen normal form ([11]), where it was called one-sided normal form. Every (λ-free) context-sensitive language can be generated by a grammar having rules of the forms $A \to BC$, $AB \to AC$, $A \to a$.

Generative grammars having no restriction on the form of the rules are phrase-structure (type 0) grammars. They generate the class of recursively enumerable languages. Every recursively enumerable language can be generated by a grammar having rules only of the forms $A \to BC$, $AB \to AC$, $A \to a$ and $A \to \lambda$. This normal form is the Penttonen normal form for type 0 grammars.

Context-free derivations are represented by derivation-trees (this is one of the main reasons why context-free grammars are popular and frequently used). The leaves of the tree (from left to right) give the actual sentential form or the derived word. Based on Penttonen normal form derivations in non context-free grammars can also be represented in similar tree-like graphs [9]. We consider grammars in which every non context-free rule is of the form $AB \to AC$. The used context-free productions are represented in these graphs in the same way as in the derivation trees. At (context-sensitive) rules type $AB \to AC$ the neighborhood of the replaced non-terminal is important, therefore a new type of edges is introduced. These context-edges connect the node of the graph which play the context (A) at the derivation step to the node which is substituted (B) by the production. For instance, let $G = (\{S, A, B, C, D, E\}, \{a, b, c\}, S, \{S \to AD, A \to AC, A \to a, B \to DC, B \to b, B \to \lambda, C \to AB, C \to DE, C \to c, D \to AC, D \to BC, E \to CC, BA \to BB, BD \to BE, BE \to BA, BE \to BB, EB \to EE, EC \to EB\})$. Having a derivation it is very easy to make its graph representation, as it can be seen on Figure 1, where context edges are broken (dashed) arrows. These derivation graphs have the following important properties. If a context edge goes from a node α to a node β, then they are neighbors in the following sense. They are in neighbor branches (see [9] for detailed formal description) or λ is derived from the branches that are between them – as it can be seen on the figure. It corresponds to the fact that the used context-sensitive rule could be applied, therefore the nodes α and β represent neighbor symbols of the corresponding sentential form. Several context edges can start from a node, but there is at most 1 context edge goes into any of them. There are no context-edges crossing each other, i.e., if there is a context-edge from a node α to a node β, then there is not any context edge to a node γ having ancestor β from any ancestor of α and from any node between the branches of α and β. Considering the graph theoretical meaning of these two properties, we could roughly say that context-edges cannot cross other edges (nor traditional derivation edges, nor other context-edges).

Derivation graphs with the above properties exactly correspond to real derivations (see [9] for more details), i.e., we have:

Theorem 1. *For a grammar G (in Penttonen normal form) there exists a derivation-graph for a string w if and only if $S \Rightarrow^* w$ (i.e., w is a sentential form).*

For technical purpose we introduce a new normal form for context-sensitive and phrase-structure grammars. Let $G = (N, T, S, H)$ be a grammar in Penttonen one-sided normal form that generates L. One could consider the grammar $G_1 = (N_1, T, S, H_1)$, where $N_1 = N \cup \overline{N}$ with $\overline{N} = \{\overline{A} \mid A \in N\}$ and $H_1 = \{A \rightarrow u\overline{A} \mid A \rightarrow u \in H, A \in N, u \in (N \cup T)^*\} \cup \{(\overline{A}B \rightarrow \overline{A}C\overline{B} \mid AB \rightarrow AC \in H\} \cup \{\overline{A} \rightarrow \lambda \mid \overline{A} \in \overline{N}\}$. Then, the classical algorithm for deleting erasing (now type $\overline{A} \rightarrow \lambda$) rules yields the grammar in our *context normal form*: $G_2 = (N_1, T, S, H_2)$, where: $H_2 = \{A \rightarrow u\overline{A}, A \rightarrow u \mid A \rightarrow u \in H, A \in N, u \in (N \cup T)^*\} \cup \{\overline{A}B \rightarrow \overline{A}C\overline{B}, \overline{A}B \rightarrow \overline{A}C, \overline{A}B \rightarrow C\overline{B}, \overline{A}B \rightarrow C \mid AB \rightarrow AC \in H\}$. One can see that for any successful derivation in G_2, a symbol $\overline{A}$ appears only if it is used subsequently in the derivation as a context. (We call the elements of $\overline{N}$ shadow symbols, as they are not real non-terminals, the derivation is already continued form them, but they are only remained 'shadows' of the corresponding non-terminals.) Since the shadow symbols used only as contexts it is obvious that the difference between grammars generating context-sensitive and recursively enumerable languages is the existence of rules of erasing non shadow symbols as in Penttonen normal form.

Now we recall the concept of stateless pushdown automata. A (non-deterministic, stateless) pushdown automaton M consists of an input tape, a finite control and a pushdown stack (we consider it as a string having right the top-most part), formally it is a system $(\Sigma, \Gamma, \delta, Z_0)$, where Σ is a finite alphabet called the input alphabet, Γ is a finite alphabet called stack alphabet, $Z_0 \in \Gamma$ is a special symbol, initially in the bottom of the stack and δ is a mapping form $(\Sigma \cup \{\lambda\}) \times \Gamma$ to finite subsets of Γ^*. The interpretation of δ is the following. The relation $z \in \delta(\lambda, Z)$ with $Z \in \Gamma$ and $z \in \Gamma^*$ means that M can replace the top of the stack Z by z without moving its head. The relation $z \in \delta(a, Z)$ with $a \in \Sigma$, $Z \in \Gamma$ and $z \in \Gamma^*$ means that M can erase the top of the stack Z and put z to the top of the stack advancing the input head on a symbol a. If $z = \lambda$, then the content of the stack is decreasing. A configuration of a pushdown automaton M is a pair (v, z) where $v \in \Sigma^*$ is the unprocessed part of the input word and $z \in \Gamma^*$ is the actual content of the stack. A word $w \in \Sigma^*$ is accepted by M if there is a finite sequence of configurations starting by (w, Z_0) and finishing by (λ, λ) such that in each step the configuration is given by δ. The automaton accepts a word w simulating a left-most derivation in the grammar. A derivation is called left-most (for context-free grammars) if the first non-terminal of the sentential form is replaced in each step by an appropriate derivation rule. The set of all accepted words forms the accepted language of the automaton. It is well-known in formal language and automata theory that the class of (non-deterministic stateless) pushdown automata accepts exactly the context-free languages. In the literature pushdown automata with several states are also known. In this paper we will use pushdown automata only with one state, they are universal among pushdown automata.

3 Variations of Pushdown Automata for the Chomsky Hierarchy

In this section, first, the concept of the new stateless pushdown automata is introduced. Then we show that restricted versions accept regular, context-free and context-sensitive languages. The accepting power of the general version is shown to coincide with the class of recursively enumerable languages.

Definition 1. *A shadow-pushdown automaton M consists of an input tape, a finite control and a stack with two kinds of symbols, formally it is a system $(\Sigma, \Gamma \cup \overline{\Gamma}, \delta, Z_0)$, where Σ is the input alphabet, Γ is the stack alphabet and it is extended by (half-translucent) symbols $\overline{\Gamma} = \{\ \overline{A}\ |A \in \Gamma\}$ (they are called the shadow symbols of the original ones). $Z_0 \in \Gamma$ is a special stack symbol as for pushdown automata. The mapping δ goes from $(\Sigma \cup \{\lambda\}) \times \Gamma \times (\overline{\Gamma} \cup \{\lambda\})$ to finite subsets of $\overline{\Gamma}\Gamma^* \cup \Gamma^* \cup \Gamma^*\overline{\Gamma} \cup \overline{\Gamma}\Gamma^*\overline{\Gamma}$ as we detail below. The interpretation of δ is the following. The top $A \in \Gamma$ symbol is accessed (together with the shadow-symbol directly above). There are five possible types of movements:*

(a) $\{\lambda, \overline{A}\} \subseteq \delta(a, A, \lambda)$. The transition is possible if there is no shadow symbol above the top non-shadow symbol (A). The head reads the letter a from the input tape and the top symbol A is replaced by its shadow-symbol $\overline{A}$ or deleted from the stack.
(b) $\{B, \overline{A}B\} \subseteq \delta(a, A, \lambda)$. This transition is allowed if there is no shadow symbol above the top non-shadow symbol. The head reads an input letter a, the top non-shadow symbol A is replaced by its shadow or deleted and the non-shadow symbol B is pushed on the top of the stack.
(c) $\{CB, \overline{A}CB\} \subseteq \delta(\lambda, A, \lambda)$. In these steps the symbols C and B are put immediately above the top symbol A, and A is replaced by its shadow or deleted. All shadow symbols that were above A remain above B.
(d) $\{\lambda, \overline{A}\} \subseteq \delta(\lambda, A, \lambda)$. The top non-shadow symbol A is replaced by the corresponding shadow symbol or deleted from the stack.
(e) $\{C, \overline{B}C, \overline{B}C\overline{A}, C\overline{A}\} \subseteq \delta(\lambda, B, \overline{A})$. This step can be used, if there is a shadow symbol $\overline{A}$ directly above the top symbol B in the stack. The symbol C is put immediately above the top non-shadow symbol B, B is replaced by its shadow or deleted, and the used shadow symbol $\overline{A}$ may be deleted from the stack.

A configuration is a pair (r, z) where $r \in \Sigma^*$ the unprocessed part of the input word and $z \in (\Gamma \cup \overline{\Gamma})^*$ is the actual content of the stack. A word $w \in \Sigma$ is accepted *by the shadow-pushdown automaton, if there is a sequence of transitions starting from* (w, Z_0) *and finishing by* (λ, λ) *according to the mapping* δ *in each step. The set of accepted words form the accepted language.*

We name and describe the five different types of (allowed) movements of the automata.

(a) terminating: an input letter is processed and the number of non-shadow stack symbols is decreasing by 1 in the stack.
(b) reading an input letter: it is processed, but the number of non-shadow symbols does not change in the stack.
(c) extending the stack: the number of stack symbols is increasing (only in these steps).
(d) empty-deletion: the number of non-shadow stack symbols is decreasing without reading an input letter.
(e) shadow using: only these steps use shadow symbols, a shadow symbol directly above the top non-shadow symbol is needed to these transitions.

Steps of type (e) extend the power of the traditional pushdown automata; since no other steps depend on the shadow symbols, they work in the same way in the traditional pushdown automata without creating and deleting shadow symbols.

In the next subsections restricted forms of these automata will be shown.

3.1 Pushdown Automata for Regular Languages

Let us restrict the automata defined previously in the following way. Only steps of types (b) and (d) are allowed.

Let a regular grammar $G = (N, T, S, H)$ be given in normal form such that it generates L. The automaton $(\Sigma, \Gamma \cup \overline{\Gamma}, \delta, Z_0)$ with $\Sigma = T, \Gamma = N, Z_0 = S$ accepts exactly L, where δ is given in the following way: $(B) \in \delta(a, A, \lambda)$ if $(A \to aB) \in H$ and $(\lambda) \in \delta(\lambda, A, \lambda)$ if $(A \to \lambda) \in H$.

Starting with S in the stack steps type (b) keep exactly 1 non-shadow symbol in the stack, while a step type (d) make the stack empty. The automaton accepts L. Moreover the construction works also in the opposite direction. Having a shadow-pushdown automaton with only steps type (b) and (d) a regular grammar can be constructed which generates the accepted language. The shadow symbols are never used in any steps of this automaton.

Actually, if the shadow symbol was stored in every step (i.e., transitions $(\overline{A}B) \in \delta(a, A, \lambda)$ and $(\overline{A}) \in \delta(\lambda, A, \lambda)$ were used), then they will keep a historic trace of the way of accepting (and so, it will show how the word can be generated/accepted). The machine would work till there is a non-shadow symbol in the stack.

Without using shadow symbols a traditional stateless pushdown automaton is used. The work of this automaton simulates a non-deterministic finite automaton. The pushdown automaton has no states, but the stack symbols and the states of a corresponding finite automaton has a one-to-one correspondence. The final states coincide with symbols which are used in transitions type (d).

Thus, stateless pushdown automata with only stack having length (depth) at most 1 accept exactly the regular (type 3) languages. We note here that any pushdown automata having stack of bounded depth (i.e., there is an $n \in \mathbb{N}$ such that the stack never can contain more than n symbols) recognize exactly the regular languages. The finitely many possible contests of the stack can be stored by states in a finite automaton. However, for complexity theoretical view, they are interesting devices [2].

3.2 Accepting Context-Free Languages

Without step (e) the shadow symbols are not used at all, therefore the work of the shadow-pushdown automata is effectively the same as the the work of the pushdown automata. We note, that every λ-free context-free language can be accepted by a shadow-pushdown automaton using only transitions of types (a) and (c). These two types of transitions refer to forms of the rules in Chomsky normal form.

Now we are turning to more interesting cases, where shadow symbols are also used.

3.3 The Case of Context-Sensitive Languages

Let us consider shadow-pushdown automata with allowed steps of the types (a), (c) and (e). The context normal form is used to connect shadow-pushdown automata and generative grammars: let the following coincidence be given $\Sigma = T, \Gamma = N, Z_0 = S$. The possible movements of the shadow-pushdown automata correspond to the three kinds of derivation steps of the grammar in normal form.

- steps type (a) correspond to rules type $A \to a$ and $A \to a\overline{A}$,
- steps type (c) correspond to rules type $A \to BC$ and $A \to BC\overline{A}$,
- steps type (e) correspond to context-sensitive rules type $\overline{A}B \to C, \overline{A}B \to \overline{A}C, \overline{A}B \to C\overline{B}$ and $\overline{A}B \to \overline{A}C\overline{B}$.

3.4 The General Case: The Whole Recursively Enumerable Class

The shadow-pushdown automaton is a formally defined system and it can be simulated by a Turing-machine, therefore only recursively enumerable languages can be accepted. In other way around, the possible steps refer for derivation rules. In addition to the context-sensitive case there are:

- steps type (b) correspond to rules type $A \to aB, A \to aB\overline{A}$,
- steps type (d) correspond to rules type $A \to \lambda, A \to \overline{A}$,

Since every recursively enumerable language can be generated by rules in Penttonen, and so in context normal form, we can say that transitions of types (a), (c), (d) and (e) are enough to accept any recursively enumerable languages.

In the next section we show how and why the automata work.

4 The Work of the Shadow-Pushdown Automata

The introduction of shadow-symbols in context normal form and in shadow pushdown automata allow to keep tracks of the variables previously derived. They are used to represent and simulate context-sensitive derivation steps.

To understand how the automata work we introduce the concept of left-most construction of derivations. This construction is a method how the derivation graphs can be obtained. The left-most construction of a derivation graph is the following.

In each step the left-most non-terminal leaf symbol is used to provide new node(s) by derivation edge(s) (maybe using an in-context-edge from one of its (left) neighbor nodes).

The left-most construction is exactly the same order of construction as a left-most traversal goes in the tree (using the order given by the tree with the derivation edges). In this construction the already constructed left-hand-side part of the graph is never changed, but the right-most non-terminals of the graph may needed as contexts to build up the remaining part. One can imagine that the non-terminals have shadows to the right-neighbor branch. And when a context is needed one can choose among the non-terminals having shadow at this node.

We note here, that in regular case each derivation is a left-most construction itself, while in context-free case the concept of the usual left-most derivation coincides with the concept of left-most construction. Using the classical definition of left-most derivation only context-free languages can be obtained even if the grammar is phrase-structure [12]. Therefore in non context-free cases the left-most construction is more powerful than the (sentential) left-most derivation.

In our example presented in Figure 1 the left-most construction goes in the following way (using grammar in context normal form it is a left-most derivation):

$S \Rightarrow AD \Rightarrow ACD \Rightarrow aCD \Rightarrow aDED \Rightarrow aACED \Rightarrow aaCED \Rightarrow aaABED \Rightarrow aaaBED \Rightarrow aaab\overline{B}ED \Rightarrow aaab\overline{B}B\overline{E}D \Rightarrow aaab\overline{BBE}D \Rightarrow aaab\overline{BBE}BC \Rightarrow aaab\overline{BB}EC \Rightarrow aaab\overline{B}A\overline{E}C \Rightarrow aaabB\overline{E}C \Rightarrow aaabb\overline{BE}C \Rightarrow aaabb\overline{B}B \Rightarrow aaabb\overline{B}DC \Rightarrow aaabbEC \Rightarrow aaabbCCC \Rightarrow^* aaabbccc.$

Since the shadow-pushdown automata can access only the top non-shadow symbol and make only local changes the following theorem can be proved based on our context normal form.

Theorem 2. *The configuration (v, z) of a shadow-pushdown automaton working on the word $w \in \Sigma^*$ represents the sentential form uz^{-1} in the left-most derivation in corresponding context normal form grammar, where $u \in \Sigma^*$ is the processed part of w, i.e., $w = uv$; and z^{-1} is the content of the stack reading top down. Thus, the shadow-pushdown automata simulate the left-most derivations (and so, construction of derivation trees). The accepting power of the automata is equal to the generative power of the corresponding grammar class.*

Proof. We use induction by steps. At the starting configuration (w, S) the condition holds. Now assume that it is true for the configuration (v, z). There are five cases (by definition) according to the type of the next transition to get (v', z') and u'.

The shadow-pushdown automata have the following operations coinciding to various types of derivation rules:

- (a) For rules of type $A \to a$:
  $(\lambda) \in \delta(a, A, \lambda)$ or $(\overline{A}) \in \delta(a, A, \lambda)$. This step can be used if there is no shadow symbol above the top non-shadow A. It means that derivating a terminal symbol a branch is terminated (by terminal a). The new configuration will be: $u' = ua$, the first letter (that is an a is deleted from v to get v' and z' is obtained from z in the following way: The top symbol A is replaced by its shadow-symbol $\overline{A}$ or deleted. Actually if this letter A is needed later as a context, then $(\overline{A}) \in \delta(a, A, \lambda)$ is used, otherwise $(\lambda) \in \delta(a, A, \lambda)$ is used.
- (b) For rules of type $A \to aB$:
  $(B) \in \delta(a, A, \lambda)$ or $(\overline{A}B) \in \delta(a, A, \lambda)$. Since the automaton reads a letter from the input this step is allowed only if there is no shadow symbol on the top non-shadow symbol in the stack. In these steps there is also a terminated derivation branch with terminal symbol a: $u' = ua$, the first letter (that is an a is deleted from v to get v', furthermore z' is obtained from z by replacing the top A of the stack to B or $\overline{A}B$ depending on the fact that A is needed as a context in the derivation-tree or not.
  (*Remark*: There is no way to use context (shadow symbols) for the top symbol B.)
- (c) For rules of type $A \to BC$:
  $(CB) \in \delta(\lambda, A, \lambda)$ or $(\overline{A}CB) \in \delta(\lambda, A, \lambda)$. In these steps $u' = u$ and $v' = v$ and z' is obtained from z: the two non-shadow symbols (C and B) are inserted immediately above the top non-shadow symbol A according to their place in the graph; we delete symbol A from the stack, but it may needed as a context later, therefore we may leave its shadow ($\overline{A}$) in the stack instead of the original symbol.
  (*Remark*: The shadow symbols above them remain in the stack, because the set of possible context nodes of the next derivation step has not changed, they are needed later as contexts.)
- (d) For rules of type $A \to \lambda$:
  $(\lambda) \in \delta(\lambda, A, \lambda)$ or $(\overline{A}) \in \delta(\lambda, A, \lambda)$. This step is allowed even if there are shadow symbols on A in the stack. A branch is finished with λ, therefore the actual top shadow symbols are still possible contexts in the next branch: $u' = u$ and $v' = v$, z' is obtained from z by deleting or replacing the top non-shadow symbol A to the corresponding shadow symbol $\overline{A}$.
- (e) For rules of type $AB \to AC$:
  $(C) \in \delta(\lambda, B, \overline{A})$, $(\overline{B}C) \in \delta(\lambda, B, \overline{A})$, $(C\overline{A}) \in \delta(\lambda, B, \overline{A})$ or $(\overline{B}C\overline{A}) \in \delta(\lambda, B, \overline{A})$. In these steps $u' = u$ and $v' = v$. The content of the stack are changing in the following way: The used shadow symbol $\overline{A}$ (directly on B) may be deleted form the stack, a new non-shadow symbol C is put immediately above the top non-shadow symbol B, and finally B may be deleted or replaced by its shadow symbol.
  (*Remark*: The top non-shadow symbol and the shadow symbol immediately above are together used and the shadow-symbol may be deleted.)

So, the new configuration refers for the sentential form $u'(z')^{-1}$ in all these cases.

It is remained to show only that the order of the steps of the shadow-pushdown automata coincide with the left-most derivation. The automata can check only the top non-shadow symbol which represents the first (real) non-terminal letter of the sentential form, i.e., the left-most non-terminal leaf of the derivation graph. In each step this non-terminal will be processed. Therefore the theorem holds. □

An input letter is being read at exactly steps (a) and (b). According to the derivations, steps (b) and (c) lengthen the sentential form, and only step (c) increases the number of non-shadow symbols in the stack (i.e., the number of non-terminals in the sentential form). All the derivation steps are context-free, but those ones that refer for step (e). Step (d) refers for not context-sensitive derivation steps. Only steps (a) and (d) allow to empty the stack (i.e., allow to finish derivations).

Since the various types of grammars allow different types of rules, the various types of restricted shadow-pushdown automata correspond to different language classes. Based on the previous reasoning we have the following statement.

Corollary 1. *The class of shadow-pushdown automata with allowed steps of the forms (a), (c) and (e) accepts exactly the (λ-free) context-sensitive languages. The shadow-pushdown automata with transitions types of (a), (c), (d) and (e) are universal.*

We note here that a new iteration-free normal form for context-sensitive grammars can be found in [10] excluding the context-sensitive iterations of the Penttonen normal form, such as, $AB \to AC, AC \to AB$. This normal form can effectively be used in shadow-pushdown automata as well.

5 Conclusions

In this paper a special extension of stateless pushdown automata is presented. The shadow symbols are used in the stack representing the non-terminals from which the derivation is already continued, but they have shadow, i.e., they are needed as contexts in context-sensitive rules later. The stacks of these automata are used in the following way: it is assumed that the shadow symbols are half-translucent. One can access the top non-shadow symbol through them and it is also possible to check the shadow-symbol directly above the top non-shadow symbol. There are five different types of steps allowed, and the new class of automata is universal, i.e., every recursively enumerable language can be accepted. Restricting the work by forbidding some types of steps the regular, the context-free and the context-sensitive languages are characterized in automata theoretic way. The work of our shadow-pushdown automata is based on left-most constructions of the derivation graphs. Summarizing our results we present Table 1, in which the accepted language classes are shown for the shadow-pushdown automata depending on the type of allowed transitions. Besides the basic classes described earlier all the possibilities are given in Table 1. Obviously without

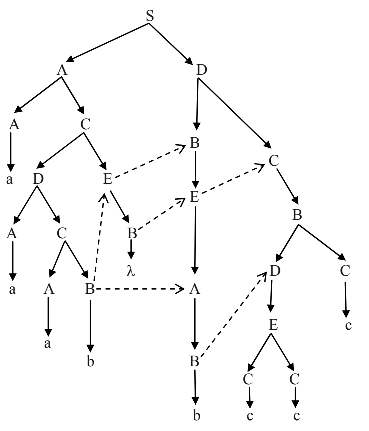


Fig. 1. Derivation-tree for a grammar in Penttonen normal form with context-edges

Table 1. Language classes accepted by shadow-pushdown automata defined by their possible movements

a	b	c	d	e	accepted language class
+	+	+	+	+	recursive enumerable
-	+	+	+	+	recursive enumerable
+	-	+	+	+	recursive enumerable (Penttonen normal form)
-	-	+	+	+	$\{\lambda\}$ or $\emptyset$
+	+	-	+	+	regular
-	+	-	+	+	regular
+		-	+	+	$L \subseteq T \cup \{\lambda\}$ only words with length maximum 1
-	-	-	+	+	$\{\lambda\}$ or $\emptyset$
+	+	+	-	+	λ-free context-sensitive
-	+	+	-	+	$\emptyset$
+	-	+	-	+	λ-free context-sensitive (Penttonen normal form)
-	-	+	-	+	$\emptyset$
+	+	-	-	+	λ-free regular
-	+	-	-	+	$\emptyset$
+	-	-	-	+	$L \subseteq T$ only words having length 1
-	-	-	-	+	$\emptyset$
+	+	+	+	-	context-free
-	+	+	+	-	context-free
+	-	+	+	-	context-free
-	-	+	+	-	$\{\lambda\}$ or $\emptyset$
+	+	-	+	-	regular
-	+	-	+	-	regular
+	-	-	+	-	$L \subseteq T \cup \{\lambda\}$
-	-	-	+	-	$\{\lambda\}$ or $\emptyset$
+	+	+	-	-	λ-free context-free
-	+	+	-	-	$\emptyset$
+	-	+	-	-	λ-free context-free (Chomsky normal form)
-	-	+	-	-	$\emptyset$
+	+	-	-	-	λ-free regular
-	+	-	-	-	$\emptyset$
+	-	-	-	-	$L \subseteq T$
-	-	-	-	-	$\emptyset$

transitions type (d) only λ-free languages can be generated. With steps type (a) and (b) all (λ-free) regular language can be accepted. Using steps type (c) instead of type (b) the (λ-free) context-free languages are accepted. Allowing rules type (e) the class of (λ-free) context-sensitive languages is accepted. Additional steps type (d) give computational universality to the automata. Automata without steps (a) and (d) cannot make empty the stack, while automata without steps (a) and (b) cannot read the input word, so they cannot accept any words containing letters. Our formalism is based on derivations in generative grammars, rather than the work of Turing-machines and other equivalent devices. Our construction is also useful from the didactic point of view.

Moreover one-turn automata (concerning the number of non-shadow symbols in the stack) without transitions type (e) accept the linear languages. Having arbitrary, but bounded stack the pushdown automata cannot go beyond regular. There are some interesting open problems. What is the language class accepted by one-turn shadow-pushdown automata allowing transitions type (e)? How can deterministic version be defined? The complexity analysis of the work of the shadow-pushdown automata (based on the normal form presented in [10]), the stack-languages of accepted runs are also subjects of future work.

Acknowledgements

This work is supported by National Office for Research and Technology (NKTH), Hungary.

References

1. Černý, A.: Description of words by cellular automata. Kuwait Journal of Science & Engineering 24, 199–215 (1997)
2. Geffert, V., Mereghetti, C., Palano, B.: More Concise Representation of Regular Languages by Automata and Regular Expressions. In: Ito, M., Toyama, M. (eds.) DLT 2008. LNCS, vol. 5257, pp. 359–370. Springer, Heidelberg (2008)
3. Gray, J.N., Harrison, M.A., Ibarra, O.H.: Two-way pushdown automata. Information and Control 11, 30–70 (1967)
4. Hopcroft, J.E., Ullmann, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading (1979)
5. Latteux, M., Simplot, D.: Context-sensitive string languages and recognizable picture languages. Information and Computation 138, 160–169 (1997)
6. Maslov, A.N.: Multilevel stack automata. Prob. Inf. Transmiss. 12, 38–42 (1976)
7. Meduna, A.: Deep pushdown automata. Acta Informatica 42, 541–552 (2006)
8. Morvan, C., Rispal, C.: Families of automata characterizing context-sensitive languages. Acta Informatica 41, 293–314 (2005)
9. Nagy, B.: Derivation trees for context-sensitive grammars. In: Ito, M., Kobayashi, Y., Shoji, K. (eds.) International Workshop on Automata, Formal Languages and Algebraic Systems (AFLAS 2008), Kyoto, Japan. World Scientific, Singapore (to appear, 2010)
10. Nagy, B., Varga, P.: A New Normal Form for Context-Sensitive Grammars. In: SOFSEM 2009, vol. II, pp. 60–71 (2009)
11. Penttonen, M.: One-sided and two-sided context in formal grammars. Information and Control 25, 371–392 (1974)
12. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages. Springer, Heidelberg (1997)

Maximum Independent Set in Graphs of Average Degree at Most Three in $\mathcal{O}(1.08537^n)$

Nicolas Bourgeois[1], Bruno Escoffier[1],
Vangelis Th. Paschos[1], and Johan M.M. van Rooij[2]

[1] LAMSADE, CNRS FRE 3234 and Université Paris-Dauphine, France*
{bourgeois,escoffier,paschos}@lamsade.dauphine.fr
[2] Department of Information and Computing Sciences, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
jmmrooij@cs.uu.nl

Abstract. We show that Maximum Independent Set on connected graphs of average degree at most three can be solved in $\mathcal{O}(1.08537^n)$ time and linear space. This improves previous results on graphs of maximum degree three, as our connectivity requirement only functions to ensure that each connected component has average degree at most three.

We link this result to exact algorithms of Maximum Independent Set on general graphs. Also, we obtain a faster parameterised algorithm for Vertex Cover restricted to graphs of maximum degree three running in time $\mathcal{O}^*(1.1781^k)$.

1 Introduction

Maximum Independent Set is one of the most intensively studied problems in the field of exact exponential time algorithms. This started in the seventies when Tarjan and Trojanowski [18] gave an algorithm solving the problem in $\mathcal{O}(1.2600^n)$ time on general graphs. This result has been improved often since.

It is well known that Maximum Independent Set is linear time solvable on graphs of maximum degree two. For graphs of maximum degree three, Johnson and Szegedy [12] have shown that subexponential time algorithms for Maximum Independent Set exist if and only if they exist for the problem on general graphs. Therefore, Maximum Independent Set on graphs of maximum degree three most likely requires exponential time to solve, as under the *Exponential Time Hypothesis* (ETH) [11] the general problem requires exponential time. Thus, on graphs of maximum degree d, the problem first becomes exponentially hard when $d = 3$: this makes it an interesting problem on its own.

The current state of the art algorithms for Maximum Independent Set on general graphs often rely on the use of non-standard measures of the instance size to prove their running times [8]. These measures allow the generally faster results on Maximum Independent Set restricted to sparse graphs to be used to

* Research supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010.

© Springer-Verlag Berlin Heidelberg 2010

Table 1. Previous result on INDEPENDENT SET on graphs of maximum degree three

Authors		Running time
J. Chen, I. Kanj, W. Jia	[5]	$\mathcal{O}(1.1740^n)$
R. Beigel	[1]	$\mathcal{O}(1.1259^n)$
J. Chen, L. Liu, W. Jia	[7]	$\mathcal{O}(1.1504^n)$
J. Chen, I. Kanj, G. Xia	[6]	$\mathcal{O}(1.1255^n)$
F. V. Fomin and K. Høie	[9]	$\mathcal{O}(1.1225^n)$
A. Kojevnikov and A. Kulikov	[14]	$\mathcal{O}(1.1225^n)$
M. Fürer[1]	[10]	$\mathcal{O}(1.1120^n)$
I. Razgon	[15]	$\mathcal{O}(1.1034^n)$
N. Bourgeois, B. Escoffier, V. Th. Paschos[1]	[2]	$\mathcal{O}(1.0977^n)$
M. Xiao	[19]	$\mathcal{O}(1.0919^n)$
I. Razgon	[16]	$\mathcal{O}(1.0892^n)$
N. Bourgeois, B. Escoffier, V. Th. Paschos, J. M. M. van Rooij[1]		$\mathcal{O}(1.0854^n)$

[1] These results are on the broader class of graphs where each connected component has average degree at most three.

prove faster running times on general graphs. In this way, Fomin et al. obtained an $\mathcal{O}(1.2203^n)$ time algorithm [8], Kneis et al. an $\mathcal{O}(1.2132^n)$ time algorithm [13], and the authors of this paper obtained an $\mathcal{O}(1.2125^n)$ time algorithm [3]. This makes the problem important for algorithmic results on general graphs as improvements on graphs of maximum degree three directly relate to these results. We should mention that the above running times are dominated by Robson who gives an $\mathcal{O}(1.1889^n)$ algorithm in his unpublished technical report [17].

In this paper, we give an $\mathcal{O}(1.08537^n)$ time algorithm for MAXIMUM INDEPENDENT SET on connected graphs of average degree at most three. As separate connected components can be solved independently, this directly improves previous algorithms on graphs of maximum degree three while being applicable to a much larger class of graphs; our connectivity requirement only exists to ensure that the average degree of each connected component is at most three. This improves the results from a long line of papers as can be seen in Table 1.

Theorem 1. *There exists an algorithm solving* MAXIMUM INDEPENDENT SET *on connected graphs of average degree at most three in* $\mathcal{O}(1.08537^n)$ *time and linear space.*

We have used this result to obtain a faster algorithm for MAXIMUM INDEPENDENT SET on general graphs. Using our own *bottom up method* based on the average degree of a graph [3] together with *measure and conquer*, we obtained the following corollary.

Corollary 1 ([3]). *There exists an algorithm solving* MAXIMUM INDEPENDENT SET *on general graphs in* $\mathcal{O}(1.2125^n)$ *time and polynomial space.*

The proof is beyond the scope of this paper and can be found in [3,4]. We note that the improvement over the very recent paper of Kneis et al. [13] is small, and wonder whether both approaches can be combined.

A third reason why the Maximum Independent Set problem on maximum degree three graphs is interesting comes from the field of parameterised complexity. In this field, the Minimum Vertex Cover problem parameterised by the size of the vertex cover is a benchmark problem. On graphs of maximum degree three, we obtain the fastest known parameterised algorithm.

Corollary 2. *There exists an algorithm solving* Minimum Vertex Cover *on graphs of maximum degree three parameterised by the size of the vertex cover k in $\mathcal{O}^*(1.1781^k)$ time and polynomial space.*

Proof. This problem admits a kernel of size $2k$ [6] allowing us to transform the problem in polynomial time into an equivalent instance G on $n \leq 2k$ vertices. Because the complement of a vertex cover is an independent set, we can use Theorem 1 to test whether G has an independent set of size at least $n - k$ which is equivalent to G having a vertex cover of size at most k. This is done in $\mathcal{O}^*(1.08537^{2k}) < \mathcal{O}^*(1.1781^k)$ time and polynomial space.

This improves the previous fastest $\mathcal{O}^*(1.1864^k)$ algorithm of Razgon [16].

Notation and definitions. We assume the reader to be familiar with standard graph notation and terminology. To avoid confusion, let $N(v)$, $N[v]$ be the open and closed neighbourhoods of a vertex v, respectively. I.e., $N[v] = N(v) \cup \{v\}$.

An *independent set* I in G is a subset $I \subseteq V$ such that no two vertices in I are adjacent. A *maximum independent set* is an independent set of maximum size. A *vertex cover* C in G is a subset $C \subseteq V$ such that every edge of G has at least one endpoint in C. It is easy to see that I is an independent set if and only if $V \setminus I$ is a vertex cover.

In this paper, $G = (V, E)$ is a graph on n vertices and m edges, and I is the independent set the algorithm is constructing.

2 Overview of the Algorithm

We propose an $\mathcal{O}(1.08537^n)$ branch and reduce algorithm for Maximum Independent Set on connected graphs of average degree at most three.

Our algorithm has the following form. First it applies a series of well known reduction rules that simplify the instance. Secondly, it looks for vertex separators of size one or two in the graph and uses these to further simplify the instance. Thirdly, it exploits any separators that consist of the closed neighbourhood of a single degree three vertex that separate a tree from the rest of the graph. Finally, the algorithm looks for a suitable structure in the graph and branches on it: it generates a series of subproblems that are solved recursively and from which the largest solution is returned.

Similar to [2,10], we use the number of edges m minus the number of vertices n as a measure of progress in the analysis of the algorithm. The resulting upper bound on the running time of $\mathcal{O}(\alpha^{m-n})$ implies an $\mathcal{O}(\alpha^{0.5n})$ algorithm on graphs in which each connected component has average degree at most three.

The requirement on the average degree of each connected component exist because trees have measure $m - n = -1$; these trees are removed by the reduction rules and can increase $m - n$ to over $0.5n$ for the remaining graph.

2.1 Simple Reduction Rules and Small Separators

Our algorithm applies the following well known reduction rules. These are thoroughly described in many publications, for example [2,8,10], and require little explanation.

- *Degree 0, 1:* If G contains a vertex of degree 0 or 1, then take it in I and remove any neighbours.
- *Connected Components:* If G is disconnected, solve each connected component separately.
- *Domination:* If for two vertices u, v: $N[u] \subseteq N[v]$, then u *dominates* v and we remove v.

The domination rule is correct because in any maximum independent set containing v, v can be replaced by u. Notice that domination forces degree two vertices with adjacent neighbours in I.

- *Degree 2:* If there exists a vertex v of degree two with non-adjacent neighbours u, w, the algorithm removes v, merges u and w to a single vertex, and adds one to the size of I.

The degree two rule is also called *vertex folding.* Its correctness is based on the fact that if v is not in I, then we can put both neighbours in I because taking v gives an alternative of the same size to taking only one neighbour of v.

If these reduction rules do not apply, the graph is of minimum degree three. The algorithm then follows the approach of Fürer [10] and looks for vertex separators of size at most two.

- *Small Separators:* If the graph contains a vertex separator of size one or two, then recursively solve the smallest component and adjust the rest of the graph to the computed solution.

The proof can be found in [10].

Finally, if G is of maximum degree four, the algorithm looks at local configurations in which the closed neighbourhood of a vertex separates a tree from the rest of the graph.

- *Tree Separators:* If in a maximum degree four graph the closed neighbourhood of a single degree three vertex v separates a tree from the rest of the graph, then we replace this local configuration with a smaller equivalent one.

This rule is a small improvement of [2, Section 4]; its details are given in Section 3.

Before considering the branching of our algorithm, we look at the effect of the reduction rules to the $m - n$ measure. The measure is invariant under the degree one and two rules: they remove as many edges as vertices. The degree zero rule increases the measure, but we always treat this (one vertex) tree separately. And, after application of these rules, the domination and small/tree separator rules decrease $m - n$ by at least two.

2.2 The Branching Rules of the Algorithm

Let us start by giving the main idea as to why our algorithm is faster than previous publications. On maximum degree three graphs, most previous branching algorithms first branch on vertices that are in a cycle of length three or four as one easily observes that this leads to a smaller number of generated subproblems. Thereafter, these algorithms give a long subcase analyses to prove that efficiently enough branchings exist for graphs without such cycles.

We use these cycles not only to restrict the worst case to small cycle free graphs, but to improve the branching on these graphs as well. As we reduce low degree vertices, the only maximum degree three graphs that our algorithm considers are 3-regular. If no small cycles exists and we are forced to perform a relatively inefficient branching on a vertex v, then vertices near v get degree two in the subproblems generated. These vertices are folded by the degree two rule, resulting in new vertices of degree at least four. If new small cycles emerge by this folding, they must contain the new higher degree vertices: this combination allows for very good branching counterbalancing the inefficient branching we started with. And, if no new small cycles emerge, then the new higher degree vertices are spread out in the graph enough to allow us to use them for branching in a way that counterbalance the initial inefficient branching even more.

If no reduction rule applies, our algorithm branches producing several subproblems that are solved recursively; the maximum independent set in the graph is the maximum over the results of the recursive calls. The description of the branching of our algorithm is divided over the proofs of four lemmas.

The proofs of the four lemmas are based on an extensive subcase analysis. For each case, we thoroughly analyse the local structures involved to removing as many vertices and edges as possible. Due to space restrictions, we will not give full proofs of all four lemmas in this paper. We will prove the two simpler Lemmas 1 and 3 and give large parts of the proofs of Lemma 4 in Section 4. For the full proofs, see [4]. In the analysis, we let $T(k)$ be the number of subproblems generated when branching on a graph G of complexity $m - n = k$.

We start with non 3-regular graphs, with extra attention to small cycles if the maximum degree is four.

Lemma 1. *If G has a vertex of degree at least 5, then $T(k) \leq T(k-4)+T(k-7)$.*

Lemma 2. *If G is of maximum degree 4 with a vertex of degree 4, then either:*

1. *G has a vertex of degree 4 that is part of a 3- or 4-cycle also containing at least one degree 3 vertex, and there are no 3- or 4-cycles containing only degree 3 vertices, then: $T(k) \leq T(k-5) + T(k-6)$ or $T(k) \leq 2T(k-8) + 2T(k-12)$.*
2. *or, G has a vertex of degree 4 that is part of 3- or 4-cycle also containing at least one degree 3 vertex, and the constraint on the degree 3 vertices in item 1 does not hold, then: $T(k) \leq T(k-4) + T(k-6)$ or $T(k) \leq 2T(k-8) + 2T(k-12)$.*
3. *or, the previous does not apply and then $T(k) \leq T(k-3) + T(k-7)$.*

If G is 3-regular and contains 3- or 4-cyles, we branch efficiently.

Lemma 3. *If G is 3-regular and contains a 3- or 4-cycle, then $T(k) \leq T(k-4) + T(k-5)$.*

And, in the remaining case, we use Lemma 2 to counterbalance the less efficient branching. This works in the following way. In the general case, we branch generating two subproblems: one to which we can apply Lemma 2 case 1 and one where we can apply Lemma 2 case 3. The rest of the proof of this lemma eliminates all cases to which this general case does not apply.

Lemma 4. *If G is 3-regular and 3- and 4-cycle free, then $T(k) \leq T_1(k-2) + T_3(k-5)$, or a better sequence of branchings exist. Here, we apply cases 1 and 3 from Lemma 2 to the branches denoted by T_1 and T_3, respectively.*

We proof Theorem 1 by taking all four Lemmas together. The worst case of all branchings from Lemmas 1 up to 4 is $T(k) \leq T(k-8)+2T(k-10)+T(k-12)+2T(k-14)$. This recurrence relation is formed by combining Lemmas 4 and 2 and leads to a running time of $\mathcal{O}(1.17802^k)$. On average degree three graphs this is $\mathcal{O}(1.17802^{0.5n}) = \mathcal{O}(1.08537^n)$.

3 The Tree Separators Rule

In this section, we give the details on our tree separators rule. This is the only new reduction rule introduced in Section 2.1. We stated it in the following way:

- *Tree Separators:* If in a maximum degree four graph the closed neighbourhood of a single degree three vertex v separates a tree from the rest of the graph, then we replace this local configuration with a smaller equivalent one.

We will now prove that such a rule exists.

Lemma 5. *We can replace any local neighbourhood satisfying the requirements of the tree separators rule by an equivalent one. This operation reduces the $m-n$ measure by at least two.*

Proof. Let v be a degree three vertex as described in the rule with neighbours a, b and c. Notice that a, b and c all have at least one edge not incident to v or the tree T: otherwise a vertex separator of size at most two would exist and we can apply the small separators rule. Because of this and the fact that G is of maximum degree four, there are at least three and at most six edges between $N(v)$ and T, and there exists at most one edge in $G[N(v)]$.

First consider the case where there exists an edge in $G[N(v)]$, hence $|T| \leq 2$. In this case, the maximum independent set in $G[N[v] \cup T]$ is of size two, therefore we can safely pick those vertices in I that pose no restrictions on the remaining graph: v and one vertex from T. This is easy to see if T is a single vertex, namely taking v or T in I forbids taking a, b and c, and because of the edge in $N(v)$ it is not possible to take all tree vertices of $N(v)$. If $|T| = 2$ and without loss

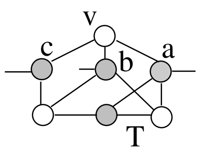


Fig. 1. T consists of three vertices and at least two neighbours of v, here a and b, have two tree neighbours

of generality a is connected to both vertices in T, then we cannot take three vertices if we take v because this forbids taking a, b and c while there is an edge in T. Also, we cannot take three vertices if we take a because this forbids taking v and T while there is an edge between b and c. The remaining vertices form a four cycle, and thus I has at most two vertices from $N(v) \cup T$.

Secondly, if there is no edge in $N(v)$ and $|T| \leq 2$, then we merge $N(v) \cup T$ to a single vertex and add two to the size of I. This is similar to vertex folding. Namely, the only maximum independent set in $G[N[v] \cup T]$ equals $N(v)$, while if we do not take these three vertices we can safely pick the non-restricting size two option consisting of v and one vertex from T. Merging the local structure to a single vertex postpones this choice: taking the merged vertex corresponds to taking the three vertex option, while discarding the merged vertex corresponds to taking the two vertex option with no vertices in $N(v)$. This is clearly correct if $|T| = 1$ since taking v or T in I forbids taking any vertex in $N(v)$. And, if $|T| = 2$, then taking v in I again forbids taking any vertex in $N(v)$ and taking any vertex of T in I forbids taking anything except for v and one of its neighbours, again not allowing three vertices to be picked.

Finally, if $|T| \geq 3$, then we can safely pick v and a maximum independent set in $G[T]$. Let us first look at the case where $|T| = 3$; see Fig. 1. At least two neighbours of v, say a and b have two neighbours in T. If we would have taken a in I, then we would forbid all vertices in $N[v] \cup T$ except for b, c and one vertex in T. By adjacencies, we can take only two of these vertices making our initial choice a safe alternative. The same argument goes for taking b in I. Thus, with a and b discarded, we can safely pick the degree one vertex v and hence also the maximum independent set in $G[T]$ to be in I. What remains is the case $|T| = 4$. If T has three leaves, then there is only one way to pick four vertices in I: v and the three leaves of T. This does not restrict any choices in the rest of the graph and hence is optimal. Otherwise, T is a four vertex path and all local configurations of $G[N[v] \cup T]$ have a maximum independent set of size three. Again, picking v and a maximum independent set in $G[T]$ is a safe choice. $\square$

4 Proof of Lemmas 1-4

What remains is to prove Lemmas 1-4. Due to space restrictions of this conference version of our work, we cannot give a full proof of all four lemmas here. The missing proofs can be found in [4]. Lemmas 1 and 3 allow short proofs; these will be given below. These proofs also function to give an idea of the proof of

Lemma 2 as this proof uses similar ideas although involving a much more extensive case analysis. Finally, we give parts of the proof of Lemma 4, also omitting a large portion of the case analysis.

We will need the concept of a *mirror* (see for example [8]). A vertex $m \in V$ is a mirror of $v \in V$ if $G[N(v) \setminus N(m)]$ is a clique, or equivalently, every combination of two non-adjacent vertices in $N(v)$ contains a vertex from $N(m)$. Whenever the algorithm branches on v and discards it, it can discard all its mirrors as well. This is because at least two neighbours of v should be in I in this branch because taking v is an alternative of equal size to taking only one.

Using the $m-n$ measure, we have to be careful when trees are separated from G since they have complexity -1. This will not happen in branches where at most two vertices are removed because of the small separator rule. The same goes for branches where the neighbourhood of a single degree three vertex (sometimes through domination) is removed in a maximum degree four graph because of the tree separators rule. When trees can be separated, we often bound this by counting the number of *external edges*. When considering to remove a set of vertices $S \subset V$ from G, these are the edges connecting S to the rest of G.

We often use counting arguments involving the external edges. If no trees are separated, each external edge reduces the $m-n$ measure by one as it is removed, while for each tree that is separated, this measure increases again by 1. Notice that at most one tree can be separated per three external edges.

4.1 Proof of Lemma 1

Lemma 1. *If G has a vertex of degree at least 5, then $T(k) \leq T(k-4)+T(k-7)$.*

Proof. Consider such a vertex v as in Fig. 2 and assume that we branch on it. If v is discarded, one vertex is removed and at least five edges are removed giving $T(k-4)$. If v is taken in I, $N[v]$ is removed. All vertices in $N(v)$ have at least one neighbour outside of $N[v]$ by domination. In the worst case, $N(v)$ consists of degree three vertices, hence there are at most two edges in $G[N(v)]$ and at least six external edges. If no trees are created, this sums to 13 edges and 6 vertices giving $T(k-7)$. And, if there are more external edges because either a vertex in $N(v)$ has degree four or more, or there are fewer edges in $G[N(v)]$, then these extra external edges compensate for the possible trees that can be separated.

What remains is to handle the special case in Fig. 2 where the minimum amount of 13 edges is removed and a separate tree is created. This tree will be a single degree three vertex t since otherwise there exists a two separator in $N(v)$.

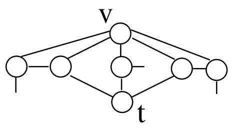


Fig. 2. A vertex of degree five whose removal removes 13 edges and separates a tree

Notice that v is a mirror of t. We branch on t. Taking t in I leads to the removal of 4 vertices and 9 edges: $T(k-5)$. And, discarding t and v leads to the removal of 8 edges and 2 vertices: $T(k-6)$. This branching with $T(k) \leq T(k-5)+T(k-6)$ is better than the required $T(k) \leq T(k-4) + T(k-7)$. □

4.2 Proof of Lemma 3

Lemma 3. *If G is 3-regular and contains a 3- or 4-cycle, then $T(k) \leq T(k-4) + T(k-5)$.*

Proof. Let a, b, c form a 3-cycle in G. When one of these vertices, say a, has a third neighbour v that is not part of a 3-cycle, then we branch on v. If v is taken in I, then 9 edges and 4 vertices are removed: $T(k-5)$. If v is discarded and by domination a is taken in I, then 8 edges and 4 vertices are removed: $T(k-4)$.

This covers the 3-cycles, unless all third neighbours of a, b and c are in a 3-cycle also. Moreover, they are in different 3-cycles because otherwise there would exist a size two vertex separator. We branch on a. In the branch where a is discarded, domination results its third neighbour to be taken in I giving $T(k-4)$ as before. In the other branch, a is taken in I, and by domination the third neighbours of b and c are taken in I. This removes their corresponding 3-cycles completely: at least 16 edges and 10 vertices are removed. We notice that a tree can be separated from G, but then we still have $T(k-5)$ or better.

Finally, suppose that G is 3-cycle free and let v be a vertex on a 4-cycle. Any vertex opposite to c on a 4-cycle is a mirror of v. We branch on v. In one branch, we take v in I and 3-cycle freeness results in the removal of 9 edges and 4 vertices: $T(k-5)$. In the other branch, we discard v and all its mirrors. This results in the removal of 6 edges and 2 vertices if v has only one mirror and possibly more if v has two or three mirrors: $T(k-4)$. Again trees can be separated from G, but then v must have three mirrors. There is only one local configuration representing this case in which 12 edges and 7 vertices are removed: this is more than enough. □

4.3 Large Parts of the Proof of Lemma 4

Lemma 4. *If G is 3-regular and 3- and 4-cycle free, then $T(k) \leq T_1(k-2) + T_3(k-5)$, or a better sequence of branchings exist. Here, we apply cases 1 and 3 from Lemma 2 to the branches denoted by T_1 and T_3, respectively.*

Proof. Our reduction rules guarantee that no trees can be separated from G since we branch on a degree three vertex in a maximum degree three graph. We can also assume that no other reduction rules than the degree one and two rules can be applied after branching. Namely, if such a reduction rule fires in the branch where we discard v, then we have the sufficient relation of $T(k) \leq T(k-4) + T(k-5)$. And, if such a rule fires only in the branch where we take v in I, then we follow the proof below and obtain $T(k) \leq T_1(k-2) + T(k-7)$.

Let v be a vertex of G with neighbours x, y and z. We systematically consider the possible local neighbourhoods around v and observe what happens to this local neighbourhood when we branch on v. Because of 3- and 4-cycle freeness, v is the only common neighbour of any two vertices from the set $\{x, y, z\}$. By the same argument, there cannot be any adjacent vertices within $N(x)$, $N(y)$ or $N(z)$. There can, however, be at most six adjacencies between vertices from two different neighbourhoods. These adjacencies are important in the branch where v is discarded. Here, the remaining neighbours of x, y and z are folded to single vertices, and hence these adjacencies determine the newly formed local structure on which we will branch next. Notice that when there are two adjacencies between the same neighbourhoods, then the vertex folding after discarding v will lead to the removal of an extra edge since we do not allow double edges.

We begin by showing that we can easily deal with cases involving more than three of these adjacencies between the neighbourhoods of x, y and z. If there are six, then we are looking at a connected component of constant size that can be solved in constant time. If there are five, then there are only two external edges out of $N[x] \cup N[y] \cup N[z]$ and hence there exists a small separator. Finally, we consider the case when there are four such adjacencies. These four adjacencies cannot all be between the same two neighbourhoods as this creates a 4-cycle. If these adjacencies form two pairs of adjacencies between the three neighbourhoods, then vertex folding will remove two additional edges giving $T(k) \leq T(k-4) + T(k-5)$. What remains is that all three neighbourhoods are adjacent. In the branch where we discard v, the neighbourhoods of x, y and z are folded resulting in two degree three vertices between which a double edge is removed that are in a 3-cycle with a degree four vertex. This allows us to apply Lemma 2 case 2 to obtain the following recurrence relations with a better branching behaviour than we are proving: $T(k) \leq T(k-5) + T(k-3-4) + T(k-3-6)$ or $T(k) \leq T(k-5) + 2T(k-3-8) + 2T(k-3-12)$. Note that the term $T(k-5)$ comes from the standard branch when taking v in I.

We will now show that we can always obtain a $T_3(k-5)$ branch when taking v in I and discarding its neighbours. Removing $N(v)$ results in the creation of six degree two vertices that will be folded. If any of these vertices are folded to degree four vertices, we can apply Lemma 2 and we are done. Consider the case in which no degree four vertices are created. In this case, the degree two vertices must form a set of chains of even length. By the previous argument, we know that there are at most three adjacencies between the vertices in $(N(x) \cup N(y) \cup N(z)) \setminus \{v\}$. These vertices are the new degree two vertices, and therefore the only possible way for them to be divided in even length chains is when they form three pairs of adjacent vertices. In this particular local structure, v lies on three 5-cycles, each pair of which overlaps in v and a different neighbour of v. We obtain the required branching behaviour by deciding not to branch on v: either this connected graph G has a vertex with a different local configuration, or G has no such vertex and it must equal the dodecahedron. We finish the argument by noting that the dodecahedron has 20 vertices and can be solved in constant time.

What remains is the $T_1(k-2)$ branch when discarding v. In this branch, vertex folding results in three folded vertices. Notice that these folded vertices are of degree four unless folding results in the implicit removal of double edges.

Because the graph is 3- and 4-cycle free before applying this lemma, a new 3- or 4-cycle created by folding after discarding v must involve the folded vertices.

If all folded vertices are of degree four and such a 3- or 4-cycle that contains a degree three vertex is created, we can apply Lemma 2 case 1. If, additional edges are removed and we also consider the possibility that 3- or 4- cycles involving only degree three vertices are created, we can apply the slightly worse case 2 of Lemma 2 on the graph of complexity $k-3$; this results in $T(k) \leq T_3(k-5) + T(k-3-4) + T(k-3-6)$ or $T(k) \leq T_3(k-5) + 2T(k-3-8) + 2T(k-3-12)$.

The only cases that remain are those in which no new 3- or 4-cycles are create by folding, or in which a 3-cycle is created consisting of folded degree four vertices only. We consider six different cases depending on the relative location of the vertices that are the result of folding in the graph obtained after discarding v. These three vertices will be of degree four unless there are double adjacencies between the neighbourhoods $N(x)$, $N(y)$, and $N(z)$: in these case folding results in double edges that will be removed. Hence, the following six cases arise:

1. Three non-adjacent degree four vertices.
2. Three degree four vertices only two of which are adjacent.
3. Three degree four vertices on a path of length three.
4. Two adjacent degree three vertices and a non-adjacent degree four vertex.
5. Two degree three vertices adjacent to a degree four vertex.
6. Three degree four vertices that form a 3-cycle.

For each of these six cases, we give efficient sequences of branchings in our technical report [4]. These are based on the following reasoning that is quite similar to exploiting mirrors. Namely, if v is discarded, we know that we need to pick at least two of the three neighbours of v: if we pick only one we could equally well have taken v which is done in the other branch already. This observation becomes slightly more complicated because we just folded the neighbours of v. The original vertex $x \in N(v)$ is taken in the independent set if and only if the vertex x' that is created by folding $N(x)$ is discarded in the reduced graph. Thus, the fact that we needed to pick at least two vertices from $N(v)$ results in us being allowed to pick at most one vertex from the three degree four vertices created by folding the neighbours of v. Therefore, picking any vertex from the three folded vertices allows us to discard the other two. □

Acknowledgements. The authors thank the anonymous referees for useful suggestions and corrections.

References

1. Beigel, R.: Finding maximum independent sets in sparse and general graphs. In: Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1999, pp. 856–857 (1999)
2. Bourgeois, N., Escoffier, B., Paschos, V.Th.: An O$^*(1.0977^n)$ exact algorithm for max independent set in sparse graphs. In: Grohe, M., Niedermeier, R. (eds.) IWPEC 2008. LNCS, vol. 5018, pp. 55–65. Springer, Heidelberg (2008)

3. Bourgeois, N., Escoffier, B., Paschos, V.Th., van Rooij, J.M.M.: A bottom-up method and fast algorithms for max independent set. In: Proceedings of the 12th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2010 (to appear, 2010)
4. Bourgeois, N., Escoffier, B., Paschos, V.Th., van Rooij, J.M.M.: Fast algorithms for max independent set in graphs of small average degree. arXiv.org; arXiv:0901.1563v1 [cs.DM] (2009)
5. Chen, J., Kanj, I.A., Jia, W.: Vertex cover: Further observations and further improvements. Journal of Algorithms 41(2), 280–301 (2001)
6. Chen, J., Kanj, I.A., Xia, G.: Labeled search trees and amortized analysis: Improved upper bounds for NP-hard problems. Algorithmica 43(4), 245–273 (2005)
7. Chen, J., Liu, L., Jia, W.: Improvement on vertex cover for low-degree graphs. Networks 35(4), 253–259 (2000)
8. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. Journal of the ACM 56(5) (2009)
9. Fomin, F.V., Høie, K.: Pathwidth of cubic graphs and exact algorithms. Information Processing Letters 97, 191–196 (2006)
10. Fürer, M.: A faster algorithm for finding maximum independent sets in sparse graphs. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 491–501. Springer, Heidelberg (2006)
11. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? Journal of Computer and System Sciences 63(4), 512–530 (2001)
12. Johnson, D.S., Szegedy, M.: What are the least tractable instances of max independent set? In: Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1999, pp. 927–928 (1999)
13. Kneis, J., Langer, A., Rossmanith, P.: A fine-grained analysis of a simple independent set algorithm. In: IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009. LIPIcs, vol. 4, pp. 287–298. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2009)
14. Kojevnikov, A., Kulikov, A.S.: A new approach to proving upper bounds for max-2-sat. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, pp. 11–17. ACM Press, New York (2006)
15. Razgon, I.: A faster solving of the maximum independent set problem for graphs with maximal degree 3. In: Algorithms and Complexity in Durham 2006 - Proceedings of the Second ACiD Workshop. Texts in Algorithmics, vol. 7, pp. 131–142. King's College, London (2006)
16. Razgon, I.: Faster computation of maximum independent set and parameterized vertex cover for graphs with maximum degree 3. Journal of Discrete Algorithms 7(2), 191–212 (2009)
17. Robson, J.M.: Finding a maximum independent set in time $O(2^{n/4})$. Technical report, LaBRI, Université Bordeaux I (2001)
18. Tarjan, R.E., Trojanowski, A.: Finding a maximum independent set. SIAM Journal on Computing 6, 537–546 (1977)
19. Xiao, M.: New branching rules: Improvements on independent set and vertex cover in sparse graphs. arXiv.org; arXiv:0904.2712v1 [cs.DS] (2009)

Simultaneity in Event Structures*

G. Michele Pinna and Andrea Saba

Dipartimento di Matematica e Informatica, Università di Cagliari, Italy
gmpinna@unica.it, andrea@sc.unica.it

Abstract. Various brand of event structures, prime, bundle, flow, asymmetric, inhibitor just to mention some, have been proposed to face the various kinds of causality and conflict arising in computation. The notion of simultaneity, i.e., the faithful representation that certain events have to occur *together*, is usually left out from the models for concurrent computations, with some notably exceptions like Pratt's Chu spaces or Bruni&Montanari's Zero-Safe nets. In this paper we propose a notion of *event structures with simultaneity* to take into account the simultaneity and we relate the introduced notion with the prime event structures and domains.

1 Introduction

In concurrent and distributed computations the focus is usually posed on two basic phenomena: *causality* and *conflict*, the first one aiming at modeling the dependencies arising among various activity and the other at capturing the impossibility that certain activities are present in the same computation.

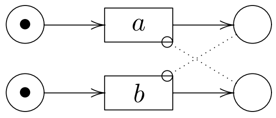
The classical way to model causality is using a partial order relation, which has been put forward in the 70s in the realm of Petri nets (e.g., Goltz and Reisig approach to non sequential processes [7]) and further consolidated by Nielsen, Plotkin and Winskel in [16] and [22]. In [16] and [22] it is also proposed to model conflicts using a symmetric and irreflexive relation on events which is inherited along the partial order modeling the causality. Since these seminal works, the investigations on the notions of causality and conflicts have revealed several aspects that are neither completely nor satisfactorily reflected by the these first suggestions, and many other proposals have been developed aiming at modeling causality and/or conflict in a more precise and accurate way, in order to be able to faithfully reflect situations arising in concurrent computations. Among many others we recall Hoogers, Kleijn and Thiagarajan ([10]) where the notion of causality is made dependent on the state; Baldan, Corradini and Montanari ([3]) where a notion of weak causality is introduced which is able to model causality, conflict and a weaker notion of conflict, namely asymmetric conflict; or flow event structure proposed by Boudol ([4]) where a flow relation among events is introduced to model causality, relation that is weaker than the partial

* Work partially supported by the *CGR Coarse Grain Recommendation*-project, PIA "Industria, Artigianato e Servizi", Regione Autonoma della Sardegna.

© Springer-Verlag Berlin Heidelberg 2010

order; or by the first author with Poigné ([18]) where a more operational model for causality and conflict is presented, and causality can be presented in a logical fashion; or Gunawardena's work on causal automata ([8,9]) and finally we recall the notion of inhibitor event structure ([2]) where a ternary relation is introduced which is able to model causality, conflict and a sort of "non monotone" enabling.

A situation which is never modeled by these approaches is what we can call *simultaneity*. Let us briefly describe this situation with some examples. Consider the novel computational paradigm of membrane computing introduced by Pãun in [17]. Here we have a system of nested membranes where to each membrane a multiset of objects is associated (the local state) and a set of rules transforming the local states. Rules may send multisets of objects in other membrane, provided that they are *close* (i.e., either directly contained in or directly containing), but the crucial point is that rules have to be applied in a maximal concurrent fashion, which account to say that they have to be regarded as *simultaneous* with respect to any observation. GCausality and conflict in membrane computing have received in recent years some attention, as these notions are central ones in models of concurrent and parallel computations, among the others Kleijn, Koutny and Rozenberg in [15] have proposed a non sequential semantics based on nets, and Busi in [6] studied the dependencies among rules occurrence in a more precise way, but none of these approaches takes into account the simultaneity of rule occurrences, which is moved to the level of the possible observations. Thus, simultaneity in these approaches can only be observed but never prescribed, which fails to capture one of the main feature of this paradigm. Another example regards the net with inhibitor arcs depicted here. According to some semantics, in this net only one of the two may happen but never both. However when the two transition would fire simultaneously, the step $\{a, b\}$ could be observed. In other words, if we could observe transitions synchronization (where the single transition still retains its identity), the observation $\{a, b\}$ is perfectly reasonable.



This situation can be observed using the *stratified order structures* of Janicki and Koutny ([13,11]) where two relations model together the synchronization of events. The two relations, *earlier than* and *not later than*, model causality (what must have happened) and inhibition, and configurations of stratified order structure allow to reflect the simultaneity of the execution. It if worth to remark that *synchronization*, despite it is often confined to the level of observation, has received attention also on the structural level. The synchronization of transitions in Petri nets have been studied and formalized by Bruni and Montanari ([5]). In their Zero-Safe Nets the synchronization of transitions is formalized with the aid of zero-safe places, this on the level of net theory rather than by means of suitable observations. This notion have been used to give another non sequential interpretation of membrane computing in [19], the idea being that rule occurrences happening together can be synchronized with the aid of zero safe places.

In this paper we begin a line of research aiming at modeling faithfully the *simultaneity* of the happening of certain events. As we want to prescribe simultaneity rather than putting some machinery to be able to infer simultaneity afterward, we extend the notion of prime event structure of [16,22] with a set of finite subsets of events that are simultaneous. The introduced notion is a conservative extension of the classical one, and to show that the model has computational ground, we prove that the configurations of this new brand of event structure is again a coherent, prime algebraic and finitary domain. When looking at the elements of the domain we abstract from the information about the simultaneity of events. On this level the happening of simultaneous events should be seen as a single activity. We will highlight the relationships with domains and prime event structures. Finally we want to stress that there are some other approaches where simultaneity can be prescribed: for example, CHU-spaces introduced by Pratt ([21,20]), where basically both states and events are represented *together* hence simultaneity can be described by *deleting* those states where some but not all of the simultaneous events appear; and the Relational Structures of Janicki ([12]) where a relation is introduced to be able to model, to some extent, simultaneity. Finally some research in a similar direction has been done in [14].

The paper is organized as follows: in the next section we recall the notions of prime event event structure and of domain and their relations, then, in section 3, we introduce the notion of event structure with simultaneity and we show that its configurations form a coherent, prime algebraic and finitary domain. In section 4 we study the notion of morphism for these event structures, showing that they can be turned out into a category. We show that prime event structures can be turned into a full subcategory of event structures with simultaneity, and we establish an adjunction between the category of event structures with simultaneity and the category of domains.

2 Prime Event Structures and Domains

In this section we recall some basic notions and results on prime event structures and domains, as developed in [16,22]. Given a set A, with $\mathbf{2}^A$ we denote the set of the subsets of A. The set of all finite sets over A is denoted by $\mathbf{2}^A_{\mathit{fin}}$.

Prime event structures (PES) [16] are a simple event-based model of concurrent computations in which events are considered as atomic and instantaneous steps, which can appear only once in a computation. The relationships between events are expressed by means of two relations: *causality* and *conflict.*

Definition 1. *A* prime event structure (PES) *is a tuple $P = \langle E, \leq, \# \rangle$, where E is a set of* events *and $\leq$, $\#$ are binary relations on E called* causality relation *and* conflict relation *respectively, such that: (1) the relation $\leq$ is a partial order and $\lfloor e \rfloor = \{e' \in E : e' \leq e\}$ is finite for all $e \in E$, and (2) the relation $\#$ is irreflexive, symmetric and hereditary with respect to $\leq$, i.e., $e\#e'$ and $e' \leq e''$ imply $e\#e''$ for all $e, e', e'' \in E$.*

An event can occur only after some other events (its causes) have taken place, and the execution of an event can prevent the execution of other events. This is formalised via the notion of *configuration* of a PES $P = \langle E, \leq, \# \rangle$, which is a subset of events $C \subseteq E$ such that for all $e, e' \in C$ $\neg(e \# e')$ (*conflict-freeness*) and $\lfloor e \rfloor \subseteq C$ (*left-closedness*). Given two configurations $C_1 \subseteq C_2$ if $e_0, \ldots, e_n$ is any linearisation of the events in $C_2 - C_1$, compatible with causality, then $C_1 \subseteq C_1 \cup \{e_0\} \subseteq C_1 \cup \{e_0, e_1\} \subseteq \ldots \subseteq C_2$ is a sequence of well-defined configurations. Therefore subset inclusion can be safely thought of as a computational ordering on configurations. The set of configurations of a prime event structure P, ordered by subset inclusion, is denoted by $\mathit{Conf}_{\mathrm{pes}}(P)$.

Given a PES $P = \langle E, \leq, \# \rangle$, with *co* we denote the relation on $E \times E$ defined as follows: $e \ co\ e' \ \Leftrightarrow\ e \not\leq e' \ \wedge\ e' \not\leq e \ \wedge\ \neg(e \#\ e')$, and with $\mathbf{co}(A)$ we indicate that either $e\ co\ e'$ $\forall e, e' \in A$ or A is a singleton.

A preordered or partially ordered set $\langle D, \sqsubseteq \rangle$ will be often denoted simply as D, by omitting the (pre)order relation. Given an element $x \in D$, we write $\downarrow x$ to denote the set $\{y \in D \mid y \sqsubseteq x\}$. Given a subset $X \subseteq D$, the *least upper bound* and *greatest lower bound* of X, when they exist, are denoted by $\bigsqcup X$ and $\bigsqcap X$, respectively. A subset $X \subseteq D$ is *compatible*, written $\uparrow X$, if there exists an upper bound $d \in D$ for X (i.e., $x \sqsubseteq d$ for all $x \in X$). It is *pairwise compatible* if $\uparrow \{x, y\}$ (often written $x \uparrow y$) for all $x, y \in X$. A subset $X \subseteq D$ is *directed* if any finite subset of X has an upper bound in X. The partial order D is *complete* (CPO) if any directed subset of X has a least upper bound in D. Let D be a CPO. Recall that an element $e \in D$ is *compact* if for any directed set $X \subseteq D$, $e \sqsubseteq \bigsqcup X$ implies $e \sqsubseteq x$ for some $x \in X$. The set of *compact* elements of D is denoted by $\mathsf{K}(D)$.

Definition 2. *A partial order D is called* coherent *(*pairwise complete*) if for all pairwise compatible $X \subseteq D$, there exists the least upper bound $\bigsqcup X$ of X in D. A* complete prime *of D is an element $p \in D$ such that, for any compatible $X \subseteq D$, if $p \sqsubseteq \bigsqcup X$ then $p \sqsubseteq x$ for some $x \in X$. The set of complete primes of D is denoted by $Pr(D)$. The partial order D is called* prime algebraic *if for any element $d \in D$ we have $d = (\bigsqcup \downarrow d \cap Pr(D))$. The set $\downarrow d \cap Pr(D)$ of complete primes of D below d will be denoted $Pr(d)$. We say that D is* finitary *if for each compact element $e \in \mathsf{K}(D)$ the set $\downarrow e$ is finite. Coherent, prime algebraic, finitary partial orders will be referred to as* (Winskel's) domains.

Being not expressible as the least upper bound of other elements, the complete primes of D can be seen as elementary indivisible pieces of information (events). Thus prime algebraicity expresses the fact that any element can be obtained by composing these elementary blocks of information.

Given a domain D and two distinct elements $d \neq d' \in D$ we say that d is an *immediate predecessor* of d', written $d \prec d'$ if $d \sqsubseteq d' \ \wedge \forall d'' \in D.\ (d \sqsubseteq d'' \sqsubseteq d' \ \Rightarrow\ d'' = d \ \vee\ d'' = d')$. Moreover we write $d \preceq d'$ if $d \prec d'$ or $d = d'$. According to the informal interpretation of domain elements sketched above, $d \prec d'$ intuitively means that d' is obtained from d by adding a quantum of information.

Both event structures and domains can be seen as models of systems where computations are built out from atomic pieces. Formalising this intuition, in [16] it is shown that domains and event structures are essentially the same.

Theorem 1. *Let P be a* PES*. Then $\mathcal{L}(P) = (Conf_{pes}(P), \subseteq)$ is a coherent, finitary and prime algebraic domain.*

Vice versa, to each domain a PES is associated whose configuration are the domain. Mapping domains back to PES's requires the introduction of the notion of prime interval. Let $\langle D, \sqsubseteq\rangle$ be a domain. A *prime interval* is a pair $[d, d']$ of elements of D such that $d \prec d'$. Let us define $[c, c'] \leq [d, d']$ if $(c = c' \sqcap d) \wedge (c' \sqcup d = d')$ and let $\sim$ be the equivalence obtained as the transitive and symmetric closure of (the preorder) $\leq$. The intuition that a prime interval represents a pair of elements differing only for a "quantum" of information is confirmed by the fact that there exists a bijective correspondence between $\sim$-classes of prime intervals and complete primes of a domain D (see [16]). More precisely, the map $[d, d']_\sim \mapsto p$ where p is the only element in $Pr(d') - Pr(d)$, is an isomorphism between the $\sim$-classes of prime intervals of D and the complete primes $Pr(D)$ of D, whose inverse is the function: $p \mapsto [\bigsqcup\{c \in D \mid c \sqsubset p\}, p]_\sim$.

Definition 3. *Let D be a domain, then $\mathcal{P}(D) = \langle Pr(D), \leq, \#\rangle$ where $p \leq p'$ iff $p \sqsubseteq p'$ and $p \# p'$ iff $\neg(p \uparrow p')$, in a* PES.

Two domains D and D' are essentially the same (we denote it with $D \simeq D'$) whenever there exists a bijection f among the elements in D and D' such that $d \sqsubseteq d'$ iff $f(d) \sqsubseteq f(d')$. Similarly two PES's P and P' are essentially the same (we will indicate this with $P \cong P'$) whenever there exists a bijection g among the events such that $e \leq e'$ iff $g(e) \leq' g(e')$ and $e \# e'$ iff $g(e) \#' g(e')$. It is worth to observe that:

Theorem 2. *$\mathcal{L}(\mathcal{P}(D)) \simeq D$ and $\mathcal{P}(\mathcal{L}(P)) \cong P$.*

3 Event Structures with Simultaneity

Events in prime event structures may occur concurrently, i.e., neither being causally related nor being in conflict. Nevertheless there are situations in reality where a *bunch* of concurrent events have to happen simultaneously, or better, have to happen *together*, as we have discussed in the introduction. In this section we introduce the notion of prime event structure with simultaneity (ESS) and study the relationship with prime algebraic domain.

We briefly illustrate the intuition behind our formalization of ESS with some examples. Assume that there are three events e_1, e_2 and e_3, none of them is in conflict with the others and e_2 depends on e_1. Assume that e_2 should be simultaneous with e_3. In a PES we would have a configuration containing only e_3, as well as one with e_1 and e_3. However now we know that, *despite* e_1 and e_3 may occur independently, e_3 is simultaneous with e_2. There is no explicit dependency on e_1, and more complex situations would rule out this *ad hoc* solution. Now we have that the intuitively *legal* configurations are $\{e_1\}$ and $\{e_1, e_2, e_3\}$, as we will formalize later. Assume now that we have 5 events, e_i with $1 \leq i \leq 5$, and $e_1 \leq e_2$, $e_3 \leq e_4$ and e_5 is simultaneous either with e_2 or with e_4, and that e_1 is in conflict with e_3 (we omit the inherited pairs). It is clear that e_5

will happen either simultaneously with e_2 or with e_4, but we cannot conclude anything concerning any dependency of e_5 from other events.

We can now turn to the formal treatment.

Definition 4. *Let* $P = \langle E, \leq, \# \rangle$ *be a* PES *and let* $s, s' \in \mathbf{2}^E_{\mathit{fin}} \setminus \{\emptyset\}$ *such that* $\mathbf{co}(s)$, $\mathbf{co}(s')$ *and* $s \cap s' = \emptyset$. *We say* s precedes s', *denoted with* $s \Subset s'$ *if there exists* $e \in s$ *and* $e' \in s'$ *such that* $e \leq e'$, *and we say the* s immediately dissents from s', *denoted with* $s \circledast^i s'$, *if there exists* $e \in s$ *and* $e' \in s'$ *such that* $e \# e'$.

Let $\mathbf{S} \subseteq \mathbf{2}^E_{\mathit{fin}}$, *with* $\hat{\Subset}$ *we denote the transitive and reflexive closure of* $\Subset$ *on* $\mathbf{S}$, *and with* $\circledast \subseteq \mathbf{S} \times \mathbf{S}$ *the relation defined as* $s \circledast s'$ *iff* $s \circledast^i s'$ *or* $\exists s'' \in \mathbf{S}$ *such that* $s'' \hat{\Subset} s'$ *and* $s \circledast^i s''$.

We enrich the notion of prime event structure with a set of subset of events that have to occur together.

Definition 5. *An* event structures with simultaneity *(*ESS*) is a tuple* $G = \langle E, \leq, \#, \mathbf{S} \rangle$ *where (1)* $\langle E, \leq, \# \rangle$ *is a* PES*; (2)* $\mathbf{S} \subseteq \mathbf{2}^E_{\mathit{fin}}$ *is such that (a)* $\bigcup \mathbf{S} = E$, *(b)* $\emptyset \notin \mathbf{S}$, *(c)* $\forall s \in \mathbf{S}.\ \mathbf{co}(s)$, *(d)* $\forall s \in \mathbf{S}\ \nexists s' \in \mathbf{S}.\ s' \subset s$, *and (e)* $\forall s, s' \in \mathbf{S}$, $s \cap s' \neq \emptyset$ *implies that* $\forall e \in s \setminus s',\ \forall e' \in s' \setminus s.\quad e \# e'$*; (3)* $(\mathbf{S}, \hat{\Subset})$ *is a partial order; and (4)* $\hat{\Subset}$ *and* $\circledast$ *on* $\mathbf{S}$ *are such that* $\hat{\Subset} \cap \circledast = \emptyset$.

The unique novelty in this definition with respect to the usual one of prime event structure is that we explicitly indicate which subsets of events may occur simultaneously ($\mathbf{S}$). On $\mathbf{S}$ we pose some obvious requirements: (a) the union of all simultaneous events must cover the whole set of events; (b) the empty set does not belong to $\mathbf{S}$; (c) each subset of simultaneous events must contain only concurrent events, unless it is a singleton; (d) a set of simultaneous events cannot be extended with other simultaneous events; and (e) two subsets of simultaneous events may overlap but when they do so then the elements not in common must be in conflict. This last requirement captures the idea that if an event can be simultaneous with two other different events which do not belong to the same set of simultaneous events, then these two must belong to alternative computations. Finally we require that the relations defined on $\mathbf{S}$ according to definition 4 are disjoint and that $\hat{\Subset}$ is a partial order on $\mathbf{S}$. These conditions capture the fact that each set of simultaneous events can be *executed*, i.e., added to a configuration (which we will formalize later).

Proposition 1. *Let* $G = \langle E, \leq, \#, \mathbf{S} \rangle$ *be an* ESS. *Then* $\circledast$ *is an irreflexive and symmetric relation which is hereditary with respect to* $\hat{\Subset}$.

Requiring that some events have to occur together implies that the notion of configuration have to be changed accordingly. We recall that a configuration of a PES is a left-closed and conflict-free subset of events. Now we have to consider the fact the certain set of events have to be simultaneous.

Definition 6. *Let* $G = \langle E, \leq, \#, \mathbf{S} \rangle$ *be an* ESS. *Then* $C \subseteq E$ *is a* configuration *iff (a) for all* $e, e' \in C$ $\neg(e \# e')$ (conflict-freeness), *(b)* $\lfloor e \rfloor \subseteq C$, (left-closedness), *and (c) there exists* $S \subseteq \mathbf{S}$ *such that* $\bigcup S = C$ (coverability). *The set of configurations of an* ESS *is denoted with* $\mathit{Conf}_{\mathit{ess}}(G)$.

A configuration C of an ESS is a left-closed and conflict free subset of events as for the underlying PES; and furthermore there should be a partition of events in C such that each element of the partition is a set of simultaneous events. It is worth noting that each configuration has a unique *decomposition* into sets of simultaneous events. To show this, assume that there exists another partition $S' \subseteq \mathbf{S}$ of C such that $C = \bigcup S'$. For an event $e \in C$ there must be sets of simultaneous event $s \in S$ and $s' \in S'$ such that $s \neq s'$ and $e \in s \cap s'$. But then, by definition of ESS, all the events in s and s' not belonging to their intersection are in conflict, contradicting the assumption we have made that S' is another coverability of the events in C. Furthermore each set of simultaneous events belong to a configuration.

Proposition 2. *Let $G = \langle E, \leq, \#, \mathbf{S}\rangle$ be an* ESS*, and let $C \subseteq E$ be a configuration. Then there exists a unique $S \subseteq \mathbf{S}$ such that $C = \bigcup S$.*

Proposition 3. *Let $G = \langle E, \leq, \#, \mathbf{S}\rangle$ be an* ESS*, and let $s \in \mathbf{S}$. Then there exists a configuration C such that $s \subseteq C$.*

We investigate now on the order on configurations.

Definition 7. *Let $G = \langle E, \leq, \#, \mathbf{S}\rangle$ be an* ESS*, and let $A, A' \subseteq E$ be sets of events. We say that A'* extends *A (written $A \sqsubseteq A'$), if (1) $A \subseteq A'$, and (2) $A' \setminus A \neq \emptyset$ implies that there exists $S \subseteq \mathbf{S}$ such that $\bigcup S = A' \setminus A$.*

We simply require that the events in $A' \setminus A$ can be covered by sets in $\mathbf{S}$.

Proposition 4. *Let G be and* ESS*, then $(Conf_{ess}(G), \sqsubseteq)$ is a partial order.*

We relate now PES and ESS. We show that ESS's are a proper extension of PES's, in the sense that, as one would expect (and require), prime event structures can be identified with a subclass of event structures with simultaneity where the set of sets of simultaneous events is given by singletons. The fact that, given a PES $\langle E, \leq, \#\rangle$, the set $\{\{e\} \mid e \in E\}$ is a well defined set of sets of simultaneous events, is a trivial observation.

Proposition 5. *Let $P = \langle E, \leq, \#\rangle$ be a* PES*. Then $\mathcal{J}(P) = \langle E, \leq, \#, \mathbf{S}\rangle$ where $\mathbf{S} = \{\{e\} \mid e \in E\}$ is an* ESS*.*

It is straightforward to observe that the configurations of a PES P are the same of the associated ESS $\mathcal{J}(P)$. In fact, given $C \in Conf_{\text{pes}}(P)$, then $C \in Conf_{\text{ess}}(\mathcal{J}(P))$ as well as it is clearly coverable being $\mathbf{S} = \{\{e\} \mid e \in E\}$.

Proposition 6. *Let P be a* PES *and let $\mathcal{J}(P)$ be the associated* ESS*. Then $Conf_{pes}(P) = Conf_{ess}(\mathcal{J}(P))$.*

Here we relate the new kind of event structure we have introduced and domains, by showing that the configurations of an ESS form a domain. We start identifying the least upper bound and the greatest lower bound of compatible configurations.

Lemma 1. *Let $G = \langle E, \leq, \#, \mathbf{S}\rangle$ be an* ESS*. Then (1) if $A \subseteq Conf_{\text{ESS}}(G)$ is pairwise compatible, then $\bigsqcup A = \bigcup A$, and (2) if $C_0 \uparrow C_1$ then $C_0 \sqcap C_1 = C_0 \cap C_1$.*

An event e can be added to a configuration provided that all the events which are simultaneous with e can be added, hence an event may have different histories, as each event may belong to various sets of simultaneous events. Recall the second example we have discussed at the beginning of this section, there the event e_5 is simultaneous either with e_2 or with e_4. Thus it can happen in two different histories. Let $s \in \mathbf{S}$, with $\Downarrow s$ we denote the set $A_s \subseteq \mathbf{S}$ defined as $A_s = \{s' \in \mathbf{S} \mid s' \hat{\Subset} s\}$. We can now formalize, for each event in an ESS, what its possible histories are.

Definition 8. *Let G be an* ESS *and let $e \in E$. Given a configuration $C \in Conf_{ess}(G)$ such that $e \in C$, then the* history *of e in C is defined as $C[\![e]\!] = (\bigcup \Downarrow s_e) \cap C$, where $s_e \in \mathbf{S}$ is such that $e \in s$. The set of possible histories of e, denoted by $Hist(e)$, is then defined as $Hist(e) = \{C[\![e]\!] \mid C \in Conf_{ess}(G) \ \wedge \ e \in C\}$.*

With $Hist(G)$ we denote the set of all possible histories of events in G, namely $Hist(G) = \bigcup\{Hist(e) \mid e \in E\}$.

The intuition is that, in a possible history of an event e with respect to a configuration C, we have to include all the events belonging to C which are simultaneous. We state now some properties of the possible histories that are needed in proving that $(Conf_{\text{ess}}(G), \sqsubseteq)$. Let $G = \langle E, \leq, \#, \mathbf{S}\rangle$ be an ESS. Then (1) if $C \in Conf_{\text{ess}}(G)$, then $C[\![e]\!] \in Conf_{\text{ess}}(G)$ and $C[\![e]\!] \sqsubseteq C$, and (2) if $C, C' \in Conf_{\text{ess}}(G)$, $C \uparrow C'$ and $e \in C \cap C'$, then $C[\![e]\!] = C'[\![e]\!]$. Furthermore, for all configurations $C \in Conf_{\text{ess}}(G)$ the followings hold: $C = \bigsqcup\{C' \in Hist(G) \mid C' \subseteq C\}$ $= \bigsqcup\{C[\![e]\!] \mid e \in C\}$, $Pr(Conf_{\text{ess}}(G)) = Hist(G)$ and $Pr(C) = \{C[\![e]\!] \mid e \in C\}$. We can now prove the following theorem, stating the main contributions of this section, namely that the configurations of an ESS are a domain.

Theorem 3. *Let G be an* ESS. *Then $\mathcal{L}_{ess}(G) = (Conf_{ess}(G), \sqsubseteq)$ is a domain.*

We associate an ESS to a domain as before, the only difference being that the sets of simultaneous events are singletons, as in the domain any information on which events may be simultaneous is not available.

Proposition 7. *Let D be a domain, then $\mathcal{P}_{ess}(D) = \langle Pr(D), \leq, \#, \mathbf{S}\rangle$, where $p \leq p'$ iff $p \sqsubseteq p'$, $p \# p'$ iff $\neg(p \uparrow p')$, and $\mathbf{S} = \{\{p\} \mid p \in Pr(D)\}$, is an* ESS.

As in the case of prime event structure we can show that the domain obtained by the configuration of the ESS associated to the original domain is essentially the original domain, namely given a domain D, then $\mathcal{L}_{ess}(\mathcal{P}_{ess}(D)) \simeq D$. However we cannot prove the vice versa, i.e., that, given an ESS G, $\mathcal{P}_{ess}(\mathcal{L}_{ess}(G)) \cong_{ess} G$, being $\cong_{ess}$ the obvious extension to ESS's of $\cong$ defined on PES's, as the following example shows. Consider an ESS with only two simultaneous events, say e, e'. The configurations are $\emptyset$ and $\{e, e'\}$. The ESS obtained by this domain has only one event, say p, hence there cannot be any bijection between $\{e, e'\}$ and $\{p\}$. We can still prove that, given a PES P, $\mathcal{P}_{ess}(\mathcal{L}(P)) \cong_{ess} \mathcal{J}(P)$.

4 Morphisms

In the previous section we have introduced the notion of ESS and shown that its configurations form a domain. The fact that $\mathcal{P}_{ess}(\mathcal{L}_{ess}(G)) \cong_{ess} G$ does not hold suggests that turning ESS into a category would not easily lead to the same positive results as in the case of PES and domains. In this section we investigate on the notion of morphism for ESS. Being able to define morphism is quite crucial, as we can relate in this way various models. As in the case of PES, the notion of morphism should capture the idea that an ESS G should simulate another one, say G'. Another requirement is that morphisms carry over the constructions we have seen so far, i.e., the domain associated to an ESS, turning out these constructions into functors.

Let first recall the notion of morphism on PES. Let $P_0 = \langle E_0, \leq_0, \#_0\rangle$ and $P_1 = \langle E_1, \leq_1, \#_1\rangle$ be two PES's. A PES-*morphism* $f : P_0 \to P_1$ is a partial function $f : E_0 \to E_1$ such that for all $e_0, e_0' \in E_0$, assuming that $f(e_0)$ and $f(e_0')$ are defined, $\lfloor f(e_0)\rfloor \subseteq f(\lfloor e_0\rfloor)$, $f(e_0) = f(e_0') \ \wedge \ e_0 \neq e_0' \ \Rightarrow \ e_0\#_0 e_0'$, and $f(e_0)\#_1 f(e_0') \ \Rightarrow \ e_0\#_0 e_0'$. With this notion of morphism, PES are turned out into the category of prime event structures and PES-morphisms, which is denoted by **PES**.

We could easily adapt this notion to the case of ESS, by requiring simply that the image of a set of simultaneous events is still a set of simultaneous events, i.e. for all $s \in \mathbf{S}_0$, $f(s) \neq \emptyset$ implies $f(s) \in \mathbf{S}_1$, but this notion basically fails in capturing the possible gluing of two events belonging to the same set of simultaneous events, which we will allow. The usual notion of morphism is injective on configurations, but this requirement forbids that simultaneous events can be identified. Similarly to our setting, Abbes in [1] drops this requirement asking in turn for an order preserving mapping. Events can be identified in a configuration (hence dropping the first part of condition 2 in the definition above). As observation it happens that some classical morphism are not Abbes' morphism, but still an adjunction with the category of domain is established. Here we propose another approach, weakening the requirement that morphisms on configurations should be injective, but requiring that another mapping is an ordinary PES morphism. Before introducing the notion, we recall that to each ESS G, another PES can be associated, different from the one obtained forgetting the set $\mathbf{S}$, which is a consequence of proposition 1.

Corollary 1. *Let $G = \langle E, \leq, \#, \mathbf{S}\rangle$ be an* ESS*. Then $\langle \mathbf{S}, \hat{\Subset}, \circledast\rangle$ is a* PES*.*

Let $f : E_0 \to E_1$ be a partial function, then $\hat{f} : \mathbf{2}^{E_0}_{\mathit{fin}} \to \mathbf{2}^{E_1}_{\mathit{fin}}$ is the mapping defined as $\hat{f}(X) = \{f(e) \mid e \in X \text{ and } f(e) \text{ is defined}\}$.

Definition 9. *Let $G_0 = \langle E_0, \leq_0, \#_0, \mathbf{S}_0\rangle$ and $G_1 = \langle E_1, \leq_1, \#_1, \mathbf{S}_1\rangle$ be two* ESS*'s. An* ESS-morphism *$f : G_0 \to G_1$ is a function $f : E_0 \to E_1$ such that: (1) $\forall s \in \mathbf{S}_0.\ f(s) \neq \emptyset$ implies $f(s) \in \mathbf{S}_1$, and (2) $\hat{f} : \langle \mathbf{S}_0, \hat{\Subset}_0, \circledast_0\rangle \to \langle \mathbf{S}_1, \hat{\Subset}_1, \circledast_1\rangle$ is a* PES*-morphism*

The last requirement states that the two PES associated to the ESS, are related by a morphism arising from f.

We show that this is well defined notion of morphism, as it composes and preserves configurations, hence ESS together with ESS-morphism form a category.

Proposition 8. *Let $G_0 = \langle E_0, \leq_0, \#_0, \mathbf{S}_0\rangle$ and $G_1 = \langle E_1, \leq_1, \#_1, \mathbf{S}_1\rangle$ be two* ESS*'s. Let $f : G_0 \to G_1$ be a morphism, and $C \in Conf_{ess}(G_0)$. Then $f(C) \in Conf_{ess}(G_1)$.*

Proposition 9. *Let $f : G_0 \to G_1$ and $g : G_1 \to G_2$ be two* ESS*-morphisms. Then $g \circ f : G_0 \to G_2$ is an* ESS *morphisms.*

Proposition 10. *Event structures with simultaneity and* ESS*-morphisms form a category which is denoted by* **ESS**.

We show that there is precise and nice relation between PES and ESS. Consider two PES P_0 and P_1. If $f : P_0 \to P_1$ is a PES-morphism, then f is an ESS-morphism between the corresponding ESS's $\mathcal{J}(P_0)$ and $\mathcal{J}(P_1)$. Moreover if $g : \mathcal{J}(P_0) \to \mathcal{J}(P_1)$ is an ESS-morphism then it is also a PES-morphism between P_0 and P_1. Thus account to say that $\mathcal{J}$ is a full embedding of **PES** into **ESS**.

Proposition 11. *The functor $\mathcal{J}$:* **PES** $\to$ **ESS** *defined by $\mathcal{J}(\langle E, \leq, \#\rangle) = \langle E, \leq, \#, \mathbf{S}\rangle$, where $\mathbf{S} = \{\{e\} \mid e \in E\}$, and $\mathcal{J}(f : P_0 \to P_1) = f$ is a full embedding of* **PES** *into* **ESS**.

Let us recall now the notion of domain morphisms.

Definition 10. *Let D_0 and D_1 be domains. A* domain morphism *$f : D_0 \to D_1$ is a function, such that: (1) $\forall x, y \in D_0$, if $x \preceq y$ then $f(x) \preceq f(y)$* ($\preceq$-preserving)*, (2) $\forall X \subseteq D_0$, X pairwise compatible, $f(\bigsqcup X) = \bigsqcup f(X)$* (Additive)*, and (3) $\forall X \subseteq D_0$, $X \neq \emptyset$ and compatible, $f(\bigsqcap X) = \bigsqcap f(X)$* (Stable)*. We denote by* **Dom** *the category having domains as objects and domain morphisms as arrows.*

Definition 11. *Given a domain morphism $f : D_0 \to D_1$, the associated* PES*-morphism $\mathcal{P}(f) : \mathcal{P}(D_0) \to \mathcal{P}(D_1)$ is the function $\mathcal{P}(f)(p_0) = p_1$ if $p_0 \mapsto [d_0, d_0']_\sim$, $f(d_0) \prec f(d_0')$ and $[f(d_0), f(d_0')]_\sim \mapsto p_1$, and $\mathcal{P}(f)(p_0) = \bot$ otherwise, i.e., if $f(d_0) = f(d_0')$.*

Now, the category **Dom** is known to be equivalent to the category **PES**, the equivalence being established by two functors $\mathcal{L}$: **PES** $\to$ **Dom** and $\mathcal{P}$: **Dom** $\to$ **PES**:

$$\begin{tikzcd} \mathbf{PES} \arrow[r, shift right, "\mathcal{L}"'] \arrow[r, phantom, "\sim"] & \mathbf{Dom} \arrow[l, shift right, "\mathcal{P}"'] \end{tikzcd}$$

We extend this machinery to **ESS** and **Dom**. We first show that the set extension of an ESS-morphism is indeed a domain morphism, and then $\mathcal{L}_{ess}$ can be easily extended to be a functor.

Proposition 12. *Let $G_0 = \langle E_0, \leq_0, \#_0, \mathbf{S}_0\rangle$ and $G_1 = \langle E_1, \leq_1, \#_1, \mathbf{S}_1\rangle$ be two* ESS*'s, and let $f : G_0 \to G_1$ be an* ESS-morphism. *Then the set extension of f is a domain morphism $f : Conf_{ess}(G_0) \to Conf_{ess}(G_1)$.*

Proposition 13. *The functor $\mathcal{L}_{ess}$:* **ESS** $\to$ **Dom** *associating to each* ESS *the domain of its configuration and to each* ESS*-morphism its set extension is a well defined functor.*

Similarly to what we have done for **PES** and **Dom**, we associate to each domain morphism an ESS-morphism, which acts on prime intervals as the PES-morphism does. Note that the definition of ESS-morphism associated to a domain morphism does not change with respect to the case of PES-morphism.

Proposition 14. *Let $f : D_0 \to D_1$ be a domain morphism, then the morphism $\mathcal{P}_{ess}(f) : \mathcal{P}_{ess}(D_0) \to \mathcal{P}_{ess}(D_1)$ is an* ESS*-morphism.*

With this machinery we have the following:

Proposition 15. *The functor $\mathcal{P}_{ess}$: $\mathbf{Dom} \to \mathbf{ESS}$ is a well defined functor. Furthermore $\mathcal{P}_{ess} = \mathcal{J} \circ \mathcal{P}$.*

We can now show that $\mathcal{P}_{ess}$ is the right adjoint to $\mathcal{L}_{ess}$.

Theorem 4. $\mathcal{P}_{ess} \vdash \mathcal{L}_{ess}$.

5 Conclusions and Future Works

In this paper we have introduced ESS in order to model simultaneity of events and shown that their configurations form a prime algebraic domain and we have established an adjunction between the categories **ESS** and **Dom**. The notion we have developed has the advantage of representing *faithfully* the situations where several independent agents, when possible, really cooperate to accomplish a task. Up to now this is modeled by observing ex-post, whereas we believe that the correct modeling of this situation helps in allowing abstractions that are not *too* abstract, making them of little interest.

Several problems remain. First of all, till now we have not put forward any relation on events which will *generate* the set **S**. This will subject to further investigation, along the line of the Relational Structure approach of Janicki ([12]). Here the problem is that an event may belong to various sets of simultaneous events, provided that they differ for conflicting ones, thus just imposing the transitivity to the relation of concurrency would not work. Furthermore this interpretation would require that concurrent events have to happen always together, that contrast our intuition. Second we have to extend out machinery to the other brands of event structures. It should be noticed that, similarly to what happen with Chu-spaces, $\mathcal{L}_{ess}$ is the left adjoint, which contrast with the adjunction that is usually established.

Acknowledgement. We thank the anonymous referees for the useful suggestions.

References

1. Abbes, S.: A cartesian closed category of event structures with quotients. Discrete Mathematics & Theor. Comput. Sci. 8(1), 249–272 (2006)
2. Baldan, P., Busi, N., Corradini, A., Pinna, G.M.: Domain and event structure semantics for Petri nets with read and inhibitor arcs. Theor. Comput. Sci. 323(1-3), 129–189 (2004)

3. Baldan, P., Corradini, A., Montanari, U.: Contextual Petri nets, asymmetric event structures and processes. Inf. Comput. 171(1), 1–49 (2001)
4. Boudol, G.: Flow Event Structures and Flow Nets. In: Guessarian, I. (ed.) LITP 1990. LNCS, vol. 469, pp. 62–95. Springer, Heidelberg (1990)
5. Bruni, R., Montanari, U.: Zero-safe nets: Comparing the collective and individual token approaches. Inf. Comput. 156(1-2), 46–89 (2000)
6. Busi, N.: Causality in membrane systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) WMC 2007. LNCS, vol. 4860, pp. 160–171. Springer, Heidelberg (2007)
7. Goltz, U., Reisig, W.: The non-sequential behavior of Petri nets. Information and Control 57(2/3), 125–147 (1983)
8. Gunawardena, J.: Geometric logic, causality and event structures. In: Groote, J.F., Baeten, J.C.M. (eds.) CONCUR 1991. LNCS, vol. 527, pp. 266–280. Springer, Heidelberg (1991)
9. Gunawardena, J.: Causal automata. Theor. Comput. Sci. 101(2), 265–288 (1992)
10. Hoogers, P.W., Kleijn, H.C.M., Thiagarajan, P.S.: An event structure semantics for general Petri nets. Theor. Comput. Sci. 153(1-2), 129–170 (1996)
11. Janicki, R., Koutny, M.: Semantics of inhibitor nets. Inf. Comput. 123, 1–16 (1995)
12. Janicki, R.: Relational structures model of concurrency. Acta Inf. 45(4), 279–320 (2008)
13. Janicki, R., Koutny, M.: Structure of concurrency. Theor. Comput. Sci. 112(1), 5–52 (1993)
14. Juhás, G., Lorenz, R., Mauser, S.: Causal semantics of algebraic Petri nets distinguishing concurrency and synchronicity. Fundam. Inform. 86(3), 255–298 (2008)
15. Kleijn, J., Koutny, M., Rozenberg, G.: Process semantics for membrane systems. Journal of Automata, Languages and Combinatorics 11(3), 321–340 (2006)
16. Nielsen, M., Plotkin, G., Winskel, G.: Petri Nets, Event Structures and Domains, Part 1. Theor. Comput. Sci. 13, 85–108 (1981)
17. Pãun, G.: Computing with membranes. J. Comput. Syst. Sci. 61(1), 108–143 (2000)
18. Pinna, G.M., Poigné, A.: On the nature of events: another perspective in concurrency. Theor. Comput. Sci. 138(2), 425–454 (1995)
19. Pinna, G.M., Saba, A.: An Event Based Semantics of P Systems. Scientific Annals of Computer Science XVIII, 99–127 (2008)
20. Pratt, V.R.: Higher dimensional automata revisited. Mathematical Structures in Computer Science 10(4), 525–548 (2000)
21. Pratt, V.R.: Transition and cancellation in concurrency and branching time. Mathematical Structures in Computer Science 13(4), 485–529 (2003)
22. Winskel, G.: Event Structures. In: Brauer, W., Reisig, W., Rozenberg, G. (eds.) APN 1986. LNCS, vol. 255, pp. 325–392. Springer, Heidelberg (1987)

Safety Verification of Non-linear Hybrid Systems Is Quasi-Semidecidable

Stefan Ratschan[⋆]

Institute of Computer Science, Academy of Science of the Czech Republic
stefan.ratschan@cs.cas.cz

Abstract. Safety verification of hybrid systems is undecidable, except for very special cases. In this paper, we circumvent undecidability by providing an algorithm that can verify safety and provably terminates for all robust and safe problem instances. It need not necessarily terminate for problem instances that are unsafe or non-robust. A problem instance x is robust iff the given property holds not only for x itself, but also when x is perturbed a little bit. Since, in practice, well-designed hybrid systems are usually robust, this implies that the algorithm terminates for the cases occurring in practice. In contrast to earlier work, our result holds for a very general class of hybrid systems, and it uses a continuous time model.

1 Introduction

Terminating algorithms for the verification of hybrid systems are known only for very special cases. In fact, most classes of hybrid systems verification problems are known to be undecidable [8] (and not even semi-decidable). Recently, there have been attempts at circumventing this [6,5] by observing that, in practice, hybrid systems can never model a given real system precisely, but only up to perturbations. Hence it suffices to verify robust systems, that is, systems that do not change the desired property under perturbations.

We say that a property is *quasi-semidecidable* iff a (possibly non-terminating) algorithm exists that can correctly check the correctness of the property, but which is required to terminate only for robust problem instances. We show quasi-semidecidability of safety verification of a class of hybrid systems that allows arbitrary Boolean combinations of non-linear differential equalities and inequalities for defining the continuous flow, and arbitrary Boolean combinations of non-linear equalities and inequalities for defining the set of initial and unsafe states, and for defining the set of possible discontinuous jumps of the system. We also have a proof for the dual case, verifying that a robust system is *not* safe, but due to space restrictions, we will leave this case for an extended version of the paper.

[⋆] The work of Stefan Ratschan has been supported by GAČR grant 201/08/J020 and by the institutional research plan AV0Z100300504.

© Springer-Verlag Berlin Heidelberg 2010

This theoretical result has heavy practical consequences: Up to now, hybrid systems verification algorithms have been evaluated purely experimentally, on finitely many benchmark examples. However, one would like a practical verification algorithm to terminate for *all* robust inputs. Hence, for the first time, we now have a formal tool to evaluate practical verification algorithms for a comprehensive class of hybrid systems.

Concerning related work, a recent article [2, Section 5] includes a survey on the role of noise and robustness in continuous-time dynamical systems.

Similar quasi-decidability results as the ones presented in the present paper have been obtained (under different names) for systems with simpler dynamics: Fränzle [6] provides results for the case where the input system is completely defined by polynomials. Especially, continuous evolution is given by explicit polynomial flows which, in general, does not even allow the modeling of linear differential equations, since these can have non-polynomial flows as solutions.

Puri and co-authors [12] show how to compute an over-approximation of Lipschitz differential inclusions with known Lipschitz constant over a finite time horizon. This implies a corresponding quasi-decidability result. In contrast to that, our result allows unbounded time, and does not require a previously known Lipschitz constant.

Collins [3] studies approximation of reach sets of dynamical system in an effective computable analysis framework which again implies a corresponding quasi-decidability result. He uses a discrete time model (such a model can in certain cases encode a continuous time model). In the continuous time case there is corresponding work on approximating reach sets over a finite time horizon [4].

Damm and co-authors [5] provide a similar result as ours for a discrete time model. The continuous time model employed in this paper, implies several additional difficulties:

– When considering syntactic descriptions of systems, in a discrete time model all variables vary over the state space of the system, whereas in a continuous time model, some variables (describing differentiation) do not. Hence these variables may take unbounded values even if the state space is bounded. This needs additional deduction mechanisms for capturing the set of possible values that these variable may take and proofs of their correctness (Theorem 2) and convergence (Lemma 1).
– In a discrete time model, a trajectory only reaches finitely many states in a finite time interval, whereas in a continuous time model it usually reaches uncountably many. This uncountable set has to be captured by corresponding algorithms. As a consequence, abstraction to a finite state systems, as used in the earlier paper, cannot capture system behavior arbitrarily closely, since even arbitrary refinements cannot separate two subsequent steps of the system.

Asarin and Bouajjani provide a comparison on the effect of perturbations in various dynamical system models [1].

On the negative side, Henzinger and Raskin showed that certain undecidability results for hybrid systems continue to hold, even if in the proof one only allows encodings into robust trajectories [9]. This does not contradict our result for two reasons: First, quasi-decidability allows an algorithm that does not always (i.e., for non-robust inputs) terminate, whereas undecidability (even when based on robust trajectories) proves non-existence of an algorithm that *terminates always*. Second, in a similar way as Fränzle [6], we require a compact state space, whereas Henzinger and Raskin do not (although their dynamics is *much* simpler than ours).

2 Hybrid Systems and Their Quasi-Decidability

In this section we describe the solved problem in detail. In the literature, state spaces of dynamical systems are usually defined using tuples (for example, tuples in $\mathbb{R}^n$). Here, we take a little bit more flexible approach, that allows us to directly access the individual tuple elements using names. For these names we use a finite set V whose elements we call *variables*. Moreover, we use the set $\dot{V} \doteq \{\dot{v} \mid v \in V\}$ to access the values of derivatives, and the set $V' \doteq \{v' \mid v \in V\}$ to access the result of a discrete state change (i.e., of jumps). Moreover, we fix a finite set M whose elements we call *modes*, and use the additional specific variable name $\texttt{mode}$ to access them, and the variable name $\texttt{mode}'$ to access them in the case of the result of a discrete jump.

Now we call a function that assigns to some symbols from $\{\texttt{mode}, \texttt{mode}'\}$ a value from M and to some elements of $V \cup \dot{V} \cup V'$ a real value a *valuation*. These valuations will take the role of tuples to form the state space of hybrid systems. For a subset X of $\{\texttt{mode}, \texttt{mode}'\} \cup V \cup \dot{V} \cup V'$ we denote the set of valuations that assigns values exactly to the elements of X by $\Gamma(X)$.

For every valuation σ in $\Gamma(\{\texttt{mode}\} \cup V)$, we denote by $\texttt{Prime}(\sigma)$ the corresponding valuation with primed variables, that is, $\texttt{Prime}(\sigma)$ is a valuation in $\Gamma(\{\texttt{mode}'\} \cup V')$, and for all $v \in \{\texttt{mode}\} \cup V$, $\texttt{Prime}(\sigma)(v') = \sigma(v)$.

For two valuations σ_1, σ_2 that coincide on joint variables, we define their concatenation $\sigma_1 \bullet \sigma_2$ as the valuation that is defined on the union of the two domains of definition and always assigns the corresponding value. That is, for $\sigma_1 \in \Gamma(X_1)$ and $\sigma_2 \in \Gamma(X_2)$ such that for all $v \in X_1 \cap X_2$, $\sigma_1(v) = \sigma_2(v)$, we have that for all $v \in X_1$, $(\sigma_1 \bullet \sigma_2)(v) = \sigma_1(v)$, and for all $v \in X_2$, $(\sigma_1 \bullet \sigma_2)(v) = \sigma_2(v)$.

Definition 1. *A* hybrid system *is a tuple of the form* $(S, Init, Flow, Jump, Unsafe)$ *where* S *(the* state space *of the hybrid system) is a subset of* $\Gamma(\{\texttt{mode}\} \cup V)$ *such that for every* $v \in V$ *we have a non-empty closed real interval* I_v *such that* $S = \{\sigma \mid \sigma \in \Gamma(\{\texttt{mode}\} \cup V), \sigma(v) \in I_v, v \in V\}$. *In other words, the continuous part of the state space has the form of a hyper-rectangle. In addition,*

- $Init \subseteq S$, $Unsafe \subseteq S$.
- $Flow \subseteq \Gamma(\{\texttt{mode}\} \cup V \cup \dot{V})$, *such that for all* $\sigma \in Flow$, *for all* $v \in V$, $\sigma(v) \in I_v$,
- $Jump \subseteq \Gamma(\{\texttt{mode}\} \cup V \cup \{\texttt{mode}'\} \cup V')$, *such that for all* $\sigma \in Jump$, *for all* $v \in V$, $\sigma(v) \in I_v$ *and* $\sigma(v') \in I_v$.

That is, a hybrid system has a set of initial and unsafe elements that are sub-sets of the state space. Moreover, it relates derivatives to state space elements, and relates state space elements to primed versions of state space elements.

We will use the following objects to describe continuous evolution of hybrid systems:

Definition 2. *A* flow of length t over $S \subseteq \Gamma(\{\mathtt{mode}\} \cup V)$ *is a function* $\phi : [0, t] \to S$ *such that*

- $\phi(t)(\mathtt{mode})$ *is constant over all* $t \in [0, t]$*, and*
- *for every* $v \in V$*, the function* ϕ^v *that assigns to every* $s \in [0, t]$ *the value* $\phi(t)(v)$*, is differentiable.*

Based on this, we define $\dot{\phi} : [0, t] \to \Gamma(\dot{V})$ *in such a way that for every* $s \in [0, t]$*,* $v \in V$*,* $\dot{\phi}(s)(\dot{v}) = \dot{\phi}^v(s)$*.*

The property we study in this paper is reachability of the set of unsafe states:

Definition 3. *For a given hybrid system* $(S, \mathit{Init}, \mathit{Flow}, \mathit{Jump}, \mathit{Unsafe})$*, an* error trajectory *is a sequence of flows* $(\phi_0, \ldots, \phi_n)$ *over* S *of lengths* $(t_1, \ldots, t_n)$ *such that, for all* $i \in \{0, \ldots, n\}$

- *if* $i < n$*, then* $\phi_i(t_i) \bullet \mathtt{Prime}(\phi_{i+1}(0)) \in \mathit{Jump}$ *or* $\phi_i(t_i) = \phi_{i+1}(0)$
- *for all* $s \in [0, t_i]$*,* $\phi_i(s) \bullet \dot{\phi}_i(s) \in \mathit{Flow}$*,*

and $\phi_0(0) \in \mathit{Init}$*,* $\phi_n(t_n) \in \mathit{Unsafe}$*. A hybrid system is* safe *if it does not have an error trajectory.*

For describing hybrid systems we use constraints. We define an *arithmetical term* to be an expression that may contain variables in V, rational constants, and function symbols in $\{+, \times, \sin, \cos, \exp\}$. Now we define a constraint to be a Boolean combination of two types of atomic constraints:

- equalities and inequalities of the form $t \; r \; 0$, where t is an arithmetical term, and $r \in \{=, \leq, \geq, <, >\}$.
- equalities and inequalities of the form $\mathtt{mode} = m$ or $\mathtt{mode} \neq m$, where $m \in M$ (we call this a *mode constraint*).

A *flow constraint* is a constraint that, in addition to the above, allows atomic constraints of the form $\dot{v} \; r \; t$, where $r \in \{=, \leq\}$, v is a variable from V, and t is an arithmetical term over V. A *jump constraint* is a constraint that, in addition to the variables in $\{\mathtt{mode}\} \cup V$, allows their primed versions, that is, variables in $\{\mathtt{mode}'\} \cup V'$.

The definition of the semantics of such constraints is straight-forward. We denote the function from valuations to real numbers described by a term t by $[\![t]\!]$. We write $\sigma \models \phi$ for the fact that a valuation σ satisfies a constraint ϕ, and we write $[\![\phi]\!]$ for the set of valuations satisfying ϕ. We use corresponding definitions for flow and jump constraints in analogy.

Now we have a way of syntactically describing hybrid systems using constraints. For a given state space S, and constraints *Init*, *Flow*, *Jump*, and *Unsafe* we call the tuple $(S, \mathit{Init}, \mathit{Flow}, \mathit{Jump}, \mathit{Unsafe})$ a *hybrid systems description*, and denote by $[\![(S, \mathit{Init}, \mathit{Flow}, \mathit{Jump}, \mathit{Unsafe})]\!]$ the hybrid system $(S, [\![\mathit{Init}]\!], [\![\mathit{Flow}]\!], [\![\mathit{Jump}]\!], [\![\mathit{Unsafe}]\!])$.

In this case we also say that the hybrid system *fulfills* the corresponding hybrid systems description. We straightforwardly lift Definition 3 from hybrid systems to hybrid system descriptions.

It is well-known that—except for very special cases—checking whether a hybrid system is safe is an undecidable problem [8]. However, in the real world, we will not be able to implement a hybrid system description exactly and we do not want to prove a hybrid systems description safe, if the system fulfilling this description is safe but there is a system that fulfills the description up to small perturbations and is unsafe. Hence it suffices to have an algorithm that can prove safety of hybrid systems descriptions for which all hybrid systems that fulfill the description up to small perturbations are safe.

Here it does not suffice to perturb hybrid systems without regard to the language they are described in (i.e., it does not suffice to consider perturbations on the semantic level; one *has to* take into account syntactic perturbations). The reason for this can be seen in the example of a constraint $0 = 0$ whose solution set does not vanish under slight changes (since every valuation is a solution), but still, small perturbations of the constraint itself change the solution set essentially.

The basic idea for introducing such perturbations is to define a distance measure on constraints that measures in how far two constraints are the same up to "addition of constants up to a certain size":

Definition 4. *We call a term* basic*, if it is either a variable, or a constant, or a term of the form $x + c$, where x is a variable, and c a constant. If the set of variables contained in a term (this is either a singleton set or the empty set) is the same in two basic terms, we define the distance between these terms as the distance between the corresponding constants, using the constant* 0 *if one of the terms does not contain a constant. If the set of contained variables is not the same in both basic terms, their distance is ∞.*

The distance $d(\phi, \phi')$ between two constraints ϕ and ϕ' is ε iff ϕ' can be obtained from ϕ by replacing some basic terms by basic terms of finite distance and ε is the maximum of these distances. Otherwise, the distance is ∞.

Example 1. For measuring the distance between the constraint $(x+2)^2+1x \leq 0$ and $x^2+2x \leq 0$ we observe that for getting from the first to the second constraint we have to replace the basic term $x+2$ by x, and the basic term 1 by the basic term 2. The distance is the maximum of the distances of corresponding basic terms, that is, the distance is 2.

The constraints $(x-2)^2-1 \leq 0$ and $x^2-4x+4-1 \leq 0$, although semantically equivalent, have infinite distance. This does not pose any problem here. On the contrary, this makes our result stronger, since it leads to many hybrid systems descriptions being robust, and hence to a strong termination condition for our algorithms.

We continue with defining an analogon of the notion of "fulfilling a hybrid systems description up to ε" for our constraint language.

Definition 5. *A set P of valuations is an ε-*perturbed solution set *of a constraint ϕ iff for every valuation $\sigma \in P$, there is a constraint ϕ^* with $d(\phi, \phi^*) \leq \varepsilon$ such that $\sigma \models \phi^*$, and for every valuation $\sigma \notin P$, there is a constraint ϕ^* with $d(\phi, \phi^*) \leq \varepsilon$ such that $\sigma \not\models \phi^*$.*

In other words, the set P may contain valuations that do *not* satisfy the constraint, and may not contain valuations that *do* satisfy constraint, but we have to make sure that in both cases the error that we make is not too large.

Lifting this definition to hybrid systems is straightforward:

Definition 6. *Given a hybrid systems description D, a hybrid system H fulfills D up to ε iff for every constraint Init, Flow, Jump, Unsafe defining D, the corresponding set in H is an ε-perturbed solution set of the constraint.*

Definition 7. *A given hybrid systems description D is* robustly safe *iff there is a real number $\varepsilon > 0$ (the* robustness margin*) such that all hybrid systems fulfilling D up to ε are safe.*

Only in the non-robust case we do not require our algorithms to terminate, that is, in that case an algorithm trying to verify safety of a given hybrid system is allowed to run forever. This is the essential point, why the following notion of quasi-semidecidability is weaker than semidecidability.

Definition 8. *We call the problem of safety verification of hybrid systems* quasi-semidecidable *iff there is an algorithm A such that for a given hybrid systems description D*

- *if $A(D)$ terminates then D is safe (i.e., A is correct),*
- *$A(D)$ terminates if D is robustly safe.*

We are now ready to formulate the main theorem of this paper:

Theorem 1. *Safety of hybrid system descriptions is quasi-semidecidable.*

3 Quasi-Semidecidability of Verification

For proving quasi-semidecidability of verification we use the fact that for every hybrid system there is a rectangular ε-approximation [7]. Here we have to overcome two major obstacles:

- The original proof of this existence property was not constructive.
- Although rectangular automata have a much simpler structure than general hybrid systems, their safety is still undecidable.

Before solving these problems, we introduce a representation of rectangular sets: A *box* is a function that assigns to some variables in $V \cup \dot{V} \cup V'$ a non-empty closed real interval, and to some variables in $\{\texttt{mode}, \texttt{mode}'\}$ a subset of modes from M. Throughout the paper we use the situation that a box B does not assign a value to a given variable as a shortcut for the value $B(v)$ being M, if $v \in \{\texttt{mode}, \texttt{mode}'\}$, and being $[-\infty, \infty]$, otherwise. We will say that a box has *dimension* d iff it assigns d real intervals (i.e., d intervals not equal to $[-\infty, \infty]$). We lift set membership to boxes by defining a valuation σ to be element of a box B iff for every variable v on which σ is defined, $\sigma(v) \in B(v)$. Analogously we lift other set operations such as $\subseteq$ and $\cap$ using the corresponding variable-wise operations on intervals and sets of modes, respectively. Box union $\uplus$ is defined by lifting union for variables in $\{\texttt{mode}, \texttt{mode}'\}$, and interval union (the smallest interval containing both arguments) for the other variables. For boxes we define concatenation analogously as for valuations. We call a box *proper*, if it only assigns intervals (and no modes).

A *sat-box* (for satisfiability box) is either a box, or the value $\perp$ which we call the *empty box*. Such sat-boxes will be used for flow constraints where we either deduce unsatisfiability or a box bounding the set of possible derivatives. A sat-box has *dimension* d iff it is equal to $\perp$ or if it is a box of dimension d (hence $\perp$ can have any dimension). Sometimes we will write $\mathbf{F}$ for $\perp$ and $\mathbf{T}$ for the unique zero-dimensional box, and use them in the role of the corresponding Boolean constants. The box operations $\cap$ and $\uplus$ can be easily lifted from boxes to sat-boxes by considering $\perp$ to be the smallest element in the $\subseteq$ order. Also, the element relation $\in$ can be naturally lifted by defining $\perp$ to have no element (which, of course, corresponds to its name "empty box").

Now we start with removing the first obstacle mentioned at the beginning of this section: computing a rectangular over-approximation of a hybrid system such that the over-approximation error is smaller than a given bound.

The algorithm uses interval arithmetic as its basis. For a term t, and proper box B, let $I(t)(B)$ denote the evaluation of t on B using interval arithmetic [11]. The result over-approximates the set of all values the term t takes in the box B, due to the so-called Fundamental Theorem of Interval Arithmetic [10], that is, $I(t)(B) \supseteq \{[\![t]\!](\sigma) \mid \sigma \in B\}$.

Now we can over-approximate the satisfiability information of constraints by defining the symbol $\models_I$ (*interval satisfiability check*) for a box B as follows:

– $B \models_I \texttt{mode} = m$ is $\mathbf{T}$ if $m \in B(\texttt{mode})$, and $\mathbf{F}$ otherwise,
– $B \models_I t\ r\ 0$, where t does not contain dotted variables, is $\mathbf{T}$ iff there exists a real value $x \in I(B)(t)$ such that $x\ r\ 0$, and $\mathbf{F}$, otherwise,
– $B \models_I \phi_1 \wedge \phi_2$ is $B \models_I \phi_1 \cap B \models_I \phi_2$, and
– $B \models_I \phi_1 \vee \phi_2$ is $B \models_I \phi_1 \uplus B \models_I \phi_2$.

Example 2. Let ϕ be the constraint $x^2 - 1 = 0 \wedge x - 2 \geq 0$, and let B be the box $x \mapsto [-10, 0]$. Interval arithmetic evaluates the terms in ϕ recursively. So $I(x^2)(B) = [0, 100]$, and $I(x^2 - 1)(B) = [-1, 99]$. Since this interval contains zero, $(B \models_I x^2 - 1 = 0) = \mathbf{T}$. Moreover, $I(x - 2)(B) = [-12, -2]$, and $(B \models_I x - 2 \geq 0) = \mathbf{F}$. In the zero-dimensional case, intersection and union of boxes implements conjunction and disjunction of the corresponding Boolean values. So $(B \models_I \phi) = \mathbf{T} \cap \mathbf{F} = \mathbf{F}$.

Remember that, by default, variables are assigned the interval $[-\infty, \infty]$. Hence the semantics is also well-defined in cases where the branches of a conjunction (or disjunction) contain different variables.

We generalize the interval satisfiability check to constraints containing dotted variables (denoting derivatives). In this case, the result is a sat-box, whose dimension (if containing a box) is equal to the number of dotted variables. The purpose of this definition is to over-approximate the projection of the solution set of the constraint to these variables:

- $B \models_I \dot{a} = t$ is defined as $\{\dot{a} \mapsto I(t)(B)\}$
- $B \models_I \dot{a} \leq t$ is defined as $\{\dot{a} \mapsto [-\infty, \overline{I(t)(B)}]\}$
- $B \models_I \dot{a} \geq t$ is defined as $\{\dot{a} \mapsto [\underline{I(t)(B)}, \infty]\}$

The rest of the definition is kept unchanged.

Example 3. Let ϕ be the flow constraint $\dot{x} = x^2 \wedge x - 2 \geq 0$, and let B be the box $x \mapsto [1, 3]$. Then $B \models_I \dot{x} = x^2$ is the box $\{\dot{x} \mapsto [1, 9]\}$ and $(B \models_I x - 2 \geq 0) = \mathbf{T}$. Hence $\{\dot{x} \mapsto [1, 9]\} \cap \mathbf{T} = \{\dot{x} \mapsto [1, 9]\}$ (remember that $\mathbf{T}$ is the unique zero dimensional box that assigns to every variable the default interval $[-\infty, \infty]$).

For the slightly modified constraint $x^2 - 1 = 0 \wedge x - 10 \geq 0$, however, $B \models_I x - 10 \geq 0$ evaluates to $\mathbf{F}$, and hence also the whole constraint.

This definition fulfills its purpose due to the following generalization of the fundamental theorem of interval arithmetic to our constraints:

Theorem 2. *For every constraint ϕ, box B on the un-dotted variables of ϕ, valuation $\sigma \in B$, and valuation $\dot{\sigma}$ on the dotted variables of ϕ such that $\sigma \bullet \dot{\sigma} \models \phi$, we have $\dot{\sigma} \in B \models_I \phi$.*

Proof. Let B, σ, $\dot{\sigma}$ arbitrary, but fixed, fulfilling the assumptions above. We prove that $\dot{\sigma} \in \ B \models_I \phi$. We proceed by induction over the structure of ϕ. Due to space restrictions we discuss only the base case where ϕ is of the form $\dot{x} = t$. In this case, since $\sigma \bullet \dot{\sigma} \models \dot{x} = t$, it holds that $\dot{\sigma} = [\![t]\!](\sigma)$. To prove that $\dot{\sigma} \in B \models_I \phi$, we have to prove that $\dot{\sigma} \in I(t)(B)$. This holds since due to the fundamental theorem of interval arithmetic, $[\![t]\!](\sigma) \in I(t)(B)$ for $\sigma \in B$. The other cases of atomic constraints are analogous, and the induction step is easy. $\square$

Example 4. Continuing Example 3, let in addition σ be the valuation $\{x \mapsto 2\}$ (which is an element of B), and let $\dot{\sigma}$ be the valuation $\{\dot{x} \mapsto 4\}$ (for which $\sigma \bullet \dot{\sigma} \models \phi$). Then the box $B \models_I \phi$ which is $\{\dot{x} \mapsto [1, 9]\}$ contains the valuation $\{\dot{x} \mapsto 4\}$.

In the special case of constraints without dotted variables, the interval satisfiability just over-approximates satisfiability:

Corollary 1. *For every constraint ϕ without dotted variables, box B on the variables of ϕ, if B contains a valuation σ such that $\sigma \models \phi$, then $(B \models_I \phi) = \mathbf{T}$.*

Reading this corollary in the opposite direction, we can conclude that $(B \models_I \phi) = \mathbf{F}$ implies that there is no valuation $\sigma \in B$ such that $\sigma \models \phi$.

Now we are preparing to present an algorithm for which we will prove that it over-approximates a given hybrid system arbitrarily closely. For bounding the over-approximation error we use a bound on the size of the boxes. For a non-empty interval $[a, b]$, its width is defined to be $b - a$, and for a non-empty set of modes $M^* \subseteq M$, we define its width to be zero if M^* is a singleton set, and ∞, otherwise. We define the diameter $diam(B)$ of a box B to be the maximum width of $B(v)$ over all variables v on which B is defined (i.e., not equal $[-\infty, \infty]$).

The algorithm will approximate a given hybrid systems description using a hybrid systems description completely defined by boxes. Here we use the notation $x \in [\underline{a}, \overline{a}]$ as a short-cut for the constraint $\underline{a} \leq x \wedge x \leq \overline{a}$. The idea is to put a grid of boxes onto the state space, and then testing on each box using the interval satisfiability check, whether it might contain an initial or unsafe state, testing for every pair of boxes whether it might contain a jump between them, and computing an interval containing the possible derivatives for each box.

In contrast to the discrete time case [5], here it does not suffice to abstract to a purely discrete system. The reason is that in discrete time, if the hyper-rectangles are sufficiently small, they can separate two subsequent steps of the system. However, for continuous evolution, this is not possible. We use the following algorithm that takes as input a hybrid systems description $(S, Init, Flow, Jump, Unsafe)$, and a strictly positive real value δ:

```
$G \leftarrow$ set of boxes of diameter $\delta$ covering the state space $S$
$Init_R \leftarrow \bigvee_{B \in G, B \models_I Init} [\mathtt{mode} = B(\mathtt{mode}) \wedge \bigwedge_{v \in V} v \in B(v)]$
$Flow_R \leftarrow \bigvee_{B \in G} [\mathtt{mode} = B(\mathtt{mode}) \wedge$
                $\bigwedge_{v \in V} v \in B(v) \wedge \bigwedge_{v \in V} \dot{v} \in (B \models_I Flow)]$
$Jump_R \leftarrow \bigvee_{B, B' \in G, \langle B, B' \rangle \models_I Jump} [$
                $\mathtt{mode} = B(\mathtt{mode}) \wedge \bigwedge_{v \in V} v \in B(v) \wedge$
                $\mathtt{mode}' = B'(\mathtt{mode}) \wedge \bigwedge_{v \in V} v' \in B'(v)]$
$Unsafe_R \leftarrow \bigvee_{B \in G, B \models_I Unsafe} [$
                $\mathtt{mode} = B(\mathtt{mode}) \wedge \bigwedge_{v \in V} v \in B(v)]$
$(S, Init_R, Flow_R, Jump_R, Unsafe_R)$
```

We denote the result computed by this algorithm by $A(H, \delta)$. This is again a hybrid system description, and from Theorem 2 it easily follows that $A(H, \delta)$ over-approximates H, and hence the result of the algorithm can be used to prove safety of the original system.

Theorem 3. *For the result $(S, Init_R, Flow_R, Jump_R, Unsafe_R)$ of the algorithm application $A((S, Init, Flow, Jump, Unsafe), \delta)$, Init implies $Init_R$, Flow implies $Flow_R$, Jump implies $Jump_R$, and Unsafe implies $Unsafe_R$. Hence the safety of $[\![A((S, Init, Flow, Jump, Unsafe), \delta)]\!]$ implies the safety of $[\![(S, Init, Flow, Jump, Unsafe)]\!]$.*

However, the amount of over-approximation of the algorithm can still be arbitrarily large. In order to arrive at bounds for this over-approximation, we first study such bounds for constraints. In earlier work [5] we proved results bounding the over-approximation of $\models_I$ for constraints without dotted variables. We generalize those results here to the case with dotted variables:

Lemma 1. *For every constraint ϕ, box B defined on all un-dotted variables of ϕ, for all $\varepsilon > 0$ there is a $\delta > 0$ such that for every box B' with $B' \subseteq B$, $diam(B') < \delta$, for every $\sigma \in B'$, and for every $\dot{\sigma} \in (B' \models_I \phi)$, there is a ϕ^* with $d(\phi, \phi^*) \leq \varepsilon$, such that $\sigma \bullet \dot{\sigma} \models \phi^*$.*

Proof. For proving this lemma we use the fact (which we will call *convergence of interval arithmetic* in the rest of the proof) that for every arithmetical term e with function symbols in the set $\{+, *, \hat{}, \exp, \sin, \cos\}$, denoting a function $[\![e]\!]$ and box S, for every $\varepsilon > 0$ there is a $\delta > 0$ such that for every box B with $B \subseteq S$, $diam(B) < \delta$, for all $y \in I(e)(B)$, there is an $x \in B$ such that $d([\![e]\!](x), y) \leq \varepsilon$. This fact follows from Lipschitz continuity of interval arithmetic (e.g., Theorem 2.1.1 in Neumaier's book [11]).

Now let ϕ, B, ε be as required by the assumptions of the lemma. We discuss the case where ϕ is of the form $\dot{a} = t$.

Let δ_t be the value ensured for t, B, and ε by convergence of interval arithmetic. We choose δ as $\min\{\delta_t, \varepsilon\}$, assume an arbitrary but fixed box B', σ, and $\dot{\sigma}$ with $B' \subseteq B$, $diam(B') < \delta$, $\sigma \in B'$, and $\dot{\sigma} \in (B' \models_I \dot{a} = t)$.

From $\dot{\sigma} \in B' \models_I \dot{a} = t$ we know that $\dot{\sigma} \in I(t)(B')$. We construct a ϕ^* with $\sigma \bullet \dot{\sigma} \models \phi^*$ by providing the necessary perturbations. Let x be such that $d([\![t]\!](x), \dot{\sigma}) \leq \varepsilon$, as ensured by the convergence of interval arithmetic.

We perturb every un-dotted variable v of ϕ by $x(v) - \sigma(v)$ (this perturbation is smaller than ε since $d(\sigma(v), x(v)) \leq diam(B') \leq \delta = \min\{\delta_t, \varepsilon\} \leq \varepsilon$); the dotted variables by $[\![t]\!](x) - \dot{\sigma}$ (this perturbation is smaller than ε by choice of x); and do not perturb the right-hand side of the constraint. Then $\sigma \bullet \dot{\sigma} \models \phi^*$ is equivalent to $x \bullet \{\dot{a} \mapsto [\![t]\!](x)\} \models \phi$, that is, $x \bullet \{\dot{a} \mapsto [\![t]\!](x)\} \models \dot{a} = t$ which holds according to the definition of $\models$.

In the case where ϕ is of the form $\texttt{mode} = m$, the lemma easily holds by choosing ϕ^* to be equal to ϕ, in which case $d(\phi, \phi^*) = 0$. The other base cases are similar to the case $\dot{a} = t$ above.

For considering general constraints with conjunction and disjunction, we proceed by induction. This easily goes through by choosing the minimum of the δ for the different atomic constraints and combining the ϕ^* for the different branches. □

Using Lemma 1 we can bound the over-approximation of the algorithm up to arbitrary precision.

Theorem 4. *For every hybrid system description D, for all $\varepsilon > 0$ there is a $\delta > 0$ such that $[\![A(D, \delta)]\!]$ is an ε-perturbed instance of D.*

Proof. Let $\delta_{\phi,B,\varepsilon}$ be the value of δ, as ensured by Lemma 1 for the constraint ϕ, the box B, and ε. Let $\varepsilon > 0$ be arbitrary, but fixed. Choose δ as the minimum of $\delta_{\phi,B,\varepsilon}$ over all constraints ϕ defining D, and boxes B forming the state space (one box for each mode).

We assume that D is of the form (S, *Init*, *Flow*, *Jump*, *Unsafe*), and $[\![A(D,\delta)]\!]$ is of the form $(S, [\![Init_R]\!], [\![Flow_R]\!], [\![Jump_R]\!], [\![Unsafe_R]\!])$. To prove that $[\![A(D,\delta)]\!]$ is an ε-perturbed instance of D we have to prove the corresponding result for each pair of corresponding constraints of D and $A(D,\delta)$. Here, in each case, Theorem 3 implies the second item of Definition 5. Hence it suffice to prove the first item for each pair of corresponding constraints:

- To prove that $[\![Init_R]\!]$ is a ε-perturbed instance of *Init*, we have to prove that for every $\sigma \in [\![Init_R]\!]$, there is a constraint $Init^*$ with $d(Init, Init^*) \le \varepsilon$ such that $\sigma \models Init^*$. Let σ be an arbitrary, but fixed element of $[\![Init_R]\!]$. Then σ satisfies at least one disjunct of $Init_R$. Let B be the mode/box pair generating this disjunct. Then $\sigma \in B$, $B \models_I Init$ and $diam(B) \le \delta_{Init,S,\varepsilon} \le \delta$, where S is the box forming the state space of the mode of σ. Then, by Lemma 1, there is a constraint $Init^*$ with $d(Init, Init^*) \le \varepsilon$, $\sigma \models Init^*$.
- Flow: To prove that $[\![Flow_R]\!]$ is a ε-perturbed instance of *Flow*, we have to prove that for every $\sigma \bullet \dot{\sigma} \in [\![Flow_R]\!]$, there is a constraint $Flow^*$ with $d(Flow, Flow^*) \le \varepsilon$ such that $\sigma \bullet \dot{\sigma} \models Flow^*$. Let $\sigma \bullet \dot{\sigma}$ be an arbitrary, but fixed element of $[\![Flow_R]\!]$. Then $\sigma \bullet \dot{\sigma}$ satisfies at least one disjunct of $Flow_R$. Let B the mode/box pair generating this disjunct. Hence $\sigma \in B$, $\dot{\sigma} \in (B \models_I Flow)$ and $diam(B) \le \delta_{Flow,S,\varepsilon} \le \delta$, where S is the box forming the state space of the mode of σ. Then, by Lemma 1, there is a constraint $Flow^*$ such that $d(Flow, Flow^*) \le \varepsilon$, and $\sigma \bullet \dot{\sigma} \models Flow^*$.
- *Jump* and *Unsafe*: analogous to *Init* □

The hybrid system $A(H,\delta)$ has a very simple form that is equivalent to a rectangular automaton. Still, this rectangular automaton is not necessarily initialized and hence it belongs to an undecidable class [8]. However, after explicitly solving the flow constraints, it can be completely defined by polynomials. Moreover, it has a bounded state space. Hence one can apply a result by Fränzle [6] which provides an algorithm that, while it does not terminate always, still terminates for all *robust* inputs. Hence we have:

Theorem 5. *Safety verification of the results of $A(H,\delta)$ is quasi-decidable.*

However, it is possible that $A(H,\delta)$ is not robust—even if H is robust. In the case of such non-robustness Fränzle's algorithm does not terminate. This can be circumvented, resulting in the following proof of the main theorem of this paper (i.e., Theorem 1):

Proof. We use the following algorithm:

```
$i \leftarrow 1$
while there is no $j \in \{1,\dots,i\}$ such that $F_{2^i}(A(H, 1/2^j))$
    $i \leftarrow i+1$
return true
```

Here F_t is a version of Fränzle's algorithm [6] for safety verification that, if it terminates within t time units, it return the corresponding (Boolean) result, and otherwise returns false. The above algorithm obviously is correct. It remains to prove termination for robustly safe H.

Due to Theorem 4, if H is robustly safe, then there is a strictly positive real number δ such that also $[\![A(H, \delta)]\!]$ is robustly safe. Moreover, due to the nature of Definition 5, also for all $\delta' < \delta$, $[\![A(H, \delta')]\!]$ is robustly safe. Hence we can choose n such that $[\![A(H, 1/2^n)]\!]$ is robustly safe. Assume that Fränzle's algorithm (that terminates for all robustly safe inputs) needs time t to prove safety of $[\![A(H, 1/2^n)]\!]$. Eventually the above algorithm will start $F_{2^i}(A(H, 1/2^n))$ with 2^i being greater than t which will prove safety. $\square$

Note that in the case where the constraint defining the hybrid system contain transcendental function symbols like sin, or exp, the test $\models_I$ cannot be implemented exactly. However, by using conservatively rounded finite arithmetic of a suitable precision, one can retain the above results. Our experience on related algorithms shows that for practical problems, one can even expect hardware floating point computation (again conservatively rounded for soundness) to suffice.

References

1. Asarin, E., Bouajjani, A.: Perturbed Turing machines and hybrid systems. In: Proc. LICS 2001, pp. 269–278 (2001)
2. Bournez, O., Campagnolo, M.L.: A survey on continuous time computations. In: New Computational Paradigms, pp. 383–423 (2008)
3. Collins, P.: Continuity and computability of reachable sets. Theoretical Computer Science 341, 162–195 (2005)
4. Collins, P.: Semantics and computability of the evolution of hybrid systems. Research Report MAS-R0801, CWI (2008)
5. Damm, W., Pinto, G., Ratschan, S.: Guaranteed termination in the verification of LTL properties of non-linear robust discrete time hybrid systems. International Journal of Foundations of Computer Science (IJFCS) 18(1), 63–86 (2007)
6. Fränzle, M.: Analysis of hybrid systems: An ounce of realism can save an infinity of states. In: Flum, J., Rodríguez-Artalejo, M. (eds.) CSL 1999. LNCS, vol. 1683, pp. 126–140. Springer, Heidelberg (1999)
7. Henzinger, T.A., Ho, P.-H., Wong-Toi, H.: Algorithmic analysis of nonlinear hybrid systems. IEEE Transactions on Automatic Control 43, 540–554 (1998)
8. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata. Journal of Computer and System Sciences 57, 94–124 (1998)
9. Henzinger, T.A., Raskin, J.-F.: Robust undecidability of timed and hybrid systems. In: Lynch, N., Krogh, B. (eds.) HSCC 2000. LNCS, vol. 1790, p. 145. Springer, Heidelberg (2000)
10. Moore, R.E.: Interval Analysis. Prentice Hall, Englewood Cliffs (1966)
11. Neumaier, A.: Interval Methods for Systems of Equations. Cambridge Univ. Press, Cambridge (1990)
12. Puri, A., Borkar, V., Varaiya, P.: ε-approximation of differential inclusions. In: Alur, R., Henzinger, T.A., Sontag, E.D. (eds.) HS 1995. LNCS, vol. 1066. Springer, Heidelberg (1996)

Closed Rectangle-of-Influence Drawings for Irreducible Triangulations

Sadish Sadasivam and Huaming Zhang[⋆]

Computer Science Department
University of Alabama in Huntsville
Huntsville, AL, 35899, USA
sadish.sadasivam@gmail.com, hzhang@cs.uah.edu

Abstract. A *closed rectangle-of-influence (RI for short)* drawing is a straight-line grid drawing in which there is no other vertex inside or on the boundary of the axis parallel rectangle defined by the two end vertices of any edge. Biedl et al. [2] showed that a plane graph G has a closed RI drawing, if and only if it has no *filled 3-cycle* (a cycle of 3 vertices such that there is a vertex in the proper interior). They also showed that such a graph G has a closed RI drawing in an $(n-1) \times (n-1)$ grid, where n is the number of vertices in G. They raised an open question on whether this grid size bound can be improved [2]. Without loss of generality, we investigate maximal plane graphs admitting closed RI drawings in this paper. They are plane graphs with a quadrangular exterior face, triangular interior faces and no filled 3-cycles, known as *irreducible triangulations* [7]. In this paper, we present a linear time algorithm that computes closed RI drawings for irreducible triangulations. Given an arbitrary irreducible triangulation G with n vertices, our algorithm produces a closed RI drawing with size at most $(n-3) \times (n-3)$; and for a random irreducible triangulation, the expected grid size of the drawing is $(\frac{22n}{27} + O(\sqrt{n})) \times (\frac{22n}{27} + O(\sqrt{n}))$. We then prove that for arbitrary $n \geq 4$, there is an n-vertex irreducible triangulation, such that any of its closed RI drawing requires a grid of size $(n-3) \times (n-3)$. Thus the grid size of the drawing produced by our algorithm is tight. This lower bound also answers the open question posed in [2] negatively.

1 Introduction

A *planar graph* is a graph which can be drawn in a plane so that the edges do not intersect at any point other than their end vertices. A planar graph with a fixed planar embedding is called a *plane graph*. A plane graph divides the plane into regions called *faces*. The unbounded region is called *exterior face*, the other faces are called *interior faces*. An edge is an *exterior edge* if it belongs to the exterior face, otherwise it is an *interior edge*. A vertex is an *exterior vertex* if it is an endpoint of an exterior edge, otherwise it is an *interior vertex*. A 3-cycle of a plane graph is a *filled 3-cycle* if the proper interior of the 3-cycle contains a vertex. A plane graph G is an *irreducible triangulation*, if G has a quadrangular exterior face, triangular interior faces and no filled 3-cycles.

[⋆] Corresponding author. His research is supported by NSF grant CCF-0728830.

© Springer-Verlag Berlin Heidelberg 2010

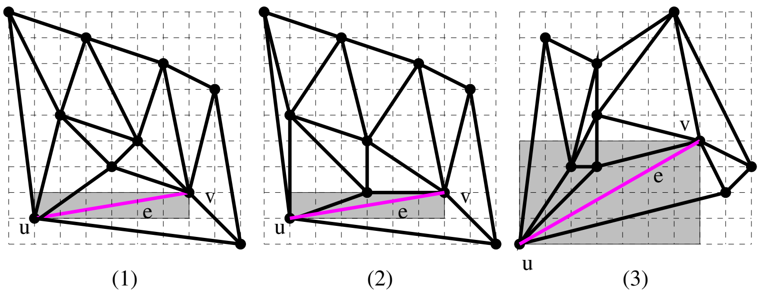


Fig. 1. (1) A closed RI drawing, (2) an open RI drawing, (3) a non RI drawing, of an inner triangulated plane graph with no filled 3-cycles

A *straight-line planar* drawing of a planar graph G is a drawing in which all the vertices of the graph are represented by points and all edges are represented by straight-line segments without intersections except at their common ends. A *straight-line grid drawing* is a straight-line planar drawing in which every vertex is placed on a grid point with integer coordinates. A *rectangle-of-influence (RI for short) drawing* is a straight-line grid drawing in which the axis parallel rectangle defined by the two vertices of any edge does not contain any other vertex [8]. The classes of graphs that admit RI drawings are investigated in [11]. A RI is *closed* if it includes the boundary of the axis parallel rectangle defined by the two vertices of an edge and *open* if it does not include the boundary (see Fig. 1). In [2], Biedl et al. showed that a planar graph has a closed RI drawing, if and only if it has no filled 3-cycles and presented a linear time algorithm to construct a closed RI drawing in a grid of size $(n-1) \times (n-1)$.

Applying the above necessary and sufficient conditions for the existence of closed RI drawings in [2], it is easy to see that the maximal plane graphs admitting closed RI drawings are irreducible triangulations. Without loss of generality, we only consider irreducible triangulations in this paper. We present a linear time algorithm that computes a closed RI drawing of an arbitrary irreducible triangulation G with n vertices. Its grid size is at most $(n-3) \times (n-3)$. For a random irreducible triangulation, the expected grid size is $(\frac{22n}{27} + O(\sqrt{n})) \times (\frac{22n}{27} + O(\sqrt{n}))$. We then prove that the size of the drawing produced by our algorithm is tight, by showing that a lower bound on the grid size for a closed RI drawing for an arbitrary irreducible triangulation is $(n-3) \times (n-3)$. This lower bound also answers the open question posed in [2] negatively.

The reminder of the paper is organized as follows. In Section 2, we introduce some preliminaries. In Section 3, we introduce our algorithm that computes a closed RI drawing for an irreducible triangulation. We also present a lower bound on closed RI drawings for an arbitrary graph.

2 Preliminaries

A graph $G = (V, E)$ is called a *directed graph* (digraph for short) if each edge of G is assigned a direction. An *orientation* of a graph G assigns a direction to every edge of G. An orientation is said to be *acyclic* if it does not contain any directed cycle in the graph. A *st-orientation* [7] (also known as *bipolar orientation* or *st-numbering* [4]) is an acyclic orientation with a unique source s and a unique sink t. The properties of st-orientations have been extensively studied [10,12,3]. For a 2-connected plane graph G with st-orientation, we have for any vertex $v \in V(G)$ where $v \neq s, t$, the edges incident to v are partitioned into a non-empty consecutive block of incoming edges and a non-empty consecutive block of outgoing edges around v. Each face f of G has two vertices s_f and t_f, known as the *source vertex* and *sink vertex* of f respectively, such that the boundary of the face f consists of two non-empty directed paths both originating at s_f and ending at t_f. Going from s_f to t_f, the path on the left (right, respectively) of f is defined as the *left-lateral path* of f (*right-lateral path* of f, respectively).

Let G be an irreducible triangulation. Let W, S, E and N be four exterior vertices in counterclockwise order. A *transversal structure* (also known as *regular edge labeling* [9]) $\mathcal{T}(G)$ of G assigns a direction to each interior edge of G and partitions the set of all the interior edges of G into two groups, say red and blue, such that the following conditions are satisfied:

- Interior vertices: In clockwise order around each interior vertex v, its incident edges form a non empty interval of red edges entering v, a non empty interval of blue edges entering v, a non empty interval of red edges leaving v and a non empty interval of blue edges leaving v.
- Exterior vertices: All interior edges incident to S are red edges leaving S, all interior edges incident to N are red edges entering N, all interior edges incident to W are blue edges leaving W and all interior edges incident to E are blue edges entering E. Each of these blocks is non empty.

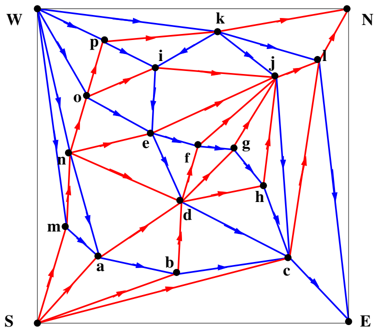


Fig. 2. The minimal transversal structure for the graph G

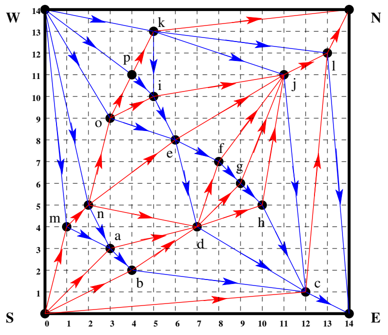


Fig. 3. The closed RI drawing of G generated by Algorithm 1 using the minimal transversal structure in Fig. 2

Given an irreducible triangulation G endowed with a transversal structure, Fusy [7,6] defined an *alternating 4-cycle* as a cycle $\mathcal{C} = (e_1, e_2, e_3, e_4)$ of 4 edges of G that are color alternating (i.e., two adjacent edges of $\mathcal{C}$ have different colors). Given a vertex v

on $\mathcal{C}$, Fusy defined the *left-edge* (*right-edge*, respectively) of v as the edge of $\mathcal{C}$ starting from v in clockwise direction (counterclockwise direction, respectively) along the cycle $\mathcal{C}$. Fusy [7] observed that an alternating 4-cycle $\mathcal{C}$ in a transversal structure satisfies either of the following two configurations: (1) All edges inside $\mathcal{C}$ and incident to a vertex v of $\mathcal{C}$ have the color of the left-edge of v. Then $\mathcal{C}$ is called a *left alternating 4-cycle*; (2) All edges inside $\mathcal{C}$ and incident to a vertex v of $\mathcal{C}$ have the color of the right-edge of v. Then $\mathcal{C}$ is called a *right alternating 4-cycle*.

The *minimal transversal structure* $\mathcal{T}(G)$ of an irreducible triangulation G is the unique transversal structure with no right alternating 4-cycle [7]. For example, consider the transversal structure $\mathcal{T}(G)$ in Fig. 2. Consider the vertex n in the cycle $\mathcal{C}_1 = \{(n,e),(e,d),(d,a),(a,n)\}$. Its left-edge is (n,e), while its right-edge is (n,a). The edge inside $\mathcal{C}_1$ and incident to n is (n,d), which has the same color as the left-edge of n. The edge (n,d) also has the same color as the left-edge (d,a) of d. Hence, $\mathcal{C}_1$ is a left alternating 4-cycle. Similarly, one can easily verify that $\mathcal{C}_2 = \{(p,k),(k,j),(j,i),(i,p)\}$ is also a left alternating 4-cycle. There is no right alternating 4-cycle. Hence, the transversal structure is the minimum transversal structure of G.

3 Algorithm for Closed RI Drawing

Consider an irreducible triangulation G with a transversal structure $\mathcal{T}(G)$. The subgraph of G induced by all red edges (blue edges, respectively) and the four exterior edges is called the *red map* of G (*blue map* of G, respectively), it is denoted by G_r (G_b, respectively). For an interior vertex v in G_r (G_b, respectively), the edges entering v and the edges leaving v form two non empty contiguous blocks. When walking from u to v in G_r (G_b, respectively), if we always pick the first feasible outgoing edge (i.e., the outgoing edge could still lead to the vertex v) in counterclockwise direction at all its intermediate vertices in the path, then the path is unique and it is called the *rightmost* path from u to v [7]. Similarly, the *leftmost* path always chooses the first feasible clockwise outgoing edge at all its intermediate vertices in the path [7].

In particular, we use $P_r^{in}(v)$ to denote the rightmost path from S to v and $P_r^{out}(v)$ to denote the leftmost path from v to N in G_r. We use $P_b^{in}(v)$ to denote the rightmost

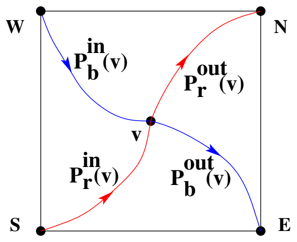


Fig. 4. The separating paths for a vertex v

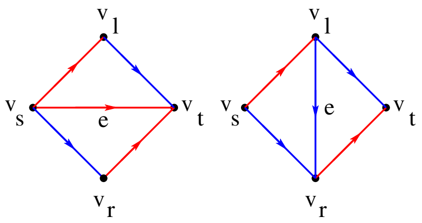


Fig. 5. The left alternating 4-cycle around a red free edge and blue free edge

path from W to v and $P_b^{out}(v)$ to denote the leftmost path from v to E in G_b. For any interior vertex v of G, let $P_r(v)$ denote the concatenated path of $P_r^{in}(v)$ and $P_r^{out}(v)$, it is called the *red separating path* for v. Similarly, let $P_b(v)$ denote the concatenated path of $P_b^{in}(v)$ and $P_b^{out}(v)$, it is called the *blue separating path* for v. Fig. 4 shows the red separating path and blue separating path of a vertex $v \in V(G)$. A red edge (blue edge, respectively) is defined as a *red separating edge* (*blue separating edge*, respectively) if it belongs to a red separating path (blue separating path, respectively) of a vertex $v \in V(G)$, otherwise we define it as a *red free edge* (*blue free edge*, respectively). We denote by $E_{free}(\mathcal{T}(G))$, the set of all the free edges in $\mathcal{T}(G)$. For example, for the transversal structure in Fig. 2, the red separating path for the vertex d is $P_r(d) = (S, b, d, f, j, l, N)$ and the blue separating path for the vertex d is $P_b(d) = (W, o, e, d, c, E)$. The edge (n, d) is a red free edge and the edge (k, i) is a blue free edge.

Let e be a free edge in a transversal structure $\mathcal{T}(G)$. Ignoring the direction of the edges, let $\mathcal{C}$ be the 4-cycle (v_s, v_r, v_t, v_l) that encloses the free edge e. This 4-cycle $\mathcal{C}$ has to be a left alternating 4-cycle of $\mathcal{T}(G)$ as per the color and orientation of the edges in $\mathcal{T}(G)$ [7]. See Fig. 5 for the two cases of the color and orientation of the edges around a free edge e. After deleting the free edge e enclosed in $\mathcal{C}$, the interior enclosed by $\mathcal{C}$ becomes a quadrangular face, which we denote by $f(e)$. It is called the *enclosing face* of the free edge e. Apparently, the mapping $e \rightarrow f(e)$ from the set of free edges to the set of their enclosing faces is a one-to-one and onto mapping. Consider the four vertices of an enclosing face $f(e)$ for the free edge e, it has a unique source, and a unique sink. We denote by $V_s(e)$ ($V_t(e)$, respectively) the unique source vertex (unique sink vertex, respectively) of $f(e)$. Consider the enclosing face $f(e)$, the vertex $\neq V_s(e), V_t(e)$ on the left-lateral path (right-lateral path, respectively) of $f(e)$ is defined as the left vertex (right vertex, respectively). We denote by $V_l(e)$ ($V_r(e)$, respectively) the left vertex (right vertex, respectively) of $f(e)$. Observe that, if e is a red free edge (blue free edge, respectively), then $e = (V_s(e), V_t(e))$ ($e = (V_l(e), V_r(e))$, respectively). See Fig. 5. It presents the two cases where e is either a red free edge or a blue free edge. The vertices v_s, v_t, v_l and v_r are the source vertex, sink vertex, left vertex and right vertex of the enclosing face $f(e)$ respectively.

We have the following lemma, its proof is omitted.

Lemma 1. *Consider a transversal structure $\mathcal{T}(G)$ of an irreducible triangulation G:*

1. *An st-orientation G_{rb} with S as the single source and N as the single sink can be constructed using $\mathcal{T}(G)$ (with all the edge directions in $\mathcal{T}(G)$) and assigning directions to the four exterior edges as S to E, S to W, E to N and W to N.*
2. *An st-orientation $G_{r\overline{b}}$ with S as the single source and N as the single sink can be constructed from G_{rb}, by simply* reversing *the direction of the blue edges, i.e., if (u, v) is a blue edge directed from u to v in G_{rb}, then that edge becomes (v, u) (i.e., directed from v to u) in $G_{r\overline{b}}$. The direction of the red edges remain as in G_{rb}.*
3. *An st-orientation G_x with S as the single source and N as the single sink can be constructed from G_{rb}, by deleting the free edges $E_{free}(\mathcal{T}(G))$ and the vertices W and E and the edges having W or E as one of their end points.*
4. *An st-orientation G_y with S as the single source and N as the single sink can be constructed from G_x, by simply* reversing *the direction of the blue edges.*

Next, we present our closed RI drawing algorithm for irreducible triangulations:

Algorithm 1. Closed RI drawing
Input: An irreducible triangulation $G = (V, E)$.
Output: A closed RI drawing for G.

1. *Construct a transversal structure $\mathcal{T}(G)$ of G.*
2. *Construct the following graphs:*
   (a) *Construct a graph G_x from $\mathcal{T}(G)$ by deleting the free edges $E_{free}(\mathcal{T}(G))$ and the vertices W and E and the edges having W or E as one of their end points.*
   (b) *Construct a graph G_y from G_x by simply* **reversing** *the direction of all the blue edges, i.e., if (u, v) is a blue edge directed from u to v in G_x, then that edge becomes (v, u) (i.e., directed from v to u) in G_y. The direction of the red edges remain as in G_x.*
3. *Compute $x(v)$ and $y(v)$ as follows:*
   x(v) = the length of a longest path from S to v in G_x
   y(v) = the length of a longest path from S to v in G_y
   Note that we have x(S) = y(S) = 0.
4. *For the vertices W and E we define:*
   x(W) = x(S), y(W) = y(N), x(E) = x(N), y(E) = y(S).
5. *Construct a closed RI drawing for G in the $xy - grid$ such that each vertex $v \in V(G)$ is located at coordinates $(x(v), y(v))$ and for each edge $(u, v) \in E(G)$, the vertices u and v are connected by a straight line in the drawing.*

Fig. 3 shows the closed RI drawing for the irreducible triangulation G, by applying Algorithm 1 on the minimum transversal structure $\mathcal{T}(G)$ in Fig. 2. In order to prove the correctness of Algorithm 1, we need to introduce several technical lemmas in the following. The proof of Lemma 2 is omitted.

Lemma 2

1. *If (u, v) is a red separating edge in G, then $x(u) < x(v)$ and $y(u) < y(v)$.*
2. *If (u, v) is a blue separating edge in G, then $x(u) < x(v)$ and $y(u) > y(v)$.*
3. *If (u, v) is a red free edge in G, then $x(v) - x(u) \geq 2$.*
4. *If (u, v) is a blue free edge in G, then $y(u) - y(v) \geq 2$.*

Lemma 3. *The drawing produced by Algorithm 1 satisfies the following:*

1. *Each vertex is placed on a grid point and each edge is drawn as a straight-line, with no edge crossings except possibly at a common vertex.*
2. *For the axis parallel rectangle formed by the end vertices of each edge, there are no other vertices (i.e., vertices other than the two end vertices of that edge) inside or on the boundary of the rectangle.*

Proof. We now show that Algorithm 1 produces a closed RI drawing for the input graph G. The proof is based on induction on k, $1 \leq k \leq (n-4)$, where the $(n-4)$ is because of $|V(G) - \{S, W, E, N\}|$. We denote by D_0 the partial drawing consisting of only the vertices S and W and the edge connecting S and W. Obviously, S and W are placed at coordinates $(x(S), y(S))$ and $(x(W), y(W))$ respectively. Let D_k denote the partial drawing after k more vertices are added to D_0. At each step, D_{k+1} is constructed from D_k by adding a vertex v_{k+1} at coordinates $(x(v_{k+1}), y(v_{k+1}))$ and all its incoming edges. This vertex v_{k+1} is chosen such that it has all its incoming edges from vertices in D_k, i.e., when v_{k+1} is added to D_k to generate D_{k+1}, all the vertices that have an outgoing edge to v_{k+1} must already be in the drawing D_k. Note that, at each step, there does exist such a vertex v_{k+1}, since we can use the topological sorting of the vertices from the st-orientation G_{rb}.

Base Case: Let v_1 be a vertex whose only two incoming edges are from S and W. Both the incoming edges to v_1 must be separating edges. We have $x(v_1) > x(S) = x(W)$ and $y(S) < y(v_1) < y(W)$. Thus, the drawing D_1 is a partial closed RI drawing.

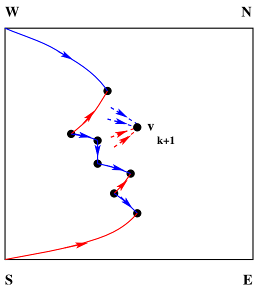


Fig. 6. Direction of incoming edges to v_{k+1} when it gets added to D_k

Induction Step: Assume that the algorithm produces a partial closed RI drawing D_k after adding k vertices. Fig. 6 shows the $(k+1)^{th}$ vertex and all its incoming edges getting added to D_k to generate D_{k+1}. Let R_{D_k} denote the closed region formed by D_k. For the region R_{D_k}, a red free edge directed from u to v cannot be an exterior edge, as the vertex v must have an incoming red separating edge and an incoming blue separating edge that enclose the red free edge. Thus, we can only have a red separating edge, blue separating edge or a blue free edge as an exterior edge. Thus, if we travel in a counterclockwise direction from S to N along the exterior edges in the boundary of D_k, the y-coordinates of the exterior vertices in D_k are in a strictly increasing order.

For every exterior vertex other than S and W in D_k, the two exterior edges adjacent to it can be *red incoming and blue incoming*, *red incoming and red outgoing*, *blue incoming and blue outgoing* or *red outgoing and blue outgoing*. Note that we cannot have the *red incoming and blue outgoing* (since the red incoming edge will be placed between a blue incoming edge and blue outgoing edge) or *blue incoming and red outgoing* (since the blue incoming edge will be placed between a red incoming edge and red outgoing edge) combination.

Due to the properties of the transversal structure, for a red (blue, respectively) exterior edge (u, v) in D_k, u and v cannot both have red (blue, respectively) outgoing edges to v_{k+1}. Fig. 7, 8, 9 and 10 show v_{k+1} getting added to D_k to generate D_{k+1}. Let $(v_1^r, v_2^r, ...v_p^r)$ be p exterior vertices on a path of red separating edges that have an outgoing blue separating edge to v_{k+1} and $(v_1^b, v_2^b, ...v_q^b)$ be q exterior vertices on a

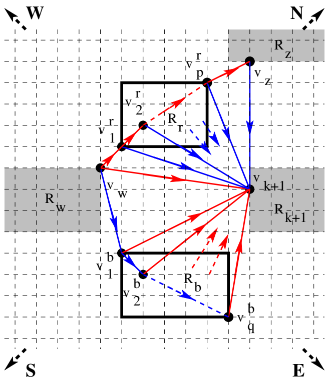


Fig. 7. Proof of Lemma 3: Case 1

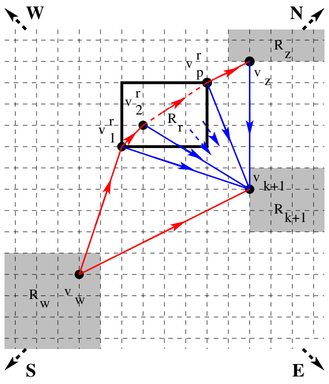


Fig. 8. Proof of Lemma 3: Case 2

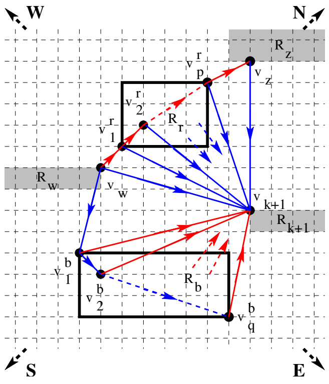


Fig. 9. Proof of Lemma 3: Case 3

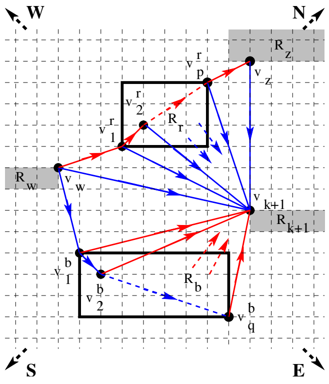


Fig. 10. Proof of Lemma 3: Case 4

path of blue separating edges that have an outgoing red separating edge to v_{k+1}, where $p, q \geq 0$. R_r (R_b, respectively) denotes the axis parallel rectangular region having the vertices v_1^r and v_p^r (v_1^b and v_q^b, respectively) on its boundary. The vertex v_w has a red separating edge to v_1^r and either a blue separating edge or a blue free edge to v_1^b. This vertex v_w has either a red free edge, red separating edge or blue separating edge to v_{k+1}. The vertex v_{k+1} may or may not have an incoming blue free edge (v_z, v_{k+1}). R_w, R_z and R_{k+1} represent the regions where v_w, v_z and v_{k+1} could possibly be placed by Algorithm 1 in the drawing respectively.

In the following cases we assume that v_{k+1} has an incoming blue free edge (v_z, v_{k+1}). The proof for the cases where v_{k+1} does not have an incoming blue free edge (v_z, v_{k+1}) is similar to the following cases and is omitted. For each case, we need to prove the following claim:

Claim 1:

1. There is no other vertex inside the closed RI of the edges (v_i^r, v_{k+1}), (v_j^b, v_{k+1}) or (v_z, v_{k+1}), where $1 \leq i \leq p$ and $1 \leq j \leq q$.

2. There is no other vertex inside the closed RI of the edge (v_w, v_{k+1}) since the closed RI of the edge (v_w, v_1^r) or the edge (v_w, v_1^b) does not contain any other vertex (by induction).
3. The vertex v_{k+1} is outside of the region R_{D_k} and is not in the closed RI of any of the edges of D_k.
4. There are no edge crossings except at a common vertex.

Case 1: (v_w, v_{k+1}) is a red free edge. Then (v_w, v_1^b) must be a blue separating edge. In this case we have $p \geq 1$ and $q \geq 1$. This case is illustrated in Fig. 7. From Lemma 2, we have: (1) $x(v_w) < x(v_1^r) < x(v_2^r) < ... < x(v_p^r) < x(v_z)$ and $x(v_p^r) < x(v_{k+1})$; (2) $y(v_w) < y(v_1^r) < y(v_2^r) < ... < y(v_p^r) < y(v_z)$ and $y(v_{k+1}) < y(v_1^r)$; (3) $x(v_1^b) < x(v_2^b) < ... < x(v_q^b) < x(v_{k+1})$ and $x(v_w) < x(v_1^b)$; (4) $y(v_q^b) < ... < y(v_2^b) < y(v_1^b) < y(v_{k+1})$ and $y(v_1^b) < y(v_w)$.

From the above inequalities, we have: (1) R_z is strictly above and strictly to the right of R_r; (2) R_w is strictly below R_r, strictly above R_b and strictly to the left of both R_r and R_b; and (3) R_{k+1} is strictly below R_r, strictly above R_b and strictly to the right of both R_r and R_b. Thus, Claim 1 is proved for this case.

Case 2: (v_w, v_{k+1}) is a red separating edge. In this case we have $p \geq 1$ and $q = 0$ (see Fig. 8). The proof of Case 1 covers this case as well, since we only have an added inequality that $y(v_w) < y(v_{k+1})$, i.e., R_w is strictly below R_{k+1}.

Case 3: (v_w, v_1^b) is a blue free edge. Then (v_w, v_{k+1}) must be a blue separating edge. In this case we have $p \geq 0$ and $q \geq 1$. This case is illustrated in Fig. 9. We have: (1) $x(v_w) < x(v_1^r) < x(v_2^r) < ... < x(v_p^r) < x(v_z)$ and $x(v_p^r) < x(v_{k+1})$; (2) $y(v_w) < y(v_1^r) < y(v_2^r) < ... < y(v_p^r) < y(v_z)$ and $y(v_{k+1}) < y(v_w)$; (3) $x(v_1^b) < x(v_2^b) < ... < x(v_q^b) < x(v_{k+1})$; and (4) $y(v_q^b) < ... < y(v_2^b) < y(v_1^b) < y(v_{k+1})$.

From the above inequalities, we have: (1) R_z is strictly above and strictly to the right of R_r; (2) R_w is strictly below R_r, strictly above R_{k+1} and strictly to the left of R_r; and (3) R_{k+1} is strictly above R_b, strictly below R_w and strictly to the right of both R_r and R_b. Thus, Claim 1 is proved for this case.

Case 4: Both (v_w, v_{k+1}) and (v_w, v_1^b) are blue separating edges. In this case we have $p \geq 0$ and $q \geq 1$ (see Fig. 10). The proof of Case 3 covers this case as well, since we only have an added inequality that $x(v_w) < x(v_1^b)$, i.e., R_w is strictly to the left of R_b.

Next, we present our main theorem, its proof is omitted.

Theorem 1

1. *Let G be an irreducible triangulation. Algorithm 1 produces a closed RI drawing for G in linear time, with size bounded by $(n-3) \times (n-3)$.*
2. *Let G be an irreducible triangulation with n vertices taken uniformly at random. Then G has a closed RI drawing obtainable in linear time whose expected grid size is $(\frac{22n}{27} + O(\sqrt{n})) \times (\frac{22n}{27} + O(\sqrt{n}))$.*
3. *There exists an irreducible triangulation G with n vertices, whose closed RI drawing requires a grid size of at least $(n-3) \times (n-3)$.*

Fig. 2 shows the minimal transversal structure for the graph G and Fig. 3 shows the closed RI drawing generated with this minimal transversal structure by Algorithm 1. Figure 11 present a graph G whose closed RI drawing requires a grid size of at least $(n-3) \times (n-3)$.

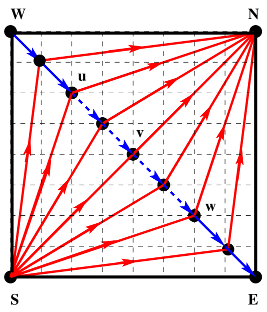


Fig. 11. Lower bound for a closed RI drawing

References

1. Barrire, L., Huemer, C.: 4-labelings and grid embeddings of plane quadrangulations. In: Gansner, E.R. (ed.) GD 2009. LNCS, vol. 5849, pp. 413–414. Springer, Heidelberg (2010)
2. Biedl, T.C., Bretscher, A., Meijer, H.: Rectangle of influence drawings of graphs without filled 3-cycles. In: Kratochvíl, J. (ed.) GD 1999. LNCS, vol. 1731, pp. 359–368. Springer, Heidelberg (1999)
3. de Mendez, P.O.: Orientations bipolaires. PhD thesis, cole des hautes tudes en sciences sociales (1994)
4. Even, S., Tarjan, R.E.: Computing an st-numbering. Theoretical Computer Science 2(3), 339–344 (1976)
5. Felsner, S., Huemer, C., Kappes, S., Orden, D.: Binary labelings for plane quadrangulations and their relatives (April 12, 2006)
6. Fusy, E.: Combinatoire des cartes planaires et applications algorithmiques. PhD thesis, Ecole Polytechnique (2007)
7. Fusy, E.: Transversal structures on triangulations: A combinatorial study and straight-line drawings. Discrete Mathematics 309(7), 1870–1894 (2009)
8. Ichino, M., Sklansky, J.: The relative neighborhood graph for mixed feature variables. Pattern Recognition 18(2), 161–167 (1985)
9. Kant, G., He, X.: Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. Theoretical Computer Science 172(1-2), 175–193 (1997)
10. Lempel, A., Even, S., Cederbaum, I.: An algorithm for planarity testing of graphs. In: Theory of Graphs: International Symposium, pp. 215–232 (1967)
11. Liotta, G., Lubiw, A., Meijer, H., Whitesides, S.H.: The rectangle of influence drawability problem. Computational Geometry: Theory and Applications 10(1), 1–22 (1998)
12. Rosenstiehl, P., Tarjan, R.E.: Rectilinear planar layouts and bipolar orientations of planar graphs. Discrete and Computational Geometry 1(1), 343–353 (1986)
13. Zhang, H., Vaidya, M.: On open rectangle-of-influence drawings of planar graphs. In: Du, D.-Z., Hu, X., Pardalos, P.M. (eds.) COCOA 2009. LNCS, vol. 5573, pp. 123–134. Springer, Heidelberg (2009)

Recovering Social Networks from Contagion Information

Sucheta Soundarajan[*] and John E. Hopcroft[*]

Dept. of Computer Science, Cornell University
{sucheta,jeh}@cs.cornell.edu

Abstract. Many algorithms for analyzing social networks assume that the structure of the network is known, but this is not always a reasonable assumption. We wish to reconstruct an underlying network given data about how some property, such as disease, has spread through the network. Properties may spread through a network in different ways: for instance, an individual may learn information as soon as one of his neighbors has learned that information, but political beliefs may follow a different type of model. We create algorithms for discovering underlying networks that would give rise to the diffusion in these models.

Keywords: Social Networks, Diffusion, Contagion, Graph Algorithms.

1 Introduction

The area of social network analysis raises many interesting and important questions, like determining which members of a population to vaccinate to stem the spread of a virus[5]. Such questions assume that the structure of the underlying network is known. This assumption is often valid, but in some cases, such as for criminal networks, it is less plausible. However, even in such cases, we may have information about the spread of something through the network. For instance, if authorities know that some new drug first appeared in City X, and then in Cities Y and Z, and so on, can we use this knowledge to recreate the underlying network of drug dealers? The general structure of criminal networks has been studied[6][4], but algorithms for analyzing social networks are often sensitive to the exact, as opposed to general, structure of the network[2]. Algorithms to help authorities recreate such networks would help them greatly.

In this paper, we investigate the situation in which some property is spreading through a population and we know when each individual adopts the property as well how the property spreads. For instance, information spread may follow the rule that an individual learns information as soon as one of his neighbors learns that information, but political belief might follow the rule that an individual adopts a belief after some proportion of his neighbors have adopted that belief[2]. First, we briefly describe an algorithm for constructing a graph under

[*] Supported by Air Force Office of Scientific Research Grant FA9550-09-1-0675 AFOSR.

© Springer-Verlag Berlin Heidelberg 2010

the model of information spread. After that, we consider the case when a vertex adopts a property after half of its neighbors adopt the property. The model in which every vertex adopts a property after half of its neighbors have adopted it is applicable to many situations[3]. This is a reasonable threshold whenever individuals desire to be in the majority. For example, an individual may decide to attend a social event after a majority of his friends decide to attend that event. There are certainly more complex cases, and we believe that algorithms for such cases will be based upon the work in this paper.

2 The General Problem

We wish to reconstruct a network given information about how a property has spread through it, and the time at which each vertex adopted that property.

2.1 Terminology and Problem Statement

A *model of contagion* describes how a property spreads. We consider the case of one property spreading through the network, and assume that vertices cannot 'unadopt' that property. A simple model of contagion, corresponding to the spread of information, is: "A vertex adopts the property in the time interval after at least one of its neighbors adopts the property."

A *time vector* is a vector of positive integers whose ith element is the time at which vertex i adopts the property. The smallest integer in the vector is 1 (these vertices are the first to adopt the property) and the vector contains all integers up to its maximum value. In other words, no times are 'skipped.'

If vertex x adopts the property at time t, then x is in *level* t. In this paper, we depict vertices sorted into levels so that vertices in level 1 are at the top, and vertices in the last level are at the bottom (Fig. 1).

An edge (w, x) is *incoming* to x if w is in a level prior to x, and *outgoing* from x otherwise. Despite this terminology, edges in this graph are undirected. Although the graph itself is not directed, we use such terminology because we can treat the property as 'flowing' from one vertex to another in some direction.

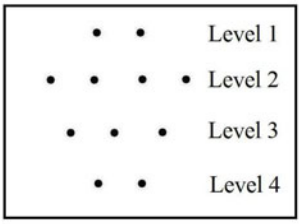


Fig. 1. Vertex Levels

For a given model of contagion, a graph G *solves* a time vector if, when we introduce the property to vertices in level 1, and the property spreads in accordance with the model of contagion, all other vertices adopt the property at the appropriate time. Given a set of vertices, a time vector, and a model of contagion, we wish to find a graph G that solves the time vector.

3 Algorithm for a Simple Model of Contagion

To help develop intuition about this problem, first consider a model of contagion in which a vertex adopts a property immediately after a neighbor adopts the property. Creating a graph to solve a time vector for this model of contagion is easy- begin by connecting every level 2 vertex to some level 1 vertex, and then connect every level 3 vertex to some level 2 vertex, and so on. Note that there are multiple ways to do this.

4 Algorithms for a Proportional Model of Contagion

Now consider the model where a vertex adopts the property after some proportion p of its neighbors adopt the property. The algorithms here are for $p = \frac{1}{2}$.

For G to solve a time vector under this model of contagion, every vertex in level 2 and later must have at least one edge from the previous level. Call this requirement the *recency condition,* and an edge fulfilling this requirement a *recency edge.* Also, a vertex in level 2 or later cannot have more outgoing than incoming edges; if it did, then less than half of its neighbors would be in a level earlier than it, but for the vertex to adopt the property at the correct time, half of its neighbors must have already adopted the property by that time. Call this requirement the *balance condition.* Vertices in level 1 can have arbitrarily many outgoing edges, since we assume that they always adopt the property at time 1. A vertex is *satisfied* if it meets both the balance and recency conditions.

4.1 Problem Intuition

We will satisfy each vertex by giving it a recency edge while ensuring that the balance condition is not violated. Consider the vertices in Fig. 2a. The level 2 vertices need a recency edge from a level 1 vertex, so we begin by adding two

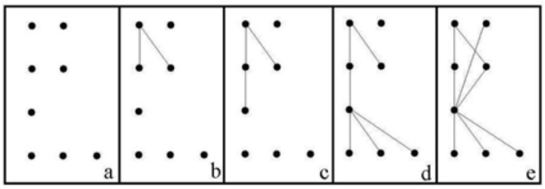


Fig. 2. Finding a Solution Graph for the Proportional Model of Contagion

such edges (Fig. 2b). Note that there are other possible choices. Next, the level 3 vertex gets a recency edge from a level 2 vertex (Fig. 2c). The first vertex in level 2 now has one outgoing edge, but it also has one incoming edge, so the balance condition is not violated. Next, the three vertices in level 4 must obtain recency edges from the vertex in level 3 (Fig. 2d). The vertex in level 3 only has one incoming edge, but three outgoing edges, so the balance condition is violated. To fix this, the level 3 vertex needs two additional incoming edges (Fig. 2e).

4.2 A Flow-Based Algorithm

The balance condition requirement resembles a flow problem, where outgoing flow is no more than incoming flow, so we will use a flow network[1] to determine which edges to put in the graph. Consider what a solution graph looks like. Vertices in levels 2 or later need a recency edge from a vertex in the previous level. If a vertex has outgoing edges, then it needs the same number of incoming edges. In our flow network, most edges will have a maximum capacity of 1, and all capacities will be integers. Thus, we will be able to find an integral flow. If an edge has a flow of 1 in this network, then we will add that edge to the solution graph. If it has a flow of 0, we will not.

The flow network contains all edges except those between vertices in the same level or vertices in adjacent levels. The edges are directed from earlier levels to later levels and have a capacity of 1. Edges between vertices in the same level are not a necessary part of a minimal solution, and so are not included in the flow network. Edges between vertices in adjacent levels are potential recency edges and must be handled in a special way, so we create a special structure representing these edges. See Fig. 3a. The ‘unended’ edges at level 4 originate at level 1, and the dashed box represents the special structure.

To construct the special structure between vertices in adjacent levels, add a row of vertices between each pair of adjacent levels k and $k + 1$. In this intermediate row, there is one vertex for each vertex in level $k + 1$. There will be an edge from each vertex in level k to each vertex in the intermediate row, each with a maximum capacity of 1. There will be an edge from each vertex in the intermediate row to the corresponding vertex in level $k + 1$. These edges have a *minimum* capacity of 1 and unlimited maximum capacity. See Fig. 3b. Because

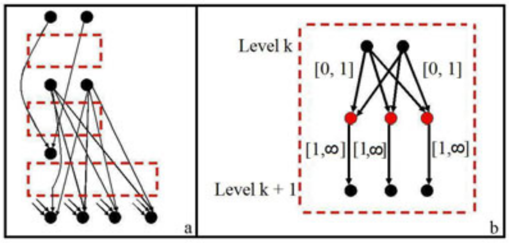


Fig. 3. Constructing the Flow Network

these edges have a minimum capacity of 1, each vertex in the intermediate row must have an edge from at least one vertex in level k. This ensures that every vertex in level $k+1$ has a recency edge from level k.

The vertices in level 1 serve as sources of unlimited flow. We will also need a sink vertex, because some vertices may have one incoming recency edge and no outgoing edges. Thus, each vertex in levels 2 or later should have an edge to the sink vertex with capacity 1.

Find a satisfying flow over this network. If an edge has a flow of 1, we add it to the solution graph.

4.3 Generalizing the Flow Based Algorithm

This algorithm works for the case $p = \frac{1}{2}$, where flow in is equal to flow out, but what about for other values of p? A natural case to consider is $p = \frac{2}{3}$, where a vertex adopts the property once $\frac{2}{3}$ of its neighbors have adopted the property. In other words, for every outgoing edge, there must be two incoming edges. Unfortunately, the flow-based algorithm cannot be generalized to this case. Consider the general problem of finding a valid integer flow over an arbitrary network with multiple sinks and sources, where the flow out of a vertex is the floor of half the flow into the vertex (i.e., for every unit of outgoing flow there are two units of incoming flow). With such an algorithm, we could use the algorithm given in the previous section to solve the problem for $p = \frac{2}{3}$: simply use the same network as described above, and then find a satisfying flow for the case when every unit of outgoing flow must be supported by two units of incoming flow. Then each vertex would have two incoming edges for every outgoing edge. Unfortunately, this general flow problem is NP-complete. (Note that this does not necessarily imply that our original problem is NP-complete).

Theorem 1. *The problem of finding a satisfying integer flow over an arbitrary network with multiple sinks and sources, where the flow out of a vertex is the floor of half the flow into the vertex, is NP-complete.*

Proof. We will reduce 3SAT to this problem. Fig. 4a represents one variable and Fig. 4b represents one clause. There is one sink vertex for each variable and one sink vertex for each clause. Each sink vertex has a demand of 1. The thick black lines carry flow from the source. In this flow network, there will be one source vertex for the entire network (not shown).

Consider Fig. 4a, corresponding to one variable, first. There is one version of this structure for each variable in the 3SAT formula, so, for instance, the flow network will contain multiple instances of A and B. Sink vertex B, on the right side, has a demand of 1. Vertex A, on the left side, has an incoming flow of 2 from a source vertex and thus an outgoing flow of 1. Vertex A can send this outgoing flow across either the top row or bottom row of solid black vertices. In this example, the flow was sent across the top row. This corresponds to setting the variable to FALSE. In order for the flow across the top row to reach vertex B, each solid black vertex in the top row must receive one additional unit of flow. The vertices with a cross through the middle send out one unit of flow. This unit of flow can go to either a solid black vertex or to a clause vertex. In this case, since the top row of solid black vertices needs supplementary flow at each vertex, the clause vertices at the top receive no flow from this structure. The clause vertices at the bottom can each receive one unit of flow. In this example, there are 5 clauses containing the variable as TRUE and 3 clauses containing the variable as FALSE.

Now consider Fig. 4b, corresponding to one clause. There is one version of this structure for each clause in the 3SAT formula, so the final flow network will contain multiple instances of vertex C. One instance of Vertex C is shown in Fig. 4b, but observe that Fig. 4a contains multiple instances of vertex C along the top and bottom, corresponding to various different clauses. Vertex C in Fig. 4b has an edge incoming from the source and three edges incoming from either the top row or the bottom row of the structures representing the three variables contained in the clause. Vertex C must receive 2 units of flow to send 1 unit to the sink vertex. Vertex C receives 1 unit of flow from the source, so it must also get 1 more unit of flow from one of the variable structures. There is a satisfying flow for this problem if and only there is a solution for the 3SAT problem. □

The general flow problem is NP-complete, but it is possible that we do not need the full power of a flow algorithm to solve our problem. Following is a different algorithm for the case $p = \frac{1}{2}$, in which some of the details are more explicit.

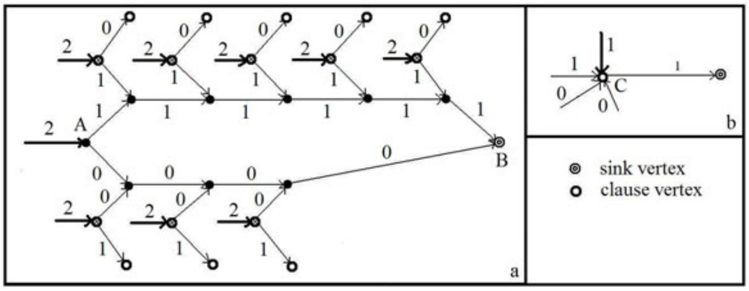


Fig. 4. Reducing 3SAT to a Flow Problem

4.4 Simplified Problem

We will simplify the original problem, and then show that the algorithm for solving this simplified problem also solves the original problem.

Given a set of vertices and a time vector, create a graph G over the vertices such that every vertex x in levels 2 or later has at least one neighbor in the previous level and at least half of x's edges are incoming. In the following algorithm and proof, the term *solves* will refer to this simplified problem. A graph solving this problem may not solve the original problem; although a vertex has at least half of its edges incoming, if too many edges are incoming from much earlier levels, it may still adopt the property too early.

4.5 A Backtracking Algorithm

In this algorithm we will construct a graph to solve the simplified problem. We proceed through the vertices in order, adding recency edges to each vertex. If the vertex being asked to provide the outgoing recency edge does not have sufficiently many incoming edges to send out another outgoing edge, then we attempt to modify the graph so that we can safely add the recency edge. Section 4.7 describes an implementation of this algorithm.

Construct a graph G by first connecting each vertex in level 2 to a vertex in level 1. Then for each vertex v in level 3 or later find a path from v which goes through a vertex in the previous level and then back through earlier and earlier levels until it reaches a vertex with more incoming edges than outgoing edges or level 1, such that no edge in the path is already in G. Call such a path a *satisfying sequence.* This sequence represents adding an edge to v from a vertex in the previous level, but that vertex must now find a new incoming edge to allow the addition of the outgoing edge to v, and so on until a vertex with an excess of incoming edges is found. If a satisfying sequence can be found for v, then adding these edges to G will satisfy v while still leaving all previous vertices with at least as many incoming edges as outgoing edges.

Perhaps no such sequence can be found unless G is modified. For example, suppose we find a sequence from v back to some vertex w, but then we are unable to proceed further because w is unable to obtain any more incoming edges. Because w is in a level prior to v, w has already been satisfied and so there must be some edge incoming to w. This edge is currently being used to support some other satisfying sequence going through w and eventually on to some later vertex x. If x can find a different satisfying sequence which does not use w, then the edge incoming to w can be used to support a satisfying sequence to v.

More precisely, the algorithm must find a path $(v, u_k), (u_k, u_{k-1}), ..., (u_2, u_1)$ such that:

1. u_k is in the level immediately before v.
2. For all $i < k$, either:
   2a. u_i is in a level previous to u_{i+1} and (u_i, u_{i+1}) does not exist in G, or
   2b. u_i is in a level after u_{i+1} and (u_i, u_{i+1}) does exist in G.
3. If (u_i, u_{i+1}) exists in G and is the only recency edge for u_i, then u_{i-1} is in the level immediately before u_i.
4. If u_1 is in a level previous to u_2, then u_1 is either in level 1 or has more incoming than outgoing edges in G. If u_1 is in a level after u_2, then u_1 has more incoming edges than outgoing edges in G and the edge (u_1, u_2) was not necessary to satisfy the recency condition for u_1.
5. No vertex appears more than twice in the path.

Condition 2a represents vertex u_{i+1} obtaining an additional incoming edge, and 2b represents vertex u_{i+1} removing an outgoing edge to u_i. Condition 3 represents removing a recency edge from vertex u_i, and then u_i obtaining a new recency edge. Call such a path a *backtracking sequence* . Although a backtracking sequence is represented as a series of edges, it is actually a series of proposed actions. If an edge in the sequence does not yet exist in the graph, then its presence in the sequence represents a proposal to add that edge to the graph. If an edge in the sequence already exists in the graph, then its presence in the sequence represents proposing removing that edge from the graph.

Another way to interpret this method is to consider an individual edge at a time. If v is attempting to obtain an edge from a vertex w in a previous level, then w must have more incoming than outgoing edges. If that is not currently the case, then w can either obtain a new incoming edge or remove an outgoing edge. If we remove an edge outgoing from w and incoming to some vertex x, then x may now have fewer incoming edges than outgoing edges, and so x must either obtain a new incoming edge or remove an outgoing edge, and so on. If the edge removed was necessary to satisfy the recency condition for x, then x must obtain a new incoming edge which satisfies the recency condition.

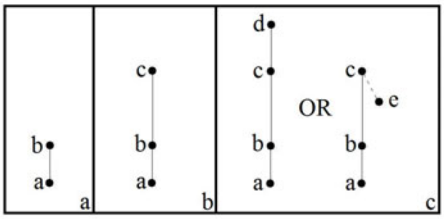


Fig. 5. A Backtracking Example

Consider Fig. 5. In Fig. 5a, vertex a must get a recency edge, so we attempt to add an edge from vertex b in the previous level. If vertex b does not have enough incoming edges to add this edge, then b must obtain a new incoming edge. Suppose that vertex b asks vertex c for an edge (Fig. 5b). If vertex c does not have enough incoming edges to send an outgoing edge to vertex b, then there are two possible options (Fig. 5c). Vertex c may attempt to get a new incoming edge from a vertex in a previous level, such as vertex d, or vertex c may remove an existing outgoing edge to a later vertex, such as vertex e. We then apply this same procedure to either d or e.

Note that although we originally intended for this algorithm to solve the simplified problem, as stated above, it actually solves the original problem as well. A vertex only obtains a non-recency condition incoming edge when it needs that edge to support some outgoing edge. Thus, a vertex will never have too many incoming edges relative to the number of outgoing edges, so the vertex will not adopt the property too early.

4.6 Proof of Correctness

Claim. Given a set of vertices and a time vector, then for the model of contagion in which a vertex adopts the property after half of its neighbors have adopted the property, the backtracking algorithm described above will create a minimal graph solving the time vector.

Proof. Clearly, any graph constructed by the algorithm solves the problem, so we must show that if there is a solution to a time vector, then this algorithm can find it. Suppose we have a solution graph S to a time vector, and the algorithm has created a partial solution G to that same time vector. We can assume without loss of generality that S is minimal; that is, no edges can be removed from S while leaving a graph that still solves the time vector. Let v be the first vertex in G that is not satisfied. We will create a backtracking sequence for v.

Since v is satisfied in S, v in S has an edge incoming from some vertex u in the previous level. This edge doesn't exist in G, since if it did, then v would be satisfied. This edge is the beginning of the backtracking sequence. If u in G has enough incoming edges to send an outgoing edge to v right away, then we are done. If not, then u in G either doesn't have enough incoming edges or has too many outgoing edges, as compared to u in S. So either u in S has some incoming edge that it doesn't have in G, or u in G has some outgoing edge that it doesn't have in S. This edge is next in the backtracking sequence.

Now the next vertex in the backtracking sequence is in the same position as u was previously. We may be able to immediately add or remove the edge from G, but if not, then this vertex must get a new incoming edge or remove an outgoing edge. To find this edge, again compare G to S. Continue this process recursively. To show that this process terminates, we must show that the backtracking sequence is of finite length. Every edge in the sequence represents an edge in either G or S, and there are only finitely many such edges, so we need to show that each edge is added at most once. See Lemma 1 in the Appendix for proof of this.

After this process ends, some vertices may appear multiple times. However, in most situations, we can easily remove duplicates. Suppose a vertex x appears multiple times. If both the first and last copies of x can obtain any new incoming edge or remove any existing outgoing edge, then the edge that was obtained or removed at the last occurrence could have been obtained or removed at the first occurrence, and so we can delete the part of the sequence between these two copies. In the case when the first copy of x must obtain a new recency edge, but the last copy of x can obtain or remove any edge, we must allow 2 copies of x. The other two cases are similar to the first case. In all cases, we can remove cycles so that there are at most 2 copies of x. □

4.7 A Recursive Closure Algorithm and Running Time

The recursive closure algorithm implements the backtracking algorithm. Going through the vertices in order of level, for each vertex recursively create a set containing vertices in a potential backtracking sequence. Progressing through the vertices in order of level, for each vertex x, do the following:

Recursively create a set of vertices, initially containing only x. Add all vertices from which x can obtain a recency edge. At each successive step, expand the set to contain vertices from which newly added members can obtain a new incoming edge or remove an outgoing edge. Continue until a vertex with an excess of incoming edges is found (including level 1 vertices). If no such vertex is found, then there is no backtracking sequence for x. Note that if a vertex is deprived of a recency edge, it must obtain a new recency edge. To handle this, associate a 'flag' with each vertex in the set. If a vertex is not flagged, it may obtain any new incoming edge or remove an existing outgoing edge. If it is flagged, it can only obtain a new recency edge. See Fig. 6 for pseudocode showing how to satisfy one vertex. x' is a flagged version of x and S is an ordered list.

This algorithm runs in polynomial time, since duplicate elements are removed from S. Each vertex appears at most twice- once flagged and once unflagged. For each vertex, we may inspect every edge adjacent to that vertex, so the satisfying operation takes $O(n^2)$ time, where n is the number of vertices. This operation is performed once for every vertex, so the overall running time is $O(n^3)$.

```
Create list S containing x' //x needs recency edge
FOR every element y in S
 IF y is unflagged
  FOR every vertex z
   IF z occurs before y AND (z, y) does not exist //potential incoming edge
    Add z to S, Add a pointer from z in S to y in S
   ELSEIF z occurs after y AND (y, z) exists //existing outgoing edge
    IF (y, z) is the only recency edge for z
     Add z' to S, Add a pointer from z' in S to y in S
    ELSE
     Add z to S, Add a pointer from z in S to y in S
 ELSE //y is flagged, so must get a new recency edge
  FOR every vertex z in the previous level
   Add z to S, Add a pointer from z in S to y in S
  FOR every vertex w in S
   IF w' and w both appear in S
    Remove w' from S
   IF w appears multiple times in S
    Remove extra copies of w
  FOR every unflagged vertex w in S
   IF vertex w has an excess of incoming edges or w is in level 1
   //found a satisfying sequence
    Reduce number of edges incoming to w by 1
    Follow pointers from w back to x' to determine satisfying sequence
    Modify graph in accordance with satisfying sequence
    TERMINATE SATISFY
No satisfying sequence exists //exited for loop without finding a sequence
TERMINATE SATISFY
```

Fig. 6. The Satisfy Function of the Recursive Closure Algorithm

4.8 Generalizing the Backtracking Algorithm

Unfortunately, the backtracking algorithm runs in exponential time for $p = \frac{2}{3}$. For $p = \frac{1}{2}$, the backtracking algorithm runs in polynomial time because each vertex is added to or modified in S at most twice. This does not hold for $p = \frac{2}{3}$.

5 Conclusion and Future Directions

We have described a new problem: recovering social networks given information about how a property has spread through the network. We primarily focused on the case in which a vertex adopts the property after $p = \frac{1}{2}$ of its neighbors adopt the property. A flow-based algorithm solved this case but did not generalize to other values of p. We thus considered the backtracking algorithm, in which details were made explicit. Finally, we discussed some difficulties in extending the backtracking algorithm.

The next step is to find algorithms for other values of p. Another extension is to consider multiple properties spreading through the network with different starting points, so instead of a time vector, we would have a time matrix. We could also weight each edge, so that heavier edges are more likely. Another class of problems acknowledges that many solutions are possible, and asks questions about the set of all solutions, such as: do any edges appear in all the graphs? which edges are the most likely to appear? There is much to be done in this area, and we believe that the work presented here will provide a foundation for future work.

References

1. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, pp. 708–766. MIT Press and McGraw Hill (2009)
2. Granovetter, M.: Threshold Models of Collective Behavior. Am. J. Sociol. 83(6), 1420–1443 (1978)
3. Kleinberg, J.: Cascading Behavior in Networks: Algorithmic and Economic Issues. In: Algorithmic Game Theory, ch. 24. Cambridge Universisty Press, Cambridge (2007)
4. Krebs, V.: Mapping Networks of Terrorist Cells. Connections 24(3), 43–52 (2001)
5. Miller, J.C., Hyman, J.M.: Effective Vaccination Strategies for Realistic Social Networks. Physica A 386(2), 780–785 (2007)
6. Xu, J., Chen, H.: Criminal Network Analysis and Visualization. Communications of the ACM 48(6), 100–107 (2005)

Appendix

Lemma 1. *When showing that there is always a valid backtracking sequence, if multiple outgoing edges are requested or incoming edges are deleted from a vertex, then that vertex can obtain that many new incoming edges or delete that many existing outgoing edges, or some combination.*

Proof. In the backtracking sequence, some vertex x may be encountered multiple times. Each time, we add an edge outgoing from x or remove an edge incoming to x. These are edges outgoing from x in S but not G or incoming to x in G but not S. In response to this, x must find a new incoming edge or remove an existing outgoing edge. These edges are incoming to x in S but not G, or outgoing from

x in G but not S. We need to show that there are enough edges of the latter two types that we can respond to edges of the first two types without repeating edges. Define the following: Let a_G be the number of edges incoming to x in G but not S, and a_S be the number of edges incoming to x in S but not G. a_{GS} is the number of edges incoming to x in both G and S. Define b_G, b_S, and b_{GS} similarly, but for outgoing edges. So we need to show that $a_G + b_S \leq a_S + b_G$.

Consider the possible values of these variables. S is minimal, so either x in S has the same number of incoming and outgoing edges, or it has one incoming edge and zero outgoing edges. The same holds for G, since when an edge incoming to x in G is created, it is either to satisfy x's recency condition or to allow the addition of an edge outgoing from x on to some later vertex. In the first case, x has one incoming edge and zero outgoing edges, and in the second case, x has an equal number of incoming and outgoing edges. Thus, we must consider four cases, based on the combinations of the above possibilities for x in G and S.

Case 1: x has an equal number of incoming and outgoing edges in G and S. We need to show that $a_G + b_S \leq a_S + b_G$. x has an equal number of incoming and outgoing edges in both G and S, so $a_G + a_{GS} = b_G + b_{GS}$ and $a_S + a_{GS} = b_S + b_{GS}$. Subtracting the second equation from the first, we get $a_G - a_S = b_G - b_S$, and rearranging gives us the desired result $a_G + b_S = a_S + b_G$.

Case 2: x has an equal number of incoming and outgoing edges in S, x has one incoming edge and zero outgoing edges in G. We need $a_G + b_S \leq a_S + b_G$. x has an excess of incoming edges in G, so if an outgoing edge is requested from x the algorithm terminates and we do not need to make any more modifications. Thus, we can assume that $b_S = 0$, so we need $a_G \leq a_S + b_G$.

There are two sub-cases here: first, that $a_G = 1$ and $a_{GS} = 0$, and second, that $a_G = 0$ and $a_{GS} = 1$. Suppose the first sub-case holds, so $a_G = 1$ and $a_{GS} = 0$. We assumed that $b_S = 0$, so $a_G + b_S = 1$, and $a_S \geq 1$, so certainly $a_G + b_S \leq a_S + b_G$. Suppose the second sub-case holds, so $a_G = 0$ and $a_{GS} = 1$. Since $a_G = 0$, we need to show that $b_S \leq a_S$. We treat b_S as 0, so $b_S \leq a_S$.

Case 3: x has an equal number of incoming and outgoing edges in G, x has one incoming edge and zero outgoing edges in S. Again, there are two sub-cases: first, $a_S = 1$ and $a_{GS} = 0$, and second, $a_{GS} = 1$ and $a_S = 0$. Suppose the first sub-case holds. We need that $a_G + b_S \leq a_S + b_G$. Since $b_S = 0$, we need $a_G \leq 1 + b_G$. Since $b_{GS} = 0$ and $a_{GS} = 0$, and $a_G + a_{GS} = b_G + b_{GS}$, we get that $a_G = b_G$, so certainly $a_G \leq 1 + b_G$ and $a_G + b_S \leq a_S + b_G$. Now suppose we are in the second sub-case. We need $a_G + b_S \leq a_S + b_G$. Since $b_S = 0$ and $a_S = 0$, we need that $a_G \leq b_G$. We know that $a_G + a_{GS} = b_G + b_{GS}$, so $a_G + 1 = b_G$, so then $a_G \leq b_G$ and so $a_G + b_S \leq a_S + b_G$.

Case 4: x has one incoming edge and zero outgoing edges in G and S. We need $a_G + b_S \leq a_S + b_G$. b_G, b_{GS}, and b_S are all zero, so we need that $a_G \leq a_S$. Since x has one incoming edge in both G and S, either $a_G = 1$, $a_S = 1$, and $a_{GS} = 0$, or $a_G = 0$, $a_S = 0$, and $a_{GS} = 1$. In either case, $a_G \leq a_S$.

In each of the four cases, we showed that $a_G + b_S \leq a_S + b_G$. □

Two-Layer Planarization Parameterized by Feedback Edge Set*

Johannes Uhlmann and Mathias Weller

Institut für Informatik, Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{johannes.uhlmann,mathias.weller}@uni-jena.de

Abstract. Given an undirected graph G and an integer $k \geq 0$, the NP-hard 2-Layer Planarization problem asks whether G can be transformed into a forest of caterpillar trees by removing at most k edges. Since transforming G into a forest of caterpillar trees requires breaking every cycle, the size f of a minimum feedback edge set is a natural parameter with $f \leq k$. We improve on previous fixed-parameter tractability results with respect to k by presenting a problem kernel with $O(f)$ vertices and edges and a new search-tree based algorithm, both with about the same worst-case bounds for f as the previous results for k, although we expect f to be smaller than k for a wide range of input instances.

1 Introduction

The focus of this work is on finding good 2-layered drawings of graphs. Such drawings appear in the "Sugiyama" approach [17,13] to multilayered graph drawing [5,8].

In this context, a graph is called *biplanar* if it can be drawn in two layers without edge crossings (while edges are drawn as straight lines). It has been shown that biplanar graphs are exactly the graphs that consist of disjoint caterpillar trees (or caterpillars for short) [13,5]. A caterpillar is a tree where every vertex is adjacent to at most two non-leaf vertices. In this work, we concentrate on the NP-hard 2-Layer Planarization (2LP) problem, that is, the problem to transform a given graph into a forest of caterpillars by deleting a minimum number of edges. Formally, this problem is defined as follows. Given an undirected graph $G = (V, E)$ and a non-negative integer k, determine whether there is an edge subset $E' \subseteq E$ with $|E'| \leq k$ such that $(V, E \setminus E')$ is biplanar.

Apart from being proposed as an alternative method to minimize crossings [13], solving 2LP is important in DNA mapping [19] and global routing for row-based VLSI layout [12]. 2LP is NP-hard even in the case that the input graph is bipartite and in one partition each vertex has degree at most two [7]. Shahrokhi et al. [15] presented a dynamic programming based linear-time algorithm solving the problem on trees. Concerning the parameter k (number of edge deletions),

* Supported by the DFG, research projects PABI (NI 369/7) and DARE (GU 1023/1, NI 369/11).

© Springer-Verlag Berlin Heidelberg 2010

Dujmović et al. [5] showed that 2LP can be solved in $O(k \cdot 6^k + |G|)$ time by devising a search tree algorithm and several polynomial-time data reduction rules leading to a problem kernel with $O(k)$ vertices and edges. Later, Fernau [8] presented a refined search tree for 2LP leading to a running time of $O(k^2 \cdot 5.19276^k + |G|)$. Finally, based on a different branching analysis, Suderman [16] developed an $O(k \cdot 3.562^k + |G|)$-time algorithm.

2LP is a special case of the problem of transforming a binary matrix into a matrix with so-called "consecutive ones property" by a minimum number of column removals. More specifically, 2LP coincides with this problem for matrices without identical columns that have a maximum of two 1s in each column [3].

In this work, we investigate the parameterized complexity of 2LP with respect to the parameter "feedback edge set number" f, that is, the minimum number of edges whose removal results in an acyclic graph. Note that the feedback edge set number of a connected n-vertex and m-edge graph is $f(G) = m - n + 1$ and a minimum feedback edge set can be determined by the computation of a spanning tree in $O(n + m)$ time via depth-first search. We develop efficient preprocessing rules for 2LP that lead to a problem kernel with $O(f)$ vertices and $O(f)$ edges. Moreover, we present a new search tree algorithm leading to a total running time of $O(6^f + f \cdot m)$ for solving 2LP.

Our work is motivated as follows. First, note that for 2LP the number of necessary edge deletions is at least the feedback edge set number, since one has to destroy all cycles to obtain a forest of caterpillars. In this sense, we improve on the results of Dujmović et al. [5] by providing fixed-parameter algorithms and kernelizations with about the same worst-case bounds for a parameter that we expect to be significantly smaller for a wide range of input instances. Second, Dujmović et al. [5] pointed out that "instances of 2-Layer Planarization for *dense* graphs are of little interest from a practical point of view" since the resulting drawings are unreadable anyway. Thus, they expect the solution size to be small in practice. This is even more plausible for the feedback edge set number of a graph which is directly linked to the number of edges, and, hence, the sparseness of the graph. Also note that the solution size can be arbitrarily large even for trees (the sparsest connected graphs) while the feedback edge set number of trees is zero. Measuring the distance from trees by the feedback edge set number can be seen as a parameterization by "distance from triviality" [10]. In this sense, our results generalize the liner-time algorithm for trees by Shahrokhi et al. [15]. Third, the feedback edge set number f is a parameter that can easily be computed in advance and, hence, allows for a meta-algorithm that chooses an algorithm for a given input by computing an estimation on the running time prior to running the algorithm for the problem itself. Since the parameter k ("number of edge deletions") is NP-hard to compute, such an algorithm could not efficiently determine the running time of an algorithm parameterized by k in advance. Fourth, looking for smaller parameters may help to further extend the range of solvable instances. However, this seems only possible if the new algorithms have a modest exponential part of the running time.

Last but not least, efficient preprocessing or polynomial-time data reduction seems to be essential to obtain good fixed-parameter algorithms. Indeed, kernelization has been recognized as one of the key techniques of parameterized algorithmics [1,11,14]. In this context, one of our main contributions is to provide a set of new polynomial-time data reduction rules for 2LP. Due to the lack of space, most proofs are deferred to a long version of this paper.

Preliminaries. Let G be a graph. For every vertex set $V' \subseteq V(G)$, we denote the subgraph of G that is induced by V' by $G[V']$ and we write $G - V'$ for $G[V(G) \setminus V']$. Equivalently, for every edge-set $S \subseteq E(G)$, consider $G - S$ an abbreviation for $(V(G), E(G) \setminus S)$ and $V(S)$ the set of endpoints of edges in S. We denote the neighborhood of a vertex $v \in V(G)$ in G with $N_G(v)$ and the degree of v in G with $\deg_G(v)$. If clear from the context, we omit the index. Furthermore, let $I(G)$ (isolated vertices) and $L(G)$ (leaves) denote the set of vertices in G with degree zero and one, respectively. Following [5], we define the *non-leaf degree* $\widehat{\deg}_G(v) := |N_G(v) \setminus L(G)|$ for every vertex $v \in V(G)$.

A *caterpillar tree* (or caterpillar for short) is a tree where every vertex has non-leaf degree at most two. Equivalently, a caterpillar is a tree that does not contain a 2-claw [7] (see Figure 1*a*)). Thus, caterpillars have a forbidden subgraph characterization. A leaf $v \in L(G)$ is called *critical* if its only neighboring vertex has non-leaf degree two. The definition of critical vertices is motivated by the observation that being a caterpillar is invariant with respect to adding neighbors to non-critical vertices.

Informally speaking, G^* denotes the subgraph of G that contains all edges that are contained in a cycle or that connect vertices that are contained in a cycle. Formally, set $G^0 := G$ and recursively define $G^{i+1} := G^i - (L(G_i) \cup I(G_i))$. Finally, let G^* denote the graph G^i with minimum i such that $G^i = G^{i+1}$. Note that G^* is the empty graph iff G is acyclic (a forest). Moreover, for G being a forest of caterpillar trees, G^1 is a forest of paths. Furthermore, note that $G - V(G^*)$ is acyclic (a forest). For a vertex $v \in V(G^*)$ let T^v denote the tree of $G \setminus E(G^*)$ that is rooted at v. The tree T^v is called the pendant tree of v and v is called its connection point. Furthermore, for a rooted tree T and for a vertex $x \in T$ let T_x denote the subtree of T rooted at x.

The following special pendant trees are of particular interest in this work. For a vertex v, let $L(v) := L(G) \cap N(v)$. A path $p = (\{v, w\}, \{w, x\})$ is called a P_2 with connection point v if $\deg(v) \geq 2$, $\deg(w) = 2$, and $\deg(x) = 1$, see Figure 1*b*) for an example. Vertex w is called the middle point and we refer to it as $m(p)$ and vertex x is called the leaf of p denoted by $l(p)$. For a vertex v let $\mathcal{P}_2(v)$ denote the set of all P_2's that have v as their connection point. A Y-graph is defined as shown in Figure 1*c*). Vertex v is called the connection point and vertex w is called the center point of the Y-graph. We refer to w by $c(Y)$. Let $\mathcal{Y}(v)$ denote the set of all Y-graphs that have v as their connection point.

Our results are in the context of parameterized complexity, which is a two-dimensional framework for studying computational complexity [4,9,14]. One dimension is the input size n, and the other one is the *parameter* (usually a positive integer). A problem is called *fixed-parameter tractable* (fpt) with respect to a

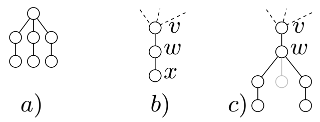


Fig. 1. Terminology. *a*) A 2-claw. The degree-three vertex of a 2-claw is called its center vertex. Caterpillars can be characterized as graphs containing neither cycles nor 2-claws. Figure *b*) shows a P_2 and Figure *c*) a Y-graph. In *c*) the gray leaf may or may not be present in the Y-graph (formally, there are two different Y-graphs, one with and one without the gray leaf).

parameter k if it can be solved in $f(k) \cdot n^{O(1)}$ time, where f is a computable function only depending on k. A core tool in the development of fixed-parameter algorithms is polynomial-time preprocessing by *data reduction*. Here, the goal is to transform a given problem instance x with parameter k into an equivalent instance x' with parameter $k' \leq k$ such that the size of x' is upper-bounded by some function only depending on k. This is usually achieved by applying data reduction rules. We call a data reduction rule *correct* if the new instance after an application of this rule is a yes-instance iff the original instance is a yes-instance. An instance is called *reduced* with respect to some data reduction rule if this rule can not be applied to the instance. The whole process is called *kernelization*.

2 Kernelizing 2-Layer Planarization

In this section, we present a kernelization for 2LP parameterized by f, denoting the size of a minimum feedback edge set. We present a number of polynomial-time executable data reduction rules and show that a graph that is reduced with respect to these rules cannot contain more than $O(f)$ vertices and $O(f)$ edges. As noted before, this result improves previous work by Dujmović et al. [5]. The kernelization consists of two phases. In the first phase, which we call "tree reduction", roughly speaking, the goal is to reduce the "acyclic part" of the input graph. In the second phase, the goal is to reduce the long non-branching paths in the remaining "cyclic core" G^*, shrinking its size to a function linear in f. We call the second phase "path replacement".

Tree Reduction. Subsequently, we present reduction rules for repeatedly replacing a pendant tree T^u for some $u \in V(G)$ with a smaller tree, until its size is a constant value (see Figure 2 for an illustration). Tree Reduction Rule 1 below is from [5].

Tree Reduction Rule 1. *If there is a vertex v in T^u with $|L(v)| \geq 2$, then delete all but one leaf in $L(v)$.*

Tree Reduction Rule 2. *If there is a vertex v in T^u with $|\mathcal{Y}(v)| \geq 1$ and $|\mathcal{Y}(v)| + |L(v)| + |\mathcal{P}_2(v)| \geq 2$, then, for an arbitrarily chosen Y-graph $Y \in \mathcal{Y}(v)$, delete all vertices of Y except for v and decrease k by one.*

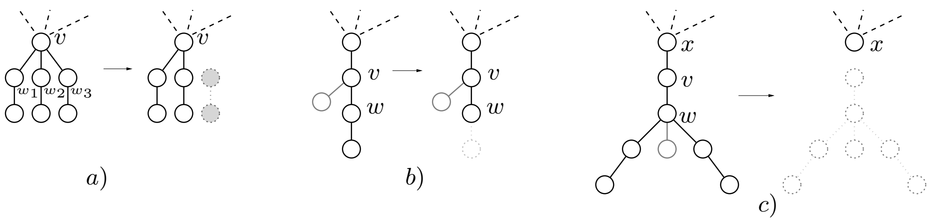


Fig. 2. a) Example for the application of Tree Reduction Rule 3. Note that we must delete one of the edges $\{v, w_i\}$, $i = 1, 2, 3$ in order to destroy the 2-claw centered at v. By symmetry, there is an optimal solution that contains the edge $\{v, w_3\}$. b) Example for the application of Tree Reduction Rule 4. Note that the edge $\{w, x\}$ is not contained in any 2-claw and hence, x can be deleted. c) Example for the application of Tree Reduction Rule 5. Since the deletion of the edge $\{x, v\}$ destroys the same 2-claws as the deletion of any other edge in the tree rooted at x, there is an optimal solution that contains $\{x, v\}$.

Tree Reduction Rule 3. *Consider a vertex v in T^u with $|\mathcal{P}_2(v)| \geq 3$. Let $\mathcal{P}_2(v) = \{p_1, p_2, \ldots, p_q\}$. Delete the vertices $l(p_i)$ and $m(p_i)$ for $3 \leq i \leq q$ and decrease k by $q - 2$.*

Tree Reduction Rule 4. *Consider a vertex v in T^u with $\widehat{\deg}_{T^u}(v) = 2$ and $|\mathcal{P}_2(v)| = 1$. Let $\mathcal{P}_2(v) = \{p\}$. Then delete the vertex $l(p)$.*

Tree Reduction Rule 5. *Consider a vertex v in T^u with $\deg_G(v) = 2$ and $|\mathcal{Y}(v)| = 1$. Let $\mathcal{Y}(v) = \{Y\}$. Then delete all vertices of Y (including v) and decrease k by one.*

Tree Reduction Rule 6. *If C is a connected component of G that is a caterpillar, then delete all vertices of C.*

Lemma 1. *Tree Reduction Rules 1–6 are correct. An instance reduced with respect to Tree Reduction Rules 1–6 can be computed in $O(|V| + |E|)$ time.*

The structure of an instance reduced with respect to these rules is described by the following lemma, which concludes the presentation of the "tree reduction".

Lemma 2. *In a reduced instance, for every vertex $v \in V(G^*)$, its pendant tree T^v is either a singleton or isomorphic to one of the trees shown in Figure 3.*

Path Replacement. The tree reduction rules presented in the previous paragraph are not sufficient to yield a problem kernel for our parameterization. For example, if the input graph G is a simple cycle, then none of the above data reduction rules applies. Recall the notions of G^* (also called the "cyclic core" of G) and pendant trees. The purpose of the subsequently presented data reduction rules is to reduce non-branching paths of G^* (hence, they are called "path reduction rules"). The first two reduction rules take care of paths containing Y-graphs as

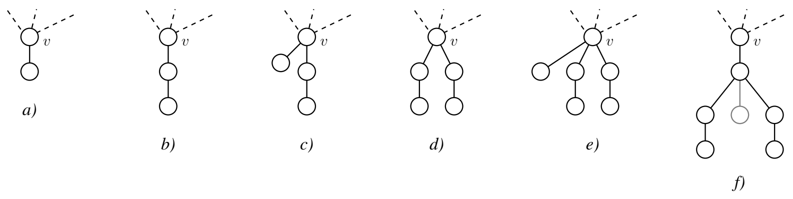


Fig. 3. In a reduced instance, the pendant tree T^v of a vertex $v \in V(G^*)$ is isomorphic to one of the trees shown in *a*) to *f*). Note that the tree shown in *f*) is exactly the Y-graph as defined in Figure 1*b*).

pendant trees. Then, we introduce the notion of tokens which allows us to handle the remaining cases in a unified manner. The reduction rules in this paragraph are more intricate than in the previous paragraph.

In the following, we assume the input to be reduced with respect to all tree reduction rules (see previous paragraph). Consider vertices $u, w \in V(G^*)$ that may be identical. We denote a path $P_{u,w} = (u = v_0, v_1, \ldots, v_\ell, v_{\ell+1} = w)$ between u and w as *degree-2 path* if $\deg_{G^*}(v_i) = 2$ for all $1 \leq i \leq l$. Its *length* is $\ell + 2$. We refer to the vertices v_i, $1 \leq i \leq l$, as *inner path vertices*. Furthermore, denote the edges $\{v_{i-1}, v_i\}$ by e_i for all $1 \leq i \leq \ell + 1$. Throughout this paragraph, let P be some degree-2 path in the given graph G and let v_i be some inner path vertex of P. In this context, let $T_P := G[\bigcup_{i=1}^{l} V(T^{v_i}) \cup \{u, w\}]$ and for $1 \leq i \leq j \leq l$, let $T_{i,j}$ denote the subtree of T_P containing all vertices reachable from v_i in $T_P - \{e_i, e_{j+1}\}$. Note that $T_{i,i} = T^{v_i}$.

The next two data reduction rules handle all Y-graphs that have a vertex on a degree-2 path as their connection point. First, observe that for any Y-graph Y, deleting the edge that is incident to the connection point is at least as good as deleting any combination of edges of Y.

Observation 1. *Let Y denote some Y-graph in G with connection point v. Then there is an optimal solution S for G with $S \cap E(Y) \subseteq \{\{v, c(Y)\}\}$.*

The first rule identifies Y-graphs Y with connection point v_i for which it is optimal to delete the edge $\{v_i, c(Y)\}$.

Path Reduction Rule 1. *Let P denote a degree-2 path and let both v_i and v_{i+1} be inner path vertices of P such that $Y := T^{v_i}$ is a Y-graph. If*

1. *$T^{v_{i+1}}$ is neither a singleton nor a Y-graph, or*
2. *$\deg_G(v_{i+1}) = 2$, v_{i+2} is an inner path vertex, and $T^{v_{i+2}}$ is either a singleton or a Y-graph,*

then delete $\{v_i, c(y)\}$ and decrease k by one.

The second rule handles almost all remaining cases where a Y-graph occurs as the pendant tree of some inner path vertex v_i of P by bypassing v_i in P.

Path Reduction Rule 2. *Let P denote a degree-2 path and let both v_i and v_{i+1} be inner path vertices of P such that $Y := T^{v_i}$ is a Y-graph. If*

1. *$T^{v_{i+1}}$ is a Y-graph, or*
2. *v_{i-1} and v_{i+1} have degree two, or*
3. *v_{i+2} is an inner path vertex, $\deg(v_{i+1}) = 2$, and $T^{v_{i+2}}$ is neither a singleton nor a Y-graph,*

then remove all vertices of Y from G, insert the edge $e = \{v_{i-1}, v_{i+1}\}$, and decrease k by one.

For the correctness of Path Reduction Rule 2, we need the following lemma.

Lemma 3. *Let v be a degree-2 vertex in G^* such that T^v is neither a singleton nor a Y-graph. Then there is an optimal solution S for G such that v is non-critical in $G - S$.*

Lemma 4. *Path Reduction Rule 2 is correct.*

Proof. Let G' denote the graph that results from applying Path Reduction Rule 2 to some v_i in G and let $e_Y := \{v_i, c(Y)\}$. For the correctness, we prove that G has a solution S of size k if and only if G' has a solution S' of size $k - 1$.

Let $\hat{G}$ denote the result of contracting e_{i+1} in $G - \{e_Y\}$ and observe that $\hat{G}$ is identical to G' with the exception of one connected component (containing all vertices of Y but v_i) which is a caterpillar. Obviously, $\hat{G}$ and G' are equivalent in the sense that a solution for one is also a solution for the other (considering that e_i in $\hat{G}$ plays the role of e in G').

"$\Rightarrow$:" If $e_Y \in S$, then, since contracting an edge does not create 2-claws or cycles, $S' := S \setminus \{e_Y\}$ is a solution of size $k - 1$ for $\hat{G}$ and, thus, for G'. Otherwise, by Observation 1, no edge of Y is in S and thus, $e_i \in S$ and $e_{i+1} \in S$. Moreover, by construction, $G' - \{e\}$ is a subgraph of $G - \{e_i, e_{i+1}\}$ and thus, $S' := S \setminus \{e_1, e_2\} \cup \{e\}$ is a solution for G' of size $k - 1$.

"$\Leftarrow$:" First, if a solution S' for G' of size $k-1$ contains e, then the equivalent solution $\hat{S}$ for $\hat{G}$ contains e_i, and clearly, $S := \hat{S} \cup \{e_{i+1}\}$ is a size-k solution for G. Thus, in the following, we assume that there is no solution for G' of size $k - 1$ that contains e (in particular, $e \notin S'$ and thus v_{i-1} and v_{i+1} are neighbors in $G' - S'$).

Second, observe that the subdivision of an edge e' of a caterpillar can only create a 2-claw if e' is incident to a critical vertex. Hence, if v_{i-1} and v_{i+1} are both non-critical in $G' - S'$, then we can subdivide e without affecting the solution and thus, $S := S' \cup \{e_Y\}$ is a size-k solution for G. Hence, in the following, we assume that v_{i-1} or v_{i+1} is critical in $G' - S'$. In the following, we consider the three cases of Path Reduction Rule 2 separately.

Case 1: The first condition of Path Reduction Rule 2 applies.
Then, $Y' := T^{v_{i+1}}$ is a Y-graph. Let $e_{Y'} := \{v_{i+1}, c(Y')\}$. By Observation 1, we can assume that S' contains either $e_{Y'}$ or both e and e_{i+2}. However, by the above assumption, $e \notin S'$ and thus, $e_{i+2} \notin S'$ and $e_{Y'} \in S'$, implying $\deg_{G'-S'}(v_{i+1}) = 2$. Thus, neither v_{i+1}, nor v_{i-1} is critical in $G' - S'$, contradicting the assumption above.

Case 2: The second condition of Path Reduction Rule 2 applies. Then, $\deg_G(v_{i-1}) = \deg_G(v_{i+1}) = 2$. Clearly, if any of v_{i-1}, v_{i+1} is a leaf in $G' - S'$, then the other cannot have non-leaf degree two in $G' - S'$. Thus, neither of them is critical in $G' - S'$, contradicting the assumption above.

Case 3: The third condition of Path Reduction Rule 2 applies. By Lemma 3 we can assume that v_{i+2} is non-critical in $G' - S'$. Clearly, v_{i-1} is non-critical in $G' - S'$ since $\deg_G(v_{i+1}) = 2$. Hence, v_{i+1} is critical in $G' - S'$ and thus, $S' \cup \{e\}$ isolates v_{i+1}. Since v_{i+2} is non-critical in $G' - S'$, it follows that $(S' \cup \{e\}) \setminus \{e_{i+2}\}$ is a size-k solution for G' containing e, contradicting the assumption above. □

The two path reduction rules presented so far eliminate Y-graphs in all long degree-2 paths. In the following, consider P to be *Y-graph-free*, that is, P does not contain an inner path vertex whose pendant tree is a Y-graph. Consider a graph that is reduced with respect to Path Reduction Rules 1 and 2. All degree-2 paths P' that are not Y-graph-free contain at most two inner path vertices, whose pendant trees are a singleton and a Y-graph, respectively. Thus, it is clear that $|V(T_{P'})| \leq 9$. In the following, we focus on Y-graph-free degree-2 paths. In this case, we can show that we do not need to consider deleting edges in pendant trees, thus allowing us to restrict our attention to deleting edges on the degree-2 path.

Lemma 5. *Let P be a Y-graph-free degree-2 path. There is an optimal solution for G that does not contain any edge of $E(T_P) \setminus E(P)$.*

To handle the remaining paths in a unified manner, we introduce the notion of "tokens" as sets of consecutive edges of P. Consider a vertex v in P that is the center of a 2-claw. Then, Lemma 5 tells us that this 2-claw must be destroyed by deleting an edge of P. We model this fact by letting v generate a token containing all edges that may be deleted in order to destroy this 2-claw. The introduction of this notion is split into three parts. First, we define tokens, second, we point out how they are generated, and third, we specify what it means to destroy a token.

In the following, a vertex v in P is called *crossable* if $\deg_G(v) = 2$. A *token* K of P is a set of at most 4 consecutive edges of P. Let v_i be a vertex in P. If $i \leq \ell - 1$, then the *upper token* $K^{\text{up}}(v_i)$ of v_i is $\{e_{i+1}, e_{i+2}\}$ if v_{i+1} is crossable and it is $\{e_{i+1}\}$, otherwise. Equivalently, if $i \geq 2$, then the *lower token* $K^{\text{low}}(v_i)$ of v_i is $\{e_{i-1}, e_i\}$ if v_{i-1} is crossable and it is $\{e_i\}$ otherwise. In this sense, we say that tokens can only "span" over crossable vertices. For the vertices u, v_1, v_ℓ, and w, we need the following auxiliary tokens: $K^{\text{low}}(u) := K^{\text{up}}(w) := \{\diamond\}$, $K^{\text{low}}(v_1) := \{\diamond, e_1\}$, and $K^{\text{up}}(v_\ell) := \{e_{\ell+1}, \diamond\}$.

Each inner path vertex v_i of P for which $\mathcal{P}_2(v_i) \neq \emptyset$ *generates* tokens in the following way: If $|\mathcal{P}_2(v_i)| = 1$ (in this case T^{v_i} is isomorphic to one of the trees shown in Figure 3 *b*) and *c*)), then v_i generates one token $K^{\text{up}}(v_i) \cup K^{\text{low}}(v_i)$. If $|\mathcal{P}_2(v_i)| = 2$ (in this case T^{v_i} is isomorphic to one of the trees shown in Figure 3 *d*) and *e*)), then v_i generates two tokens, $K^{\text{up}}(v_i)$ and $K^{\text{low}}(v_i)$. We define $\mathcal{K}(v_i)$ as the set of tokens generated by v_i and $\mathcal{K}(P)$ as the set of all tokens generated by inner path vertices of P. See Figure 4 for an illustration.

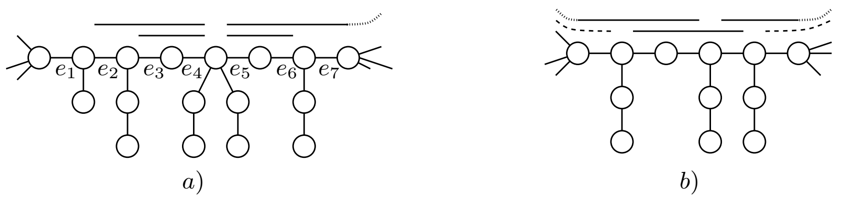


Fig. 4. *a*) Illustration of the tokens generated by a degree-2 path. The tokens are depicted by horizontal bars. That is, the tokens are $\{e_2, e_3, e_4\}$, $\{e_3, e_4\}$, $\{e_5, e_6\}$, and $\{e_5, e_6, e_7, \diamond\}$. *b*) Example of a chained degree-2 path. The tokens are depicted by horizontal bars. Furthermore, dashed lines represent auxiliary tokens.

A non-auxiliary token (token that does not contain $\diamond$) $K \in \mathcal{K}(P)$ is *destroyed* by an edge set E' if $K \cap E' \neq \emptyset$. Observe that if K contains $\diamond$, then it either contains e_1 or e_ℓ. We say that a token containing e_1 and $\diamond$ is destroyed by E' if either $K \cap E' \neq \emptyset$ or E' contains all edges incident to v_0 except for e_1. A token containing $e_{\ell+1}$ is destroyed analogously. Informally speaking, a token represents the need to delete an edge. By Lemma 5 it suffices to consider edges in P. Thus, the task is to destroy all tokens by deleting only few edges.

In the following, we present path reduction rules to shrink degree-2 paths. The first rule reduces degree-2 paths that do not contain any tokens.

Path Reduction Rule 3. *If there is a degree-2 path P with $|V(T_P)| > 7$ and $\mathcal{K}(P) = \emptyset$, then contract $T_{2,\ell-1}$ to a single vertex.*

Next, we concentrate on degree-2 paths generating tokens. Note that for some degree-2 path P, the end vertices u and w could be the center of a 2-claw containing inner path vertices of P. To account for this possibility, we define $\mathcal{K}'(P) := \mathcal{K}(P) \cup \{K^{\text{up}}(u), K^{\text{low}}(w)\}$. Furthermore, since tokens are basically edge sets, they may overlap. This behavior is exploited in the following reduction rules requiring a more formal definition. We call an inner path vertex v_i of P a *token separator* if there is no token in $\mathcal{K}'(P)$ containing both e_i and e_{i+1}. Finally, P is called *chained*, if it does not have a token separator (see Figure 4*b*)).

Path Reduction Rule 4. *Let P be a degree-2 path with $\mathcal{K}(P) \neq \emptyset$ and P is not chained. Let v_i be a token separator of P. Then replace $T_{i,i}$ by two copies of $T_{i,i}$, connect one to v_{i-1} (by inserting an edge between its connection point and v_{i-1}), and connect the other to v_{i+1}.*

See Figure 5 for an illustration of Path Reduction Rule 4.

Path Reduction Rule 5. *Let P be a chained degree-2 path with $\mathcal{K}(P) \neq \emptyset$. Let $M_P := \{i \in \mathbb{N} \mid \mathcal{K}(v_i) \neq \emptyset\}$, let $p := \min M_P$ and $q := \max M_P$, and suppose that $q - p > 1$. If $|\mathcal{K}(P)|$ is even, then delete $T_{p+1,q-1}$, insert the edge $e := \{v_p, v_q\}$, and reduce k by $(|\mathcal{K}(P)| - 2)/2$; otherwise delete $T_{p+1,q}$, insert the edge $e := \{v_p, v_{q+1}\}$, and reduce k by $(|\mathcal{K}(P)| - 1)/2$.*

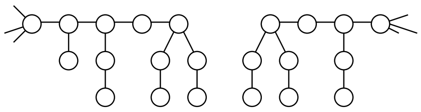


Fig. 5. The graph that results from applying Path Reduction Rule 4 to the graph shown in Figure 4a)

Lemma 6. *Let G be reduced with respect to all reduction rules presented in this section and let V_3 denote the set of vertices with degree at least 3 in G^*. If two vertices $u, w \in V_3$ are connected in G by a degree-2 path, then the subgraph of G connecting u and w contains at most 10 vertices and at most 11 edges.*

Theorem 1. 2-Layer Planarization *admits a linear problem kernel containing at most $44(f-1)$ vertices and at most $45(f-1)$ edges, with $f > 0$ being the size of a minimum feedback edge set of G. The problem kernel can be constructed in $O(f \cdot |E|)$ time.*

Proof. Assume that the input G is reduced with respect to all presented data reduction rules. For the analysis of the kernel size, we need the following notation. Let V_3 denote the set of vertices with degree at least 3 in G^* and let $G_3^* := (V_3, E_3)$ denote the mulitgraph on V_3 that contains an edge for every maximal degree-2 path in G. More specifically, E_3 contains an edge $\{u, w\}$ for every edge $\{u, w\} \in E$ with $u, w \in V_3$, and, in addition, E_3 contains an edge $\{u, w\}$ for every maximal degree-2 path of length at least three between two (not necessarily different) vertices $u, w \in V_3$. Thus, G_3^* may contain loops. Furthermore, let F with $|F| = f$ be a minimum feedback edge set of G and let F_3 be a minimum feedback edge set of G_3^* (we require that a feedback edge set of G_3^* contains all loops and all but at most one edge between any two vertices). Clearly, $|F_3| \leq f$ and $G_3^* - F_3$ is a forest and, thus, $|E_3| \leq |V_3| + f - 1$. Since the minimum degree[1] of a vertex in G_3^* is 3, we know that $\sum_{v \in V_3} \deg_{G_3^*}(v) \geq 3|V_3|$, and since the sum on the left hand side equals $2|E_3|$, we know that $2(|V_3| + f - 1) \geq 3|V_3|$, implying $|V_3| \leq 2(f-1)$ and $|E_3| \leq 3(f-1)$.

With G_3^* bounded, we can use Lemma 6 to bound G. Each edge in G_3^* corresponds to a degree-2 path in G^*. Each vertex in V_3 may additionally be incident to a pendant tree (see Figure 3). Thus, we can bound the number of vertices in G by $|V(G)| \leq |E_3| \cdot 10 + |V_3| \cdot 6 + |V_3| \leq 44(f-1)$ and the number of edges in G by $|E(G)| \leq 45(f-1)$. □

3 Search Tree, Further Results, and Open Questions

In this section, we provide an algorithm that solves the 2-Layer Planarization problem in $O(6^f \cdot f^2 + f \cdot |E|)$ time. It employs a search tree based strategy that makes use of the kernelization presented in the previous section.

[1] A loop at vertex v contributes 2 to the degree of v.

The algorithm runs in three phases. First, we apply a search tree enumerating partial solutions by branching on a certain type of 2-claw. Second, we branch on small cycles in the remaining graph. In the third phase, we branch on the tokens (see Section 2, page 438) that remain in the graph. The input graph G considered in each phase is assumed not to be subject to the previous phase, that is, G does not contain a structure that is branched on in the previous phase. Furthermore, the input graph of each phase is assumed to be reduced with respect to the data reduction rules presented in the previous section. We split the number of edge deletions done by branching in the three phases into f_1, f_2, and f_3 with $f_1 + f_2 + f_3 = f$.

Theorem 2. 2-Layer Planarization *can be solved in* $O(6^f \cdot f \cdot |E|)$ *time.*

By initially kernelizing the input instance, we can assume $|E|$ to be linear in f for the branching algorithm.

Corollary 1. 2-Layer Planarization *can be solved in* $O(6^f \cdot f^2 + f \cdot |E|)$ *time. Moreover, if* $|E| \leq |V| + O(\log |V|)$*, then* 2-Layer Planarization *can be solved in polynomial time.*

We can show that our kernelization results for 2LP can be extended to the closely related NP-hard Node Duplication based Crossing Elimination(NDCE) problem [2]. In NDCE the task is to transform an input graph into a forest of caterpillar trees by a minimum number of node duplications. Herein, duplicating a vertex v means to delete v, add two new vertices v_1, v_2, and make every former neighbor of v adjacent to exactly one of v_1 or v_2. NDCE arises in the design of molecular quantum-dot cellular automata [2] and visualization of gene ontologies in bioinformatics [18]. Chaudhary et al. [2] showed the NP-hardness of NDCE and presented an ILP formulation for it. The parameterized complexity of NDCE seems unexplored. We can show that NDCE has a linear problem kernel with respect to the feedback edge set number of the graph. Note that the feedback edge set number is also a lower bound for the number of node duplications required to transform a graph into a forest of caterpillars. Since the techniques employed by this kernelization are very similar to the ones presented for 2LP, we defer these elaborations to a long version of this paper.

We conclude with some open questions for future work. It is interesting to investigate whether our kernelization approach also holds for the edge-weighted case. Moreover, it is interesting to investigate whether the branching analysis suggested by Suderman [16] can be used to obtain a better search tree algorithm for the parameter feedback edge set number. Providing efficient fixed-parameter algorithms for parameters upper-bounded by the feedback edge set is a natural next step to extend the range of solvable instances. The feedback vertex set number would be a canonical candidate. Finally, it would be interesting to extend our results to the multilayered problem versions [6].

References

1. 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)
2. Chaudhary, A., Chen, D.Z., Hu, X.S., Niemier, M.T., Ravichandran, R., Whitton, K.: Fabricatable interconnect and molecular QCA circuits. IEEE Trans. on CAD of Integrated Circuits and Systems 26(11), 1978–1991 (2007)
3. Dom, M., Guo, J., Niedermeier, R.: Approximation and fixed-parameter algorithms for consecutive ones submatrix problems. Journal of Computer and System Sciences (2010)
4. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
5. Dujmović, V., Fellows, M., Hallett, M., Kitching, M., McCartin, G.L.C., Nishimura, N., Ragde, P., Rosamond, F., Suderman, M., Whitesides, S., Wood, D.R.: A fixed-parameter approach to 2-layer planarization. Algorithmica 45(2), 159–182 (2006)
6. Dujmović, V., Fellows, M., Hallett, M., Kitching, M., McCartin, G.L.C., Nishimura, N., Ragde, P., Rosamond, F., Suderman, M., Whitesides, S., Wood, D.R.: On the parameterized complexity of layered graph drawing. Algorithmica 52(2), 267–292 (2008)
7. Eades, P., Whitesides, S.: Drawing graphs in two layers. Theoretical Computer Science 131(2), 361–374 (1994)
8. Fernau, H.: Two-layer planarization: Improving on parameterized algorithmics. Journal of Graph Algorithms and Applications 9(2), 205–238 (2005)
9. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
10. Guo, J., Hüffner, F., Niedermeier, R.: A structural view on parameterizing problems: Distance from triviality. In: Downey, R.G., Fellows, M.R., Dehne, F. (eds.) IWPEC 2004. LNCS, vol. 3162, pp. 162–173. Springer, Heidelberg (2004)
11. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. ACM SIGACT News 38(1), 31–45 (2007)
12. Lengauer, T.: Combinatorial algorithms for integrated circuit layout. John Wiley & Sons, Inc., New York (1990)
13. Mutzel, P.: An alternative method to crossing minimization on hierarchical graphs. SIAM Journal on Optimization 11(4), 1065–1080 (2001)
14. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and Its Applications, vol. 31. Oxford University Press, Oxford (2006)
15. Shahrokhi, F., Sýkora, O., Székely, L.A., Vrto, I.: On bipartite drawings and the linear arrangement problem. SIAM Journal on Computing 30(6), 1773–1789 (2001)
16. Suderman, M.: Layered Graph Drawing. PhD thesis, School of Computer Science, McGill University Montréal (2005)
17. Sugiyama, K., Tagawa, S., Toda, M.: Methods for visual understanding of hierarchical system structures. IEEE Transactions on Systems, Man and Cybernetics 11(2), 109–125 (1981)
18. Tsiaras, V., Triantafilou, S., Tollis, I.G.: DAGmaps: Space filling visualization of directed acyclic graphs. Journal of Graph Algorithms and Applications 13(3), 319–347 (2009)
19. Waterman, M.S., Griggs, J.R.: Interval graphs and maps of DNA. Bulletin of Mathematical Biology 48(2), 189–195 (1986)

A Categorical View of Timed Weak Bisimulation*

Natalya Gribovskaya[1] and Irina Virbitskaite[1,2]

[1] A.P. Ershov Institute of Informatics Systems, SB RAS
6, Acad. Lavrentiev av., Novosibirsk, 630090, Russia
[2] Novosibirsk State University
2, Pirogova st., Novosibirsk, 630090, Russia

Abstract. Timed transition systems are a widely studied model for real-time systems. The intention of the paper is twofold: first to show the applicability of the general categorical framework of open maps in order to prove that timed weak bisimulation is indeed an equivalence relation, and, second, to investigate how several categorical (open maps, path-bisimilarity and coalgebra based) approaches to an abstract characterization of bisimulation relate to each other and to timed weak bisimulation, in the setting of timed transition systems.

1 Introduction

Monitoring real-time concurrent systems is a challenging problem. In order to cope with the problem, different formal models for real-time systems have been put forward. Over the last two decades, much of the theory of observational equivalences of models has been lifted to real-time settings (see [2,3,6,9,15,29] among others). Klusener [19] was the first who extended bisimulations with silent step (weak, delay and branching bisimulations) to the setting with time. A key property of a semantics is that it is an equivalence. In general for concurrency semantics in the presence of silent step, reflexivity and symmetry are easy to see, but transitivity is much more difficult. In a setting with time, proving equivalence of a concurrency semantics becomes more complicated, compared to untimed case. Still, equivalence properties for timed semantics are often claimed, but hardly even proved (see [4,19,32] among others). In the paper [13] it has been shown that timed branching bisimulation as defined by van der Zwaag [32] does not constitute an equivalence relation, in case of a dense time domain. The authors of [13] have proposed an adaptation and proved that the resulting timed branching bisimulation is an equivalence indeed.

In an attempt to explain and unify apparent differences between the extensive amount of research within the field of untimed behavioural equivalences, several category-theoretic approaches to the matter have appeared (see [27,17] among others). One of them was initiated by Joyal, Nielsen, and Winskel in [17] where

* This work is supported in part by DFG-RFBR (grant No 436 RUS 113/1002/01, grant No 09-01-91334).

© Springer-Verlag Berlin Heidelberg 2010

they have proposed abstract ways of capturing the notion of bisimulation through open maps based bisimilarity and its logical counterpart — path bisimilarity. As shown in [8,17,23,24], bisimilarity induced by open maps makes possible a uniform definition of the numerous suggested behavioural equivalences (e.g., trace and testing equivalences, bisimulation, barbed and weak bisimulations, strong history preserving bisimulation, etc.) across a wide range of models for concurrency (e.g., transition systems, event structures, Petri nets, higher dimensional automata, etc.). The situation is less settled in the case of real-time models. In [16] and [31], the open maps based approach has been applied to provide an abstract characterization of interleaving bisimulation on timed transition systems and of partial order based equivalences on timed event structures, respectively. The general categorical framework of open maps has been used in [14] to prove that timed delay equivalence is indeed an equivalence relation in the setting of timed transition systems with invariants.

Another way to provide categorical characterizations is to adopt the coalgebraic approach. During the last years, it is becoming increasingly clear that a great variety of state-based dynamical systems, like transition systems, automata, process calculi and class-based systems can be captured uniformly as coalgebras. There is also a coalgebraic notion of bisimulation, the research in this area has been initiated by Aczel and Mendler [1]. The papers [20,21,22,28,25,26] have provided a coalgebraic rendering of various behavioural equivalences in untimed settings. For discrete and continuous stochastic systems, researchers have investigated coalgebras for probability distribution functors on categories of metric or measurable spaces (see [7,12,30] among others). The paper [18] has given a coalgebraic formulation of timed processes and their operational semantics, where time is modelled by a monoid called a time domain, and processes are modelled by timed transition systems, which amount to coalgebras for an evolution comonad generated by the time domain.

The contribution of the paper is twofold: first to show the applicability of the general framework of open maps in order to prove that timed weak bisimulation is indeed an equivalence relation, and, second, to investigate how several categorical (open maps, path-bisimilarity and coalgebra based) approaches to an abstract characterization of bisimulation relate to each other and to timed weak bisimulation, in the setting of timed transition systems.

The rest of the paper is organized as follows. The basic notions and notations related to the structure and behaviour of timed transition systems with invariants are introduced in section 2. In the next section, we define a notion of weak bisimulation and give its alternative characterization. A category of timed transition systems is introduced in section 4. An open maps based characterization of timed weak bisimulation is given in section 5. In the subsequent section, we show how the equivalences under consideration can be captured by another category-theoretic bisimulation — path-bisimulation. In section 7, a coalgebraic formulation of the equivalences is treated. Section 8 contains conclusion and some remarks on future work. Proofs are omitted because of space limitations and can be found in a forthcoming paper.

2 Timed Transition Systems

In this section, we introduce some basic notions and notations concerning timed transition systems [16] and their behaviour.

Let $\mathbf{R}$ be the set of non-negative reals and $\mathbf{R}^+$ the set of positive reals. Also, let Σ be a finite alphabet of actions without the *silent* action τ, and $\Sigma_\tau = \Sigma \cup \{\tau\}$. A *timed word over* $\Sigma_\tau \cup \mathbf{R}^+$ is a finite sequence of the form: $\alpha = \sigma_1 \ldots \sigma_n$ such that $\sigma_i \in \Sigma_\tau \cup \mathbf{R}^+$ $(1 \leq i \leq n)$. We consider a finite set V of clock variables. A *clock valuation over* V is a mapping $\nu : V \to \mathbf{R}$ which assigns time values to the clock variables. Define $(\nu + c)(x) := \nu(x) + c$ for all clock variables $x \in V$. For a subset λ of clock variables, we shall write $\nu[\lambda \to 0](x) := 0$, if $x \in \lambda$, and $\nu[\lambda \to 0](x) := \nu(x)$, otherwise. Given a set V, we define the set $\Delta(V)$ of *clock constraints* by the following grammar: $\delta := c \ \# \ x \mid x + c \ \# \ y \mid \delta \wedge \delta$, where $\# \in \{\leq, <, \geq, >, =\}$, c is a real valued constant and x, y are clock variables from V. We shall say that a clock constraint δ is *satisfied by a clock valuation* ν if the expression $\delta[\nu(x)/x]$[1] evaluates to true. A clock constraint δ defines a subset of $\mathbf{R}^m$ (m is the number of clock variables in V). We call the subset as the meaning of δ and denote it as $\|\delta\|_V$. A clock valuation ν defines a point in $\mathbf{R}^m$ (denoted $\|\nu\|_V$). So, the clock constraint δ is satisfied by the clock valuation ν iff $\|\nu\|_V \in \|\delta\|_V$.

We are now prepared to consider the definition of timed transition systems.

Definition 1. *A* timed transition system $\mathcal{T}$ *is a sextuple* $(S, s_0, \Sigma_\tau, V, T, I)$ *where* S *is a set of states and* s_0 *is the initial state,* Σ_τ *is a finite alphabet of actions,* V *is a set of clock variables,* $T \subseteq S \times \Sigma_\tau \times \Delta(V) \times 2^V \times S$ *is a set of transitions,* $I \in \Delta(V)^S$ *assigns to each state an invariant given by the same syntax as a clock constraint. For each transition* $(s, a, \delta, \lambda, s')$*, if* $a = \tau$*, then* $\lambda = \emptyset$*. We shall write* $s \underset{\delta,\ \lambda}{\xrightarrow{\sigma}} s'$ *to denote a transition* $(s, \sigma, \delta, \lambda, s')$*.*

Define the behaviour of timed transition systems.

Definition 2. *Let* $\mathcal{T} = (S, s_0, \Sigma_\tau, V, T, I)$ *be a timed transition system.*

A configuration *of* $\mathcal{T}$ *is a pair* $\langle s, \nu \rangle$*, where* s *is a state and* ν *is a clock valuation. The set of configurations of* $\mathcal{T}$ *is denoted as* $Conf(\mathcal{T})$*.*

For $\langle s, \nu \rangle$, $\langle s', \nu' \rangle \in Conf(\mathcal{T})$ *and* $\sigma \in \Sigma_\tau \cup \mathbf{R}^+$, $\langle s, \nu \rangle \xrightarrow{\sigma} \langle s', \nu' \rangle$ *is defined as follows: 1) if* $\sigma \in \Sigma_\tau$*, then there is a transition* $s \underset{\delta,\lambda}{\xrightarrow{\sigma}} s'$ *such that* $\|\nu\|_V \in \|\delta\|_V$, $\nu' = \nu[\lambda \to 0]$ *and* $\|\nu'\|_V \in \|I(s')\|_V$*, 2) if* $\sigma \in \mathbf{R}^+$*, then* $s = s'$, $\nu' = \nu + \sigma$ *and* $\forall 0 < \sigma' \leq \sigma \diamond \|\nu + \sigma'\|_V \in \|I(s)\|_V$*.*

A run *of* $\mathcal{T}$ *is a sequence* $\langle s_0, \nu_0 \rangle \xrightarrow{\sigma_1} \langle s_1, \nu_1 \rangle \ldots \langle s_{n-1}, \nu_{n-1} \rangle \xrightarrow{\sigma_n} \langle s_n, \nu_n \rangle$*, where* s_0 *is the initial state,* ν_0 *is the constant 0 function and* $\sigma_i \in \Sigma_\tau \cup \mathbf{R}^+$ $(i = 1..n)$*. A run as above is said to* generate the timed word $\alpha = \sigma_1 \ldots \sigma_n$*. A configuration* $\langle s, \nu \rangle$ *of* $\mathcal{T}$ *is called* reachable *iff* $\mathcal{T}$ *has a run with an occurrence of* $\langle s, \nu \rangle$*. The set of reachable configurations of* $\mathcal{T}$ *is denoted by* $\mathcal{RC}(\mathcal{T})$*.*

[1] $\delta[y/x]$ is the substitution of y for x in δ.

In the following we shall use some auxiliary notations. Let $a \in \Sigma$ and $d \in \mathbf{R}^+$. For $\langle s, \nu\rangle, \langle s', \nu'\rangle \in Conf(\mathcal{T})$, we shall write:

- $\langle s, \nu\rangle \overset{\epsilon}{\Rightarrow} \langle s', \nu'\rangle$ iff $\langle s, \nu\rangle \xrightarrow{\tau} \langle s_1, \nu_1\rangle \cdots \langle s_{n-1}, \nu_{n-1}\rangle \xrightarrow{\tau} \langle s_n, \nu_n\rangle = \langle s', \nu'\rangle$, where $n \geq 0$,
- $\langle s, \nu\rangle \overset{\epsilon}{\Rightarrow} \xrightarrow{d} \overset{\epsilon}{\Rightarrow} \langle s', \nu'\rangle$ iff $\langle s, \nu\rangle \overset{\epsilon}{\Rightarrow} \xrightarrow{d_1} \overset{\epsilon}{\Rightarrow} \langle s_1, \nu_1\rangle \cdots \langle s_{n-1}, \nu_{n-1}\rangle \overset{\epsilon}{\Rightarrow} \xrightarrow{d_n} \overset{\epsilon}{\Rightarrow}$ $\langle s_n, \nu_n\rangle = \langle s', \nu'\rangle$, where $n \geq 1$ and $d = d_1 + \cdots + d_n$,
- $\langle s, \nu\rangle \overset{a}{\Rightarrow} \langle s', \nu'\rangle$ iff $\langle s, \nu\rangle \overset{\epsilon}{\Rightarrow} \xrightarrow{a} \overset{\epsilon}{\Rightarrow} \langle s', \nu'\rangle$,
- $\langle s, \nu\rangle \overset{d}{\Rightarrow} \langle s', \nu'\rangle$ iff $\langle s, \nu\rangle \overset{\epsilon}{\Rightarrow} \xrightarrow{d} \overset{\epsilon}{\Rightarrow} \langle s', \nu'\rangle$,
- for $\sigma \in \Sigma_\tau \cup \mathbf{R}^+$, we define $\langle s, \nu\rangle \widehat{\overset{\sigma}{\Rightarrow}} \langle s', \nu'\rangle$ as follows:
  - $\langle s, \nu\rangle \widehat{\overset{\tau}{\Rightarrow}} \langle s', \nu'\rangle$ iff $\langle s, \nu\rangle \overset{\epsilon}{\Rightarrow} \langle s', \nu'\rangle$,
  - $\langle s, \nu\rangle \widehat{\overset{a}{\Rightarrow}} \langle s', \nu'\rangle$ iff $\langle s, \nu\rangle \overset{a}{\Rightarrow} \langle s', \nu'\rangle$,
  - $\langle s, \nu\rangle \widehat{\overset{d}{\Rightarrow}} \langle s', \nu'\rangle$ iff $\langle s, \nu\rangle \overset{d}{\Rightarrow} \langle s', \nu'\rangle$.

A *weak run* of $\mathcal{T}$ is a sequence $\langle s_0, \nu_0\rangle \widehat{\overset{\sigma_1}{\Rightarrow}} \langle s_1, \nu_1\rangle \ldots \langle s_{n-1}, \nu_{n-1}\rangle \widehat{\overset{\sigma_n}{\Rightarrow}} \langle s_n, \nu_n\rangle$, where s_0 is the initial state, ν_0 is the constant 0 function and $\sigma_1, .., \sigma_n \in \Sigma_\tau \cup \mathbf{R}^+$. A weak run as above is said to *generate the timed word* $\widehat{\sigma_1} \ldots \widehat{\sigma_n}$ over $\Sigma \cup \mathbf{R}^+$.

3 Timed Weak Bisimulation

In this section we define the concept of timed weak bisimulation and give its alternative characterization.

Definition 3. *Given timed transition systems* $\mathcal{T} = (S, s_0, \Sigma_\tau, V, T, I)$ *and* $\mathcal{T}' = (S', s'_0, \Sigma_\tau, V', T', I')$, *a relation* $\mathcal{B} \subseteq \mathcal{RC}(\mathcal{T}) \times \mathcal{RC}(\mathcal{T}')$ *is a* timed weak bisimulation *if* $(\langle s_0, \nu_0\rangle, \langle s'_0, \nu'_0\rangle) \in \mathcal{B}$ *and for any* $(\langle s, \nu\rangle, \langle s', \nu'\rangle) \in \mathcal{B}$ *it holds:*

1. *if* $\langle s, \nu\rangle \xrightarrow{\sigma} \langle s_1, \nu_1\rangle$ $(\sigma \in \Sigma_\tau \cup \mathbf{R}^+)$, *then*
   (a) *either* $\sigma = \tau$ *and* $(\langle s_1, \nu_1\rangle, \langle s', \nu'\rangle) \in \mathcal{B}$,
   (b) *or* $\langle s', \nu'\rangle \overset{\epsilon}{\Rightarrow} \xrightarrow{\sigma} \overset{\epsilon}{\Rightarrow} \langle s'_1, \nu'_1\rangle$ *and* $(\langle s_1, \nu_1\rangle, \langle s'_1, \nu'_1\rangle) \in \mathcal{B}$ *for some* $\langle s'_1, \nu'_1\rangle$,
2. *if* $\langle s', \nu'\rangle \xrightarrow{\sigma} \langle s'_1, \nu'_1\rangle$ $(\sigma \in \Sigma_\tau \cup \mathbf{R}^+)$, *then*
   (a) *either* $\sigma = \tau$ *and* $(\langle s, \nu\rangle, \langle s'_1, \nu'_1\rangle) \in \mathcal{B}$,
   (b) *or* $\langle s, \nu\rangle \overset{\epsilon}{\Rightarrow} \xrightarrow{\sigma} \overset{\epsilon}{\Rightarrow} \langle s_1, \nu_1\rangle$ *and* $(\langle s_1, \nu_1\rangle, \langle s'_1, \nu'_1\rangle) \in \mathcal{B}$ *for some* $\langle s_1, \nu_1\rangle$.

$\mathcal{T}_1$ *and* $\mathcal{T}_2$ *are called* timed weak bisimilar *if there exists a timed weak bisimulation between the sets of their reachable configurations.*

Example 1. To illustrate the concept consider the timed transition systems shown in Fig. 1. The timed transition systems $\check{\mathcal{T}}$ and $\dot{\mathcal{T}}$ are timed weak bisimilar, whereas the timed transition systems $\widehat{\mathcal{T}}$ and $\widetilde{\mathcal{T}}$ are not because in $\widetilde{\mathcal{T}}$ after passing 5 time units an execution of a is possible, but it is not the case in $\widehat{\mathcal{T}}$.

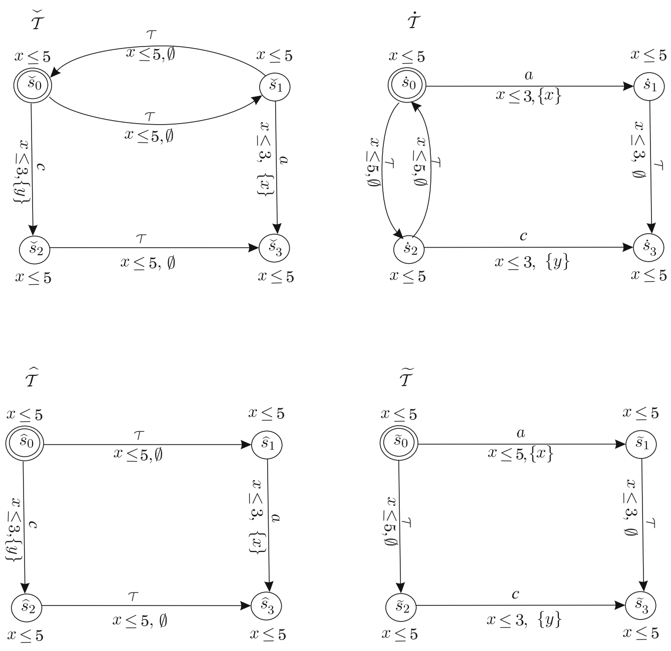


Fig. 1.

We give another characterization of timed weak bisimulation which is more convenient to work with.

Lemma 1. *Given timed transition systems $\mathcal{T}$, $\mathcal{T}'$ and a relation $\mathcal{B} \subseteq \mathcal{RC}(\mathcal{T}) \times \mathcal{RC}(\mathcal{T}')$, $\mathcal{B}$ is a timed weak bisimulation iff $(\langle s_0, \nu_0 \rangle, \langle s_0', \nu_0' \rangle) \in \mathcal{B}$ and for any $(\langle s, \nu \rangle, \langle s', \nu' \rangle) \in \mathcal{B}$ it holds:*

(i) if $\langle s, \nu \rangle \xrightarrow{\sigma} \langle s_1, \nu_1 \rangle$ ($\sigma \in \Sigma_\tau \cup \mathbf{R}^+$),

then $\langle s', \nu' \rangle \overset{\widehat{\sigma}}{\Rightarrow} \langle s_1', \nu_1' \rangle$ and $(\langle s_1, \nu_1 \rangle, \langle s_1', \nu_1' \rangle) \in \mathcal{B}$ for some $\langle s_1', \nu_1' \rangle$,

(ii) if $\langle s', \nu' \rangle \xrightarrow{\sigma} \langle s_1', \nu_1' \rangle$ ($\sigma \in \Sigma_\tau \cup \mathbf{R}^+$),

then $\langle s, \nu \rangle \overset{\widehat{\sigma}}{\Rightarrow} \langle s_1, \nu_1 \rangle$ and $(\langle s_1, \nu_1 \rangle, \langle s_1', \nu_1' \rangle) \in \mathcal{B}$ for some $\langle s_1, \nu_1 \rangle$.

4 A Category of Timed Transition Systems

In this section we first define a category of timed transition systems and then show that it has pullbacks. For this purpose, we start with a notion of a morphism.

Definition 4. *Given timed transition systems $\mathcal{T} = (S, s_0, \Sigma_\tau, V, T, I)$ and $\mathcal{T}' = (S', s_0', \Sigma_\tau, V', T', I')$, a pair (μ, η) is a morphism between $\mathcal{T}$ and $\mathcal{T}'$, if $\mu : S \to S'$ is a mapping between the states and $\eta : V' \to V$ is a mapping between the clock variables, satisfying the following conditions:*

– $\langle \mu(s_0), \eta^{-1}(\nu_0) \rangle$ *is the initial configuration of* $\mathcal{T}'$,
– *if* $\langle s, \nu \rangle \in \mathcal{RC}(\mathcal{T})$ *and* $\langle s, \nu \rangle \xrightarrow{\sigma} \langle s_1, \nu_1 \rangle$ $(\sigma \in \Sigma_\tau \cup \mathbf{R}^+)$ *in* $\mathcal{T}$, *then*

  $\langle \mu(s), \eta^{-1}(\nu) \rangle \in \mathcal{RC}(\mathcal{T}')$ *and* $\langle \mu(s), \eta^{-1}(\nu) \rangle \overset{\widehat{\sigma}}{\Rightarrow} \langle \mu(s_1), \eta^{-1}(\nu_1) \rangle$ *in* $\mathcal{T}'$.

  Here, for a function $\eta : V' \to V$ *and a clock evaluation* $\nu : V \to \mathbf{R}$, *we define* $\eta^{-1}(\nu) : V' \to \mathbf{R}$, *the inverse image of* ν *under* η, *as* $\eta^{-1}(\nu)(x) := \nu(\eta(x))$.

Notice, the morphisms defined prior to that represent some notions of simulation of the behaviour of one system by the other morphisms, with an accuracy of observable actions.

Timed transition systems with alphabet Σ_τ and morphisms between them form a category of timed transition systems, **CTTS**, in which the composition of two morphisms $(\mu, \eta) : \mathcal{T} \longrightarrow \mathcal{T}'$ and $(\mu', \eta') : \mathcal{T}' \longrightarrow \mathcal{T}''$ is defined as $(\mu', \eta') \circ (\mu, \eta) := (\mu' \circ \mu, \eta \circ \eta')$, and the identity morphism is the morphism where both μ and η are the identity functions.

The category **CTTS** has a number of useful properties. For our purpose here we only need the following.

Theorem 1. CTTS *has pullbacks.*

5 Open Maps Bisimulation

In this section, we provide an abstract characterization of timed weak bisimulation in terms of open morphisms.

We start with defining the terminology from [17]. First, a category $\mathbb{M}$ whose objects represent models of computations has to be identified. Inside the category $\mathbb{M}$, we have to choose a subcategory of 'path objects' and 'path extension' morphisms between these objects. The *path subcategory* is denoted by $\mathbb{P}$. Given a path object P in $\mathbb{P}$ and a model object X in $\mathbb{M}$, a *path* is a morphism $p : P \longrightarrow X$ in $\mathbb{M}$. We think of p as representing a particular way of realizing P in X.

Second, we identify morphisms $m : X \longrightarrow Y$ which have the property that whenever a computation of X can be extended via m in Y then that extension can be matched by an extension of the computation in X. A morphism $m : X \to Y$ in $\mathbb{M}$ is called $\mathbb{P}$-*open* if whenever $f : P_1 \to P_2$ in $\mathbb{P}$, $p : P_1 \to X$ and $q : P_2 \to Y$ in $\mathbb{M}$, and $m \circ p = q \circ f$, there exists a morphism $h : P_2 \to X$ in $\mathbb{M}$ such that $p = h \circ f$ and $q = m \circ h$.

Third, an abstract notion of bisimilarity is introduced. The definition is given in terms of spans of open maps. Two objects X and Y in $\mathbb{M}$ are said to be $\mathbb{P}$-*bisimilar* if there exists a span $X \xleftarrow{m} Z \xrightarrow{m'} Y$ with a common object Z of $\mathbb{P}$-open morphisms. Clearly, the relation of $\mathbb{P}$-bisimilarity between objects is reflexive (identities are $\mathbb{P}$-open) and symmetric (in the nature of spans). It is also transitive provided $\mathbb{M}$ has pullbacks, and so always an equivalence relation on objects, by virtue of the following fact:

Proposition 1. *([17]) Pullbacks of* $\mathbb{P}$*-open morphisms are* $\mathbb{P}$*-open.*

A category **CTTS** of timed transition systems has been identified in the previous section. Now a path subcategory should be chosen. For this purpose, we need to define the construction below.

Consider an arbitrary timed word $\alpha = d_{0,1} \ldots d_{0,n_0}\ \sigma_1\ d_{1,1} \ldots d_{1,n_1} \ldots d_{n-1,1} \ldots d_{n-1,n_{n-1}}\ \sigma_n\ d_{n,1} \ldots d_{n,n_n}$ with $d_{i,j} \in \mathbf{R}^+$ and $\sigma_k \in \Sigma_\tau$, where $n \geq 0$, $n_i \geq 0$, $i = 0..n$, $j = 1..n_i$ and $k = 1..n$. For α, define a function γ as follows: $\gamma(\alpha) = \sum_{j=1}^{n_0} d_{0,j}\ \sigma_1 \sum_{j=1}^{n_1} d_{1,j} \ldots \sum_{j=1}^{n_{n-1}} d_{n-1,j}\ \sigma_n \sum_{j=1}^{n_n} d_{n,j}$. Furthermore, for a timed word α, we shall use the set $\Gamma(\alpha) = \{\alpha' \mid \gamma(\alpha') = \gamma(\alpha)\}$.

Definition 5. *Let α be a timed word with $\gamma(\alpha)=\sum_{j=1}^{n_0} d_{0,j}\ \sigma_1 \sum_{j=1}^{n_1} d_{1,j} \ldots \sum_{j=1}^{n_{n-1}} d_{n-1,j}$ $\sigma_n \sum_{j=1}^{n_n} d_{n,j}$. We define a timed transition system $\mathcal{T}^{\Gamma(\alpha)} = (S^{\Gamma(\alpha)}, s_0^{\Gamma(\alpha)}, \Sigma_\tau, V^{\Gamma(\alpha)}, T^{\Gamma(\alpha)}, I^{\Gamma(\alpha)})$, corresponding to $\Gamma(\alpha)$, as follows:*

- $S^{\Gamma(\alpha)} = \{0, 1, \ldots, n\}$,
- $s_0^{\Gamma(\alpha)} = 0$,
- *$V^{\Gamma(\alpha)}$ consists of the $2^{|A^{\Gamma(\alpha)}|}$ subsets of the states $A^{\Gamma(\alpha)} = \{i \mid 1 \leq i \leq n, \sigma_i \neq \tau\}$,*
- *$T^{\Gamma(\alpha)} = \{(i-1, \sigma_i, \delta_i, \lambda_i, i) \mid 1 \leq i \leq n,\ \lambda_i := \{x \in V^{\Gamma(\alpha)} \mid i \in x\}, \delta_i := \bigwedge_{x \in V^{\Gamma(\alpha)}} (x = \sum_{l=Reset(i,x)}^{i-1} \sum_{j=1}^{n_l} d_{l,j})\}$. Here, $Reset(i,x) = max\{0, k \mid k < i \wedge k \in x\}$, i.e. Reset returns the last state at which x was reset,*
- *$I^{\Gamma(\alpha)}$ is inductively defined as follows. The invariant on the state 0 is $\bigwedge_{x \in V^{\Gamma(\alpha)}} (0 \leq x \leq \sum_{j=1}^{n_0} d_{0,j})$. Assume the invariant on the state $i-1$ is $\bigwedge_{x \in V^{\Gamma(\alpha)}} (c(i-1,x) \leq x \leq \widehat{c}(i-1,x))$. Then the invariant on the state i is $\bigwedge_{x \in V^{\Gamma(\alpha)}}$ (if $i \in x$ then $(0 \leq x \leq \sum_{j=1}^{n_i} d_{i,j})$, else $(\widehat{c}(i-1,x) \leq x \leq \widehat{c}(i-1,x) + \sum_{j=1}^{n_i} d_{i,j})$.*

The class of timed transition systems of the form $\mathcal{T}^{\Gamma(\alpha)}$ is denoted as $\mathcal{TTS}_{\Sigma_\tau}^{\Gamma(\alpha)}$.

Consider a helpful property of a timed transition system $\mathcal{T}^{\Gamma(\alpha)}$.

Lemma 2. *Let $\alpha' \in \Gamma(\alpha)$ be a timed word with $\gamma(\alpha') = \sum_{j=1}^{n_0} d_{0,j}\ \sigma_1 \sum_{j=1}^{n_1} d_{1,j} \ldots \sum_{j=1}^{n_{n-1}} d_{n-1,j}\ \sigma_n \sum_{j=1}^{n_n} d_{n,j}$. Then, there exists a run $\langle s_0, \nu_0 \rangle \xrightarrow{d_{0,1}} \ldots \xrightarrow{d_{0,n_0}} \langle s_0, \nu_0^{n_0} \rangle \xrightarrow{\sigma_1} \langle s_1, \nu_1 \rangle \ldots \langle s_{n-1}, \nu_{n-1} \rangle \xrightarrow{d_{n-1,1}} \ldots \xrightarrow{d_{n-1,n_{n-1}}} \langle s_{n-1}, \nu_{n-1}^{n_{n-1}} \rangle \xrightarrow{\sigma_n} \langle s_n, \nu_n \rangle \xrightarrow{d_{n,1}} \ldots \xrightarrow{d_{n,n_n}} \langle s_n, \nu_n^{n_n} \rangle$ in $\mathcal{T}^{\Gamma(\alpha)}$.*

With respect to a set of actions Σ_τ, let $\mathbf{P}$ denote the subcategory of the category **CTTS** with objects from $\mathcal{TTS}_{\Sigma_\tau}^{\Gamma(\alpha)}$ and morphisms of the form $(\mu, \eta) : \mathcal{T}^{\Gamma(\alpha)} \to$

$\mathcal{T}^{\Gamma(\alpha')}$, where α' is an extension of α, $\mu(i) = i$ ($i \in S^{\Gamma(\alpha)}$) and $\eta(x) = \{i \in A^{\Gamma(\alpha)} \mid i \in x\}$ ($x \in V_{\mathcal{T}^{\Gamma(\alpha')}}$). Notice, the morphisms in $\mathbf{P}$ simulate actions from Σ_τ in the strong sense, i.e. no additional τ's are assumed. Allowing arbitrary morphisms between the objects in $\mathbf{P}$ violates correctness of Theorem 3.

Using Theorem 1 and Proposition 1, we get the following fact.

Theorem 2. $\mathbf{P}$-*bisimulation is an equivalence relation.*

Next, we provide a behavioural characterization of $\mathbf{P}$-open morphisms which is crucial to establish the coincidence of $\mathbf{P}$-bisimilarity and timed weak bisimulation.

Theorem 3. *Given timed transition systems $\mathcal{T}$ and $\mathcal{T}'$, a morphism $(\mu, \eta) : \mathcal{T} \to \mathcal{T}'$ is $\mathbf{P}$-open iff for any $\langle s, \nu \rangle \in \mathcal{RC}(\mathcal{T})$, whenever $\langle \mu(s), \eta^{-1}(\nu) \rangle \xrightarrow{\sigma} \langle s'', \nu'' \rangle$ ($\sigma \in \Sigma_\tau \cup \mathbf{R}^+$) in $\mathcal{T}'$, then $\langle s, \nu \rangle \widehat{\overset{\sigma}{\Rightarrow}} \langle s', \nu' \rangle$ in $\mathcal{T}$ and $\langle \mu(s'), \eta^{-1}(\nu') \rangle = \langle s'', \nu'' \rangle$.*

At last, the coincidence of $\mathbf{P}$-bisimilarity and timed weak bisimulation can be established.

Theorem 4. *Two timed transition systems $\mathcal{T}_1$ and $\mathcal{T}_2$ are $\mathbf{P}$-bisimilar iff they are timed weak bisimilar.*

Corollary 1. *Timed weak bisimulation is an equivalence relation.*

6 Path-Bisimulation

To obtain a logic characteristic of bisimulation induced by open maps, Joyal, Nielsen, and Winskel [17] have proposed a second category-theoretic characterization of bisimulation — path bisimulation which is a relation based generalization of open maps bisimulation.

Definition 6. *Let $\mathbb{M}$ be a category of models, let $\mathbb{P}$ be a small category of path objects, where $\mathbb{P}$ is a subcategory of $\mathbb{M}$, let I be a common initial object[2] of $\mathbb{P}$ and $\mathbb{M}$. Then,*

1) Two objects X_1 and X_2 of $\mathbb{M}$ are called path-$\mathbb{P}$-bisimilar iff there is a set R of pairs (p_1, p_2) of paths with common domain P, so $p_1 : P \to X_1$ is a path in X_1 and $p_2 : P \to X_2$ is a path in X_2, such that

(o) $(i_1, i_2) \in R$, where $i_1 : I \to X_1$ and $i_2 : I \to X_2$ are the unique paths starting in the initial object, and for all $(p_1, p_2) \in R$ and for all $m : P \to Q$, where m is in $\mathbb{P}$, holds

(i) if there exists $q_1 : Q \to X_1$ with $q_1 \circ m = p_1$ then there exists $q_2 : Q \to X_2$ with $q_2 \circ m = p_2$ and $(q_1, q_2) \in R$ and

(ii) if there exists $q_2 : Q \to X_2$ with $q_2 \circ m = p_2$ then there exists $q_1 : Q \to X_1$ with $q_1 \circ m = p_1$ and $(q_1, q_2) \in R$.

[2] In the cases when $\mathbb{P}$ is $\mathbf{P}$ and $\mathbb{M}$ is $\mathbf{CTTS}$, the initial object I is the timed transition system $\mathcal{E} = (\{s_0\}, s_0, \Sigma_\tau, \{x\}, \emptyset, I(s_0) = (x = 0))$.

2) Two objects X_1 and X_2 are strong path-$\mathbb{P}$-bisimilar *iff they are path-$\mathbb{P}$-bisimilar and the set R further satisfies:*
(iii) If $(q_1, q_2) \in R$, with $q_1 : Q \to X_1$ and $q_2 : Q \to X_2$ and $m : P \to Q$, where m is in $\mathbb{P}$, then $(q_1 \circ m, q_2 \circ m) \in R$.

Theorem 5. $\mathbf{P}$*-bisimilarity, path-*$\mathbf{P}$*-bisimilarity, strong path-*$\mathbf{P}$*-bisimilarity all coincide with timed weak bisimulation.*

7 Coalgebraic Bisimulation

Another alternative abstract characterization of bisimulation is based on a category of coalgebras induced by an endofunctor on an arbitrary category. In [20] it has been shown that the concept of path-bisimilarity can be translated into a coalgebraic setting with lax cohomomorphisms. Notice that in [21] a coalgebraic characterization of path-bisimilarity is obtained without the use of lax notions, however, in this case one cannot define a functor from a category of computations to the category of coalgebras.

We start with defining the terminology from [20].

Let $\mathbb{M}$ be a locally small category and $\mathbb{P}$ a small path subcategory of $\mathbb{M}$. We will define an embedding of $\mathbb{M}$ into a category of coalgebras for some endofunctor on the category $Set^{|\mathbb{P}|}$ of $|\mathbb{P}|$-sorted sets, where $|\mathbb{P}|$ is a set of objects in $\mathbb{P}$. An endofunctor $F_{\mathbb{P}} : Set^{|\mathbb{P}|} \longrightarrow Set^{|\mathbb{P}|}$ is defined as follows:

$$\{X_P\}_{P\in|\mathbb{P}|} \mapsto \{ \prod_{Q\in|\mathbb{P}|} (\mathcal{P}(X_Q))^{Hom_{\mathbb{P}}(P,Q)} \}_{P\in|\mathbb{P}|},$$

where for $P \in |\mathbb{P}|$ X_P denotes a component of a $|\mathbb{P}|$-sorted set X, and $Hom_{\mathbb{P}}(P, Q)$ stands for the set of all morphisms from P to Q in $\mathbb{P}$. A *coalgebra for $F_{\mathbb{P}}$* or *$F_{\mathbb{P}}$-coalgebra* is a pair (S, tr) with S an object in $Set^{|\mathbb{P}|}$ and $tr : S \to F_{\mathbb{P}}(S)$ a morphism in $Set^{|\mathbb{P}|}$, which consists of a family of functions:

$$\{tr_P : S_P \to \prod_{Q\in|\mathbb{P}|} (\mathcal{P}(S_Q))^{Hom_{\mathbb{P}}(P,Q)}\}_{P\in|\mathbb{P}|}.$$

The set S is called the *carrier* and the function tr is called the *coalgebra structure* of the $F_{\mathbb{P}}$-coalgebra.

A morphism $\gamma : S_1 \to S_2$ in $Set^{|\mathbb{P}|}$ is called a *cohomomorphism* between $F_{\mathbb{P}}$-coalgebras (S_1, tr_1) and (S_2, tr_2) iff $F_{\mathbb{P}}(\gamma) \circ tr_1 = tr_2 \circ \gamma$, where $F_{\mathbb{P}}(\gamma)$ is defined as follows: $(F_{\mathbb{P}}(\gamma))_P = \prod_{Q\in|\mathbb{P}|} h_Q$, where $h_Q : g \longmapsto f$, $f(m) = \{\gamma_Q(x) | x \in g(m)\}$, for all $m \in Hom_{\mathbb{P}}(P, Q)$. $F_{\mathbb{P}}$-coalgebras and cohomomorphisms between them constitute a category, denoted by $\mathcal{CA}_{\mathbb{P}}$.

From now on, for an $F_{\mathbb{P}}$-*coalgebra* (S, tr), a triple $\langle m_1, m, m_2\rangle$, where $m_1 \in S_P$, $m_2 \in S_Q$ and $m \in Hom_{\mathbb{P}}(P, Q)$, satisfying $m_2 \in tr_P(m_1)(m)$ will be denoted by $m_1 \xrightarrow{m} m_2$.

As usual in the theory of coalgebras, bisimulation is a relation represented by a span of coalgebra morphisms [27]. An $F_{\mathbb{P}}$-bisimulation between two coalgebras (S_1, tr_1) and (S_2, tr_2) is a $|\mathbb{P}|$-sorted relation $R = \{R_P\}_{P\in|\mathbb{P}|} \subseteq (S_1 \times S_2)$ such that, if $(m_1, m_2) \in R_P$ and $m : P \to Q$ in $\mathbb{P}$, then

– if $m_1 \xrightarrow{m} m'_1$, then $m_2 \xrightarrow{m} m'_2$ and $(m'_1, m'_2) \in R_Q$ for some $m'_2 \in S_2$,
– if $m_2 \xrightarrow{m} m'_2$, then $m_1 \xrightarrow{m} m'_1$ and $(m'_1, m'_2) \in R_Q$ for some $m'_1 \in S_2$.

Clearly, each $F_{\mathbb{P}}$-bisimulation has a coalgabra structure $tr_R : R \to F_{\mathbb{P}}(R)$ and together with the projections $\pi_1 : R \to S_1$ and $\pi_2 : R \to S_2$ form a span of cohomomorphisms of the $F_{\mathbb{P}}$-coalgebra.

Next, following [11,20], we relax the requirement on coalgebra morphism. A morphism $\gamma : S \to S'$ in $Set^{|\mathbb{P}|}$ is called a *lax cohomomorphism* between $F_{\mathbb{P}}$-coalgebras (S, tr) and (S', tr') if for each $m \in Hom_{\mathbb{P}}(P, Q)$ and $s \in S_P$, $\{f_Q(r) \ : \ r \in t_P(s)(m)\} \subseteq tr'_P(f_P(s))(m)$. $F_{\mathbb{P}}$-coalgebras and lax cohomomorphisms constitute a category, denoted by $\mathcal{CA}^{lax}_{\mathbb{P}}$.

For $\mathbb{M}$, define a functor $\mathcal{B}eh^{\mathbb{M}}_{\mathbb{P}} : \mathbb{M} \to \mathcal{CA}^{lax}_{\mathbb{P}}$. $\mathcal{B}eh^{\mathbb{M}}_{\mathbb{P}}$ acts on objects X in $\mathbb{M}$ as follows: $\{Hom_{\mathbb{M}}(P, X)\}_{P \in |\mathbb{P}|}$ is the carrier and $\{tr_P : m_1 \mapsto \prod_{Q \in |\mathbb{P}|}(x_Q : m \mapsto \{m_2 \mid m_1 = m_2 \circ m\})\}_{P \in |\mathbb{P}|}$ is the coalgebra structure of the $F_{\mathbb{P}}$-coalgebra. $\mathcal{B}eh^{\mathbb{M}}_{\mathbb{P}}$ acts on morphisms $f : X \to Y$ in $\mathbb{M}$ as follows: $\mathcal{B}eh^{\mathbb{M}}_{\mathbb{P}}(f)_P : \alpha \mapsto (\alpha; f) : Hom_{\mathbb{M}}(P, X) \to Hom_{\mathbb{M}}(P, Y)$.

Proposition 2. *[20] For any two objects X and Y in $\mathbb{M}$, a $|\mathbb{P}|$-sorted relation R is a path-$\mathbb{P}$-bisimulation between X and Y iff it is an $F_{\mathbb{P}}$-bisimulation between $\mathcal{B}eh^{\mathbb{M}}_{\mathbb{P}}(X)$ and $\mathcal{B}eh^{\mathbb{M}}_{\mathbb{P}}(Y)$ containing the pair (i_X, i_Y), where $i_X : I \to X$ and $i_Y : I \to Y$ are paths, with an initial object I.*

Corollary 2. *For objects $\mathcal{T}$ and $\mathcal{T}'$ in* **CTTS**, **P***-bisimilarity, path-***P***-bisimilarity, strong path-***P***-bisimilarity, timed weak bisimulation coincide with an $F_{\mathbf{P}}$-bisimulation between $\mathcal{B}eh^{\mathbf{CTTS}}_{\mathbf{P}}(\mathcal{T})$ and $\mathcal{B}eh^{\mathbf{CTTS}}_{\mathbf{P}}(\mathcal{T}')$ containing the pair $(i_{\mathcal{T}}, i_{\mathcal{T}'})$, where $i_{\mathcal{T}} : I \to \mathcal{T}$ and $i_{\mathcal{T}'} : I \to \mathcal{T}'$ are paths, with an initial object I.*

8 Conclusion

In this paper, we have given a uniform characterization of what should be considered a weak bisimulation in a categorical framework of timed transition systems. In particular, we have shown how the open maps, path bisimilarity and coalgebra based approaches to an abstract characterization of bisimulation relate to each other and to timed weak bisimulation, in the setting of timed transition systems.

We conclude the paper by pointing out some possible research directions for the future. It is well-known that the traditional timed equivalences provide no abstraction mechanism on time, i.e. two timed processes which are close to each other but one of them simply performs events at a slightly slower speed are considered incomparable. Therefore, we intend to exploit the approach from [15] as part of our future work. Also, we plan to extend the obtained results to other classes of timed models (e.g. time Petri nets, networks of timed automata, etc.). In particular, relying on the paper [17], we contemplate to adapt the unfolding methods for time Petri nets from [10] and open maps based characterizations for timed event structures from [31] to transfer the general concept of bisimulation to the timed models mentioned above.

References

1. Aczel, P., Mendler, P.F.: A final coalgebra theorem. In: Dybjer, P., Pitts, A.M., Pitt, D.H., Poigné, A., Rydeheard, D.E. (eds.) CTCS 1989. LNCS, vol. 389, pp. 357–365. Springer, Heidelberg (1989)
2. Alur, R., Courcoubetis, C., Henzinger, T.A.: The observational power of clocks. In: Jonsson, B., Parrow, J. (eds.) CONCUR 1994. LNCS, vol. 836, pp. 162–177. Springer, Heidelberg (1994)
3. Andreeva, M.V., Virbitskaite, I.B.: Observational timed equivalences for timed stable event structures. Fundamenta Informaticae 72(4), 1–19 (2006)
4. Baeten, J.C.M., Middelburg, C.A.: Process algebra with timing. EATCS Monograph. Springer, Heidelberg (2002)
5. Borceux, F.: Handbook of Categorical Algebra. Encyclopedia of Mathematics and its Applications, vol. 2, 3, 51, 52. Cambridge University Press, Cambridge (1994)
6. Bihler, E., Vogler, W.: Timed Petri Nets: Efficiency of asynchronous systems. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 25–58. Springer, Heidelberg (2004)
7. van Breugel, F., Hermida, C., Makkai, M., Worrell, J.: An accessible approach to behavioural pseudometrics. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 1018–1030. Springer, Heidelberg (2005)
8. Cattani, G.L., Sassone, V.: Higher dimentional transition systems. In: Proc. LICS 1996, pp. 55–62 (1996)
9. Čerāns, K.: Decidability of bisimulation equivalences for parallel timer processes. In: Probst, D.K., von Bochmann, G. (eds.) CAV 1992. LNCS, vol. 663, pp. 302–315. Springer, Heidelberg (1993)
10. Chatain, T., Jard, C.: Time supervision of concurrent systems using symbolic unfoldings of time Petri nets. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 196–210. Springer, Heidelberg (2005)
11. Corradini, A., Grosse-Rhode, M., Heckel, R.: A coalgebraic presentation of structured transition systems. Theoretical Computer Science 260, 27–55 (2002)
12. Danos, V., Desharnais, J., Laviolette, F., Panangaden, P.: Bisimulation and cocongruence for probabilistic systems. Information and Computation 204(4), 503–523 (2006)
13. Fokkink, W., Pang, J., Wijs, A.: Is timed branching bisimilarity an equivalence indeed? In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 258–272. Springer, Heidelberg (2005)
14. Gribovskaya, N., Virbitskaite, I.B.: Timed Delay Bisimulation is an Equivalence Relation for Timed Transition Systems. Fundamenta Informaticae 93(1-3), 127–142 (2009)
15. Henzinger, T.A., Majumdar, R., Prabhu, V.S.: Quantifying similarities between timed systems. In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 226–241. Springer, Heidelberg (2005)
16. Hune, T., Nielsen, M.: Bisimulation and open maps for timed transition systems. Fundamenta Informaticae 38, 61–77 (1999)
17. Joyal, A., Nielsen, M., Winskel, G.: Bisimulation from open maps. Information and Computation 127(2), 164–185 (1996)
18. Kick, M., Power, J., Simpson, A.: Coalgebraic semantics for timed processes. Information and Computation 204(4), 588–609 (2006)

19. Klusener, A.S.: The silent step in time. In: Cleaveland, R. (ed.) CONCUR 1992. LNCS, vol. 630. Springer, Heidelberg (1992)
20. Lasota, S.: Coalgebra morphisms subsume open maps. Electronic Notes in Theoretical Computer Science 19 (1999)
21. Roggenbach, M., Majster-Cederbaum, M.: Towards a unified view of bisimulation: a comparative study. Theoretical Computer Science 238, 81–130 (2000)
22. Monteiro, L.: A Coalgebraic Characterization of Behaviours in the Linear Time – Branching Time Spectrum. In: Corradini, A., Montanari, U. (eds.) WADT 2008. LNCS, vol. 5486, pp. 251–265. Springer, Heidelberg (2008)
23. Nielsen, M., Cheng, A.: Observing behaviour categorically. In: Thiagarajan, P.S. (ed.) FSTTCS 1995. LNCS, vol. 1026, pp. 263–278. Springer, Heidelberg (1995)
24. Oshevskaya, E.: Open Maps Bisimulations for Higher Dimensional Automata Models. In: Kutylowski, M., Charatonik, W., Gębala, M. (eds.) FCT 2009. LNCS, vol. 5699, pp. 274–286. Springer, Heidelberg (2009)
25. Rothe, J., Masulovic, D.: Towards weak bisimulation for coalgebras. Electronic Notes in Theoretical Computer Science 68(1) (2002)
26. Rutten, J.: A note on coinduction and weak bisimilarity for while programs. Informatique Theorique et Applications 33(4/5), 393–400 (1999)
27. Rutten, J.: Universal coalgebra: a theory of systems. Theoretical Computer Science 249(1), 3–80 (2000)
28. Sokolova, A., de Vink, E.P., Woracek, H.: Weak Bisimulation for Action-Type Coalgebras. Electronic Notes in Theoretical Computer Science 122, 211–228 (2005)
29. Steffen, B., Weise, C.: Deciding testing equivalence for real-time processes with dense time. In: Borzyszkowski, A.M., Sokolowski, S. (eds.) MFCS 1993. LNCS, vol. 711, pp. 703–713. Springer, Heidelberg (1993)
30. de Vink, E.P., Rutten, J.J.M.M.: Bisimulation for probabilistic transition systems: a coalgebraic approach. Theoretical Computer Science 221, 271–293 (1999)
31. Virbitskaite, I.B., Gribovskaya, N.S.: Open maps and observational equivalences for timed partial order models. Fundamenta Informaticae 60(1-4), 383–399 (2004)
32. van der Zwaag, M.B.: The cones and foci proof technique for timed transition systems. Information Processing Letters 80(1), 33–40 (2001)

Community Structure in Large Complex Networks*

Liaoruo Wang and John Hopcroft

Cornell University, Ithaca NY 14853, USA
{lwang,jeh}@cs.cornell.edu

Abstract. In this paper, we establish the definition of community fundamentally different from what was commonly accepted in previous studies, where communities were typically assumed to be densely connected internally but sparsely connected to the rest of the network. A community should be considered as a densely connected subset in which the probability of an edge between two randomly-picked vertices is higher than average. Moreover, a community should also be well connected to the remaining network, that is, the number of edges connecting a community to the rest of the graph should be significant. In order to identify a well-defined community, we provide rigorous definitions of two relevant terms: "whiskers" and the "core". Whiskers correspond to subsets of vertices that are barely connected to the rest of the network, while the core exclusively contains the type of community we are interested in. We have proven that detecting whiskers, or equivalently, extracting the core, is an NP-complete problem for weighted graphs. Then, three heuristic algorithms are proposed for finding an approximate core and are evaluated for their performance on large networks, which reveals the common existence of the core structure in both random and real-world graphs. Further, well-defined communities can be extracted from the core using a number of techniques, and the experimental results not only justify our intuitive notion of community, but also demonstrate the existence of large-scale communities in various complex networks.

1 Introduction

Ever since people started to realize the importance of comprehending how interactions initiate and develop, the research on complex networks has attracted a great amount of attention. A substantial quantity of work has been devoted to the task of identifying and evaluating close-knit communities in large complex networks, most of which is based on the premise that it is a matter of common experience that communities exist in these networks [4]. In particular, as the Internet has become an indispensable part of our life, understanding community structure is not only crucial for studying real-world societies, but also helpful to improve the accuracy and reliability of predicting online behaviors, which may

* This research was partially supported by the U.S. Air Force Office of Scientific Research under Grant FA9550-09-1-0675.

© Springer-Verlag Berlin Heidelberg 2010

greatly benefit the quality and effectiveness of online services, such as search engines, recommendation systems, and so on.

A complex network is usually modeled as a graph in which vertices represent entities and edges represent interactions between pairs of entities. In previous studies, a community was often assumed to be a subset of vertices that are densely connected internally but sparsely connected to the rest of the network [2,3,4]. Accordingly, numerous measures have been proposed to capture this feature, out of which conductance has become one of the most widely adopted metrics for evaluating how community-like a subset of vertices is. Particularly, Leskovec et al. [4] conducted an extensive research on more than 100 large complex networks under the assumption that a community is more densely connected between its members than between its members and the remaining network. They carefully examined the relationship between conductance and community size, and discovered that the best community of the entire graph, i.e. the subset with the global minimum conductance, is usually a small set of vertices barely connected to the rest of the network by just a single edge.

However, it is our view that for real-world societies, communities are not only better connected than expected solely from chance, but are also well connected to the rest of the network. Actually, it is hard to imagine a small close-knit community, such as an academic department, with only one edge connecting it to the outside world. Empirically, a community displays a higher than average edge-vertex^2 ratio, which reflects the probability of an edge between two randomly-picked vertices, and it is also connected to the rest of the network via a significant number of edges, which is even possibly larger than the number of its internal edges, as depicted in Fig. 1.

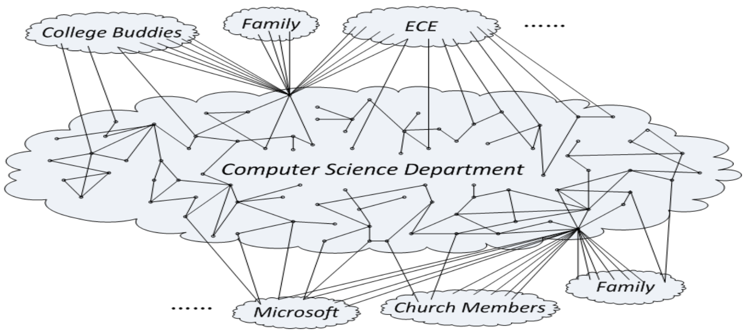


Fig. 1. An example friendship network. Vertices typically have a significant number of cut edges.

Given a subset of vertices, an edge with only one endpoint inside the subset can be thought as a cut edge. A densely connected subset with a small number of cut edges, called a whisker, is not the type of community we are interested in. Since many previously-used measures simultaneously maximize internal connections and minimize external connections, leaving whiskers in the graph will interfere with the algorithms intended to extract the type of community we are interested in. Whiskers are peripheral rather than central, thus, the type of community we would like to identify is embedded in a special structure in which no whiskers exist, called the core. To get rid of the interference generated by whiskers, a community detection algorithm can be designed consisting of two steps: 1) identifying the core in which no whiskers exist, and 2) identifying communities in the core. Apparently, any subset of the core is connected to the rest of the graph by a moderate number of edges, and conductance can still be taken as a measure of community goodness. In this way, the best community is not only more densely connected than expected from chance but also well connected to the remaining network, which exactly corresponds to our intuitive notion of community.

We prove that extracting the exact core from a weighted graph is NP-complete, and then conjecture the unweighted version of this problem to be also NP-complete. It is not difficult to see that, generally, the exact core cannot be obtained by removing whiskers one by one, but removing whiskers in a certain way can lead to an approximate core. We develop three heuristic algorithms, all of which are capable of finding an approximate core. Their performance can be verified by the experimental results obtained from random graphs and real-world graphs. In addition, we also discover that some algorithms are only suitable for a certain kind of networks but not for others. Further, the algorithms can be justified by the community profile of the core, in contrast to that of the entire graph shown in [4], which plots the smallest possible conductances with respect to fixed community sizes. In various complex networks, the best communities have a relatively large conductance, which means the communities are densely connected internally while preserving a significant number of cut edges. Moreover, they also have a relatively large size, which demonstrates the existence of large-scale well-defined communities.

The rest of this paper is organized as follows. In Section 2, we introduce some necessary background and present definitions of whiskers and the core. Then, in Section 3, we prove the NP-completeness of finding the exact core in weighted graphs and propose three heuristic algorithms for finding an approximate core. In Section 4, we apply the algorithms to random graphs and real-world graphs to evaluate their performance and compare the experimental results. Finally, we conclude in Section 5 with comments on the problems considered.

2 "Whiskers" and the "Core"

Let $G = (V, E)$ be an undirected graph with n vertices and m edges. A cut C is a collection of edges such that removing them from the graph G separates the vertex set V into two disjoint subsets S and S^c, where S^c denotes the complement of S and $C = \{(v, w) \in E \mid v \in S;\ w \in S^c\}$. Without loss of generality, we assume $|S| \leqslant |S^c|$ throughout this paper, where $|S|$ and $|S^c|$ denote the cardinality of sets S and S^c, respectively. Note that both S and S^c are not necessarily connected. Then, an edge $(v, w) \in C$ is called a *cut edge*, and intuitively, the cut size is the cardinality of the set C. Further, a cut is considered to be *suitable* if its removal divides the vertices into two disjoint subsets such that both have cardinality greater than or equal to the cut size.

Definition 1. *A cut of size k is a* ***suitable cut*** *if its removal from the graph partitions the vertex set into two disjoint subsets S and S^c, where $k \leqslant |S| \leqslant |S^c|$.*

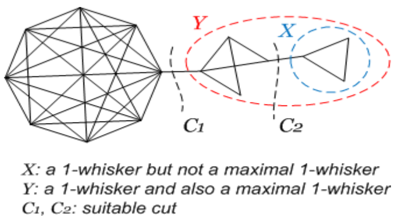
Leskovec et al. [4] defined 1-whiskers to be maximal subgraphs that can be detached from the rest of the graph by removing a single edge, and they also use the term "whiskers" informally to refer to subsets of vertices barely connected to the rest of the graph. Whiskers are generally quite small compared to the whole graph while possessing a wide range of sizes and shapes. Moreover, they usually correspond to low-conductance sets that are more densely connected inside than connected to the outside. Hence, whiskers and unions of disjoint whiskers are believed to exert a significant effect on the community structure of real-world networks, since they are extracted and interpreted as communities by the conductance measure, which, out of numerous density-based measures, has been extensively used for detecting communities and evaluating their quality [2,4,6].

However, as clarified in Section 1, this type of community neither corresponds to our intuitive notion of community nor widely exist in real-world societies, where it is a matter of common observation that communities are not only densely connected inside but also well connected to the outside. Therefore, it is of major interest to remove whiskers from the graph in order to provide insight into the community structure of the network core. For this purpose, we rigorously define whiskers and the corresponding core structure where barely-connected subsets have been removed.

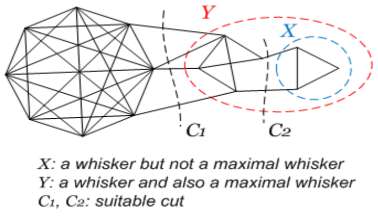
Definition 2. *Given an undirected graph $G = (V, E)$ with n vertices, a* ***k-whisker*** *is a connected subgraph $G_w(k) = (V_w(k), E_w(k))$ linked to the rest of the graph by k edges, where $k \leqslant |V_w(k)| \leqslant n/2$.*

Definition 3. *Given an undirected graph $G = (V, E)$ with n vertices, a* ***maximal k-whisker*** *is a maximal connected subgraph $G^*_w(k) = (V^*_w(k), E^*_w(k))$ linked to the rest of the graph by k edges, where $k \leqslant |V^*_w(k)| \leqslant n/2$.*

Small isolated components are frequently encountered in large complex networks, and they can simply be viewed as (maximal) 0-whiskers. Definition 2 and 3 are a


Y
X
C1
C2
X: a 1-whisker but not a maximal 1-whisker
Y: a 1-whisker and also a maximal 1-whisker
C1, C2: suitable cut

(a) k-whisker vs. maximal k-whisker


Y
X
C1
C2
X: a whisker but not a maximal whisker
Y: a whisker and also a maximal whisker
C1, C2: suitable cut

(b) whisker vs. maximal whisker

Fig. 2. Schematic illustrations of Definition 1 through Definition 5

direct extension of the definition of 1-whiskers given in [4]. Then, in a similar way, the definitions of whiskers and maximal whiskers can be formulated independent of the value of k referring to weakly-connected subsets attached to the remaining graph via a small number of edges.

Definition 4. *Given an undirected graph $G = (V, E)$ with n vertices, a* ***whisker*** *is a connected subgraph $G_w = (V_w, E_w)$ linked to the rest of the graph by a suitable cut, where $|V_w| \leqslant n/2$.*

Definition 5. *Given an undirected graph $G = (V, E)$ with n vertices, a* ***maximal whisker*** *is a maximal connected subgraph $G_w^* = (V_w^*, E_w^*)$ linked to the rest of the graph by a suitable cut, where $|V_w^*| \leqslant n/2$.*

See Fig. 2 for a detailed illustration of Definition 1 through Definition 5. A maximal whisker is obviously a whisker, but a whisker is not necessarily a maximal whisker, since it can be contained in a larger whisker. Besides, a 0-whisker is also a maximal whisker by Definition 5.

As discussed above, maximal whiskers, although argued by some to be community-like, are not what we are interested in here. Therefore, we define the core as the remaining structure after removing the union of all maximal whiskers from the graph. Meaningful communities can be further extracted from the core using a variety of algorithms, which, unlike whiskers, are not only better connected than expected from chance but also well connected to the rest of the graph.

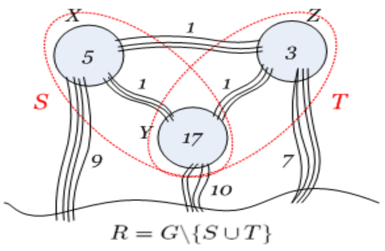
Definition 6. *The* ***core*** *is a connected subgraph that is the complement of the union of all maximal whiskers.*

Clearly, there does not exist any suitable cut in the core subgraph. Before we move on to Section 3 to design and implement algorithms for finding the core structure and its underlying communities, we first examine some properties of whiskers. If all maximal whiskers are disjoint in the graph, it is straightforward that we can remove these disjoint whiskers one by one until we obtain the core. However, whiskers may overlap with each other, and unfortunately, their union is often no longer a whisker. In fact, a number of counterexamples can be constructed to justify this statement, and we conclude the following lemma:

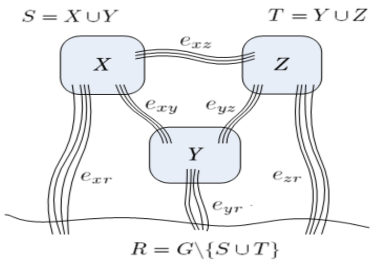
Lemma 1. *Let G be an undirected graph with two overlapping maximal whiskers S and T. The subgraph $S \cup T$ is not necessarily a whisker.*

Proof. As shown in Fig. 3(a), for instance, $S = X \cup Y$ is a maximal whisker with 22 vertices and 21 outgoing edges. Similarly, $T = Y \cup Z$ is also a maximal whisker with 20 vertices and 19 outgoing edges. However, there are a total of 25 vertices in the set $X \cup Y \cup Z$ and 26 outgoing edges that connect this union to the rest of the graph, thus $S \cup T$ is not a whisker. □

In general, there are two reasons why a subset of vertices is not a whisker: 1) it contains more than half of the vertices, and 2) the number of edges connecting it to the rest of the graph is strictly greater than its cardinality. Thus, the union



(a) Overlapping maximal whiskers.



(b) Overlapping whiskers.

Fig. 3. Each circled integer denotes the number of vertices in the corresponding set

of two disjoint whiskers is still a whisker if and only if it is no larger than its complement. In addition, the union of two overlapping maximal whiskers is not a maximal whisker, since a maximal whisker cannot have any other maximal whisker as its subset. Based on Lemma 1, there is another observation we can make about whiskers:

Lemma 2. *Let G be an undirected graph with n vertices and two overlapping whiskers S and T, where the number of vertices in the subgraph $S \cup T$ is no more than $n/2$. If $S \cup T$ is not a whisker, then $S \cap T$ must be a whisker.*

Proof. Assuming that the subgraph $S \cup T$ is not a whisker, write $S = X \cup Y$ and $T = Y \cup Z$ where $S \cap T = Y$, as shown in Fig. 3(b). Then, it follows that

$$e_{xr} + e_{xz} + e_{yr} + e_{yz} \leqslant v_x + v_y \tag{2a}$$
$$e_{yr} + e_{xy} + e_{zr} + e_{xz} \leqslant v_y + v_z \tag{2b}$$
$$e_{xr} + e_{yr} + e_{zr} > v_x + v_y + v_z \tag{2c}$$

where v_x, v_y, and v_z denote the number of vertices in the sets X, Y, and Z, respectively. Adding Equation (2a) and (2b), we have that

$$\begin{aligned} e_{xr} + 2e_{yr} + e_{zr} + e_{xy} + e_{yz} + 2e_{xz} &\leqslant v_x + 2v_y + v_z \\ &< e_{xr} + e_{yr} + e_{zr} + v_y. \end{aligned}$$

Thus,

$$e_{yr} + e_{xy} + e_{yz} + 2e_{xz} < v_y. \tag{2d}$$

Since e_{xz} is non-negative as the number of edges between the sets X and Z, by Equation (2d),

$$e_{yr} + e_{xy} + e_{yz} < v_y,$$

and the subgraph $Y = S \cap T$ is clearly a whisker. □

3 Methodology

In this section, we discuss the approach for efficiently identifying the core of a given graph. Armed with the definitions provided in Section 2, we prove in Section 3.1 that detecting whiskers in a weighted undirected graph is NP-complete and thus computationally intractable unless P=NP. This indicates that there is no feasible algorithm for finding the exact core, which is equivalent to finding the union of all maximal whiskers. Then, in Section 3.2, we propose three heuristic algorithms for finding an approximate core, whose performance will be experimentally justified in Section 4.

3.1 NP-Completeness

Define NAE-3-SAT as the problem of determining whether there exists a truth assignment for a 3-CNF Boolean formula such that each clause has at least one true literal and at least one false literal (i.e. literals in each clause are not all equal). Then, we have the following well-known theorem:

Theorem 1. NAE-3-SAT *is* NP*-complete* [5].

Now, define WHISKER as the problem of determining whether there exists a whisker in a given weighted undirected graph. We will formally prove that WHISKER is also an NP-complete problem by constructing a polynomial-time reduction from NAE-3-SAT.

Theorem 2. WHISKER *is* NP*-complete.*

Proof (Sketch). See [7] for a detailed proof. Given an instance of the WHISKER problem, we can guess a solution and verify in linear time whether it is indeed a whisker, thus WHISKER $\in$ NP.

Given a 3-CNF Boolean formula with c clauses and n variables, a weighted graph G^* can be constructed in polynomial time. The edge weights of G^* are taken as functions of ε and δ, where ε and δ are small positive numbers. Then, with proper values of ε and δ for the given c and n, the true literals of a not-all-equal assignment for the formula correspond to the vertices of a whisker in G^*, and the vertices of a whisker in G^* also correspond to the true literals of a not-all-equal assignment for the formula. Therefore, we have established a one-to-one correspondence between not-all-equal truth assignments and whiskers, that is, a weighted graph can be constructed for a given 3-CNF Boolean formula such that whiskers can be found in the graph if and only if the formula is not-all-equal satisfiable. Clearly, NAE-3-SAT reduces to WHISKER in polynomial time, thus, WHISKER is NP-complete. $\square$

We then conjecture that detecting whiskers in an unweighted graph is also an NP-complete problem.

3.2 Heuristic Algorithms

An intuitive approach to identifying the core is simply to remove maximal whiskers one by one until no more whiskers exist. However, the following claim characterizes the non-exactness and non-uniqueness of this method, which indicate the generic difficulties associated with any algorithm using this approach to find the core structure.

Claim. Removing maximal whiskers one by one leads to different subgraphs approximate to the exact core, depending on the order in which whiskers are removed.

Proof. Here, we can still take Fig. 3(a) as an example. Assume that sets S and T are both maximal whiskers and that they do not intersect with other maximal whiskers. If the set S is first removed, we will be left with the set Z of 3 vertices and 7 outgoing edges, which is apparently not a (maximal) whisker. However, if the set T is first removed instead, we will be left with the set X of 5 vertices and 9 outgoing edges, which is not a (maximal) whisker either. In this case, different sets of vertices remain as part of the ultimate subgraph, neither of which belongs to the exact core. Therefore, the approximate core subgraph depends rather crucially on the order in which we remove these maximal whiskers from the graph, which means that it is not necessarily unique. $\square$

The NP-completeness of identifying the exact core in weighted graphs has been proved in Section 3.1. We conjecture that identifying the exact core in unweighted graphs is also an NP-complete problem. Now, we present three heuristic algorithms for finding an approximate core, whose performance on random and real-world graphs will be experimentally demonstrated in Section 4.

Algorithm 1 (brute-force search). For each ordered pair of vertices, find its minimum cut and remove the smaller component if the cut is suitable.

Algorithm 2. Extract the giant component and then the giant biconnected component. Replace all degree-two vertices by a single edge and then test the existence of suitable cuts.

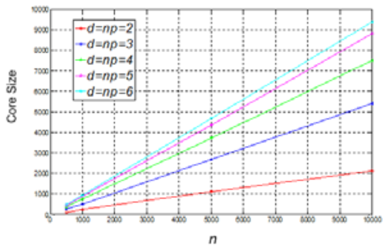
Algorithm 3 (flow-based algorithm). For a given threshold value λ, find the largest subgraph with the maximum edge-vertex ratio exceeding λ. Then, test the existence of suitable cuts. Refer to [7] for more details.

There is no particular order in which whiskers are removed by Algorithm 1. According to the above claim, larger maximal whiskers could be destroyed and the resulting graph is not necessarily unique, depending rather crucially on the order in which Algorithm 1 removes whiskers. Since a series of degree-two vertices could result in a whisker, Algorithm 2 contracts all degree-two vertices after obtaining the giant biconnected component. Although Algorithm 2 offers a better run-time performance compared to Algorithm 1, it actually encounters the same difficulties as Algorithm 1 does. The three algorithms are all capable of finding an approximate core, but we will focus on Algorithm 2 and Algorithm 3 since they require shorter running time. Empirically, Algorithm 2 works better for sparse networks, while Algorithm 3 works better for dense ones.

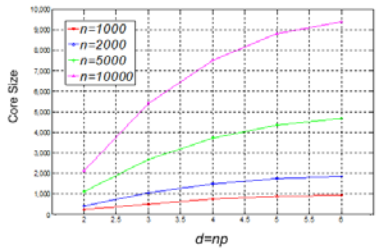
4 Experimental Results

4.1 Random Graphs

A random graph $G(n, p)$ can be obtained by starting with a set of n vertices and adding (undirected) edges between them independently with probability



(a) size of the core as a function of n for fixed d



(b) size of the core as a function of d for fixed n

Fig. 4. Random Graphs

$p \in (0, 1)$. Although a random graph does not display any community structure, we can still identify its core using the above algorithms. When p is relatively small, $G(n, p)$ is sparse with low edge-vertex ratio, where Algorithm 3 fails to find an approximate core. In this case, Algorithm 2 can positively identify an approximate core. When p is close to 1, both algorithms are successful in finding an approximate core. As illustrated in Fig. 4, the size of the core of $G(n, p)$ grows linearly with $d = np$ for fixed n and logarithmically with n for fixed d. In addition, we observe the existence of phase transition at $p = 1/n$, above which the core emerges with high probability and below which it emerges with extremely low probability.

We conjecture that every $G(n, p)$ with $p > 1/n$ displays the core structure with high probability. For any fixed (large) n, $p = 1/n$ is the threshold for phase transition at which the core structure emerges. The probability and the average size of the core both increase as p grows. For any fixed p, the average size of the core increases as n grows, but the probability of the core remains the same.

4.2 Real-World Graphs

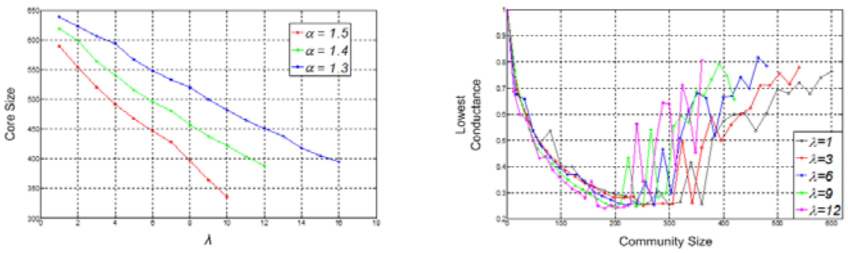
Textual Graph. A textual graph consists of vertices representing words and edges representing semantic correlations, which contains information about research topics and areas of interest. We crawl more than 10,000 scientific papers of the KDD conference from 1992 to 2003 and collect the words of each abstract. A series of pre-processing steps are carried out to simplify the data, which include word stemming, stop-word filtering, and occurrence rate thresholding. Word stemming reduces inflected or derived words to their base form and combines multiple entries of the same word in different tenses. Stop-word filtering removes extremely common but meaningless words, such as and, can, the, will, etc. Occurrence rate thresholding removes extremely rare words occurring in only a small number of abstracts, which exert a trivial effect on the overall community structure.

Pointwise mutual information or log-likelihood ratio can be applied to determine whether there is an edge between each pair of vertices of the textual graph. In this section, we will only discuss the first approach. Pointwise mutual information quantifies the semantic correlation between two words, and we may choose a critical value α above which a strong correlation can be expected. In other words, if the mutual information of two words exceeds α, then an edge exists between them, which indicates a high probability for the two words to occur together. Otherwise, no edge exists between them, which indicates a low probability for the two words to occur together. For a pair of words (i, j) and the threshold value α, there exists an edge between vertex i and vertex j if

$$\log \frac{P(i,j)}{P(i)P(j)} > \alpha,$$

where $P(i)$ and $P(j)$ are the occurrence rate of i and j, respectively, and $P(i, j)$ is the probability of i and j occurring in the same abstract.

For example, the textual graph has 685 vertices and 6,432 edges when $\alpha = 1.4$. Both Algorithm 2 and Algorithm 3 are successful in identifying an approximate core, in which no whiskers exist. In particular, the core returned by Algorithm 2 is almost identical to that returned by Algorithm 3 when λ is relatively small. Higher values of λ will result in a smaller core, and intuitively, higher values of α will result in a graph with less edges and thus a smaller core, as verified in Fig. 5(a).



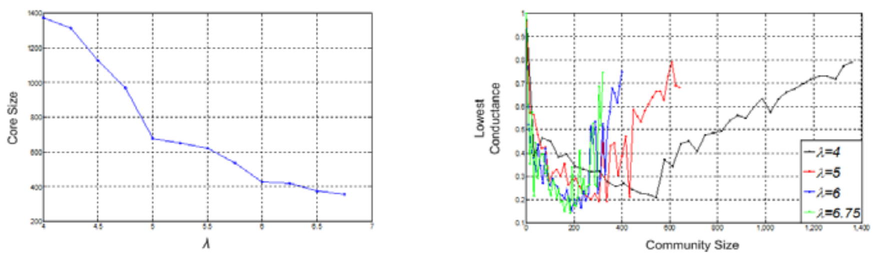
(a) size of the core as a function of λ (b) community profile of the core ($\alpha = 1.4$)

Fig. 5. Textual Graph

After the approximate core has been extracted from the graph, a simulated annealing algorithm can be performed on the core for finding a subset of a given size with the lowest conductance. As shown in Fig. 5(b), the best community of the textual graph possesses a quite large conductance around 0.3, which means the best community has only as many internal edges as cut edges. This exactly corresponds to our intuitive notion that a community should have a significant number of edges connecting it to the rest of the graph. Clearly, the community profile of the core is rather different from what was obtained in [4]. Recall that the best community of most networks examined in [4] displayed an extremely small conductance, typically at the order of 10^{-2}, which means the best community has almost 50 times as many internal edges as cut edges. Moreover, the best community of the textual graph is of size roughly 350 for $\alpha = 1.4$ and $\lambda = 1$, containing more than half of the vertices, which demonstrates the existence of large-scale well-defined communities.

Co-authorship Graph. A co-authorship graph reflects the common interests among researchers working in diverse fields, which contains information about authors' reputation and levels of activity. We collect more than 10,000 scientific papers of the KDD conference from 1992 to 2003 and refine the authors' information [1]. Different from the textual graph discussed in Section 4.2, the co-authorship graph is deterministic with 7,943 vertices and 20,488 edges, where each vertex represents an author and each edge represents a co-authorship. Here, Algorithm 2 is not successful in finding an approximate core by pulling out the giant biconnected component and contracting degree-two vertices. In contrast, Algorithm 3 is able to identify an approximate core, and its size decreases as the threshold value λ increases, as shown in Fig. 6(a).

As depicted in Fig. 6(b), the community profile of the core of the co-authorship graph is rather different from what was obtained in [4]. Recall that the best community of most networks examined in [4] displayed an extremely small conductance, typically at the order of 10^{-2}, which means the best community has almost 50 times as many internal edges as cut edges. Here, the best community of the co-authorship graph possesses quite a large conductance around 0.2, which means the best community has only twice as many internal edges as cut edges. This, again, corresponds to our intuitive notion that a community should have a moderate number of edges connecting it to the rest of the graph. Moreover, the best community of the co-authorship graph is of size roughly 500 for $\lambda = 4$, containing more than a third of the vertices, which again demonstrates the existence of large-scale well-defined communities.



(a) size of the core as a function of λ (b) community profile of the core

Fig. 6. Co-authorship Graph

5 Conclusion

We investigated large real-world complex networks and proposed an innovative definition of community as opposed to what was generally assumed in previous studies, where communities were thought to be better connected internally than connected with the rest of the network. In fact, a community is more densely connected internally than expected solely from chance, but it is also connected to the rest of the network by a significant number of edges. Further, we defined two auxiliary terms: whiskers and the core. Whiskers were often interpreted as communities, but they are not the type of community we are interested in here. In contrast, the core exclusively contains the type of community we would like to identify.

Armed with these definitions, we designed a community detection algorithm consisting of two steps: 1) identifying the core in which no whiskers exist, and 2) identifying communities within the core. However, extracting the exact core is rigorously proved to be NP-complete for weighted graphs, and we also conjecture the NP-completeness of this problem for the unweighted case. The three heuristic algorithms demonstrate their capability of finding an approximate core, and a simulated annealing algorithm is performed on the approximate core to find its best community, i.e. the subset with the lowest conductance, for a given community size. As expected, the network community profile of the core justifies our definition of community and shows the existence of large-scale well-defined communities in various real-world complex networks.

References

1. arXiv data (1992-2003), http://www.cs.cornell.edu/projects/kddcup
2. Gaertler, M.: Clustering. In: Brandes, U., Erlebach, T. (eds.) Network Analysis. LNCS, vol. 3418, pp. 178–215. Springer, Heidelberg (2005)
3. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proc. Natl. Acad. Sci. USA, 7821–7826 (2002)
4. Leskovec, J., Lang, K. J., Dasgupta, A., and Mahoney, M. W. Community structure in large networks: natural cluster size and the absence of large well-defined clusters. Tech. Rep. arXiv:0810.1355 (2008)
5. Schaefer, T.J.: The complexity of satisfiability problems. In: Proc. 10th Ann. ACM Symp. on Theory of Computing, pp. 216–226. ACM, New York (1978)
6. Schaeffer, S.E.: Graph clustering. Computer Science Review 1(1), 27–64 (2007)
7. Wang, L., Hopcroft, J.E.: Community structure in large complex networks. Tech. Rep., Cornell University (2010), http://hdl.handle.net/1813/14710

Generating Internally Triconnected Rooted Plane Graphs

Bingbing Zhuang and Hiroshi Nagamochi

Graduate School of Informatics, Kyoto University
{zbb,nag}@amp.i.kyoto-u.ac.jp

Abstract. A biconnected plane graph G is called *internally triconnected* if any cut-pair consists of outer vertices and its removal results in only components each of which contains at least one outer vertex. In a rooted plane graph, an edge is designated as an outer edge with a specified direction. For given positive integers $n \geq 1$ and $g \geq 3$, let $\mathcal{G}_3(n, g)$ (resp., $\mathcal{G}_{\mathtt{int}}(n, g)$) denote the class of all triconnected (resp., internally triconnected) rooted plane graphs with exactly n vertices such that the size of each inner face is at most g. In this paper, we present an $O(1)$-time delay algorithm that enumerates all rooted plane graphs in $\mathcal{G}_{\mathtt{int}}(n, g) - \mathcal{G}_3(n, g)$ in $O(n)$ space.

1 Introduction

The problem of enumerating (i.e., listing) all graphs in particular classes of graphs is one of the most fundamental and important issues in graph theory. Cataloguing graphs, i.e., making the complete of graphs in a particular class can be used in a various way: search for a possible counterexample to a mathematical conjecture; choosing the best graph among all candidate graphs; and experiment for measuring the average performance of a graph algorithm over all possible input graphs.

The common idea behind most of the recent efficient enumeration algorithms is to define a parent-child relationship among all graphs in a given class in order to induce a rooted tree that connects all graphs in the class, called the *family tree* $\mathcal{F}$, where each node in $\mathcal{F}$ corresponds to a graph in the class. Then all graphs in the class will be generated one by one according to the depth-first traversal of the family tree $\mathcal{F}$. Time delay of an enumeration algorithm is a time bound between two consecutive outputs. Enumerating graphs with a polynomial time delay would be rather easy since we can examine the whole structure of the current graph anytime. However, algorithms with a constant time delay in the worst case is a hard target to achieve; not only the difference between two consecutive outputs is required to be $O(1)$, but also any operation for examining symmetry and identifying the edges/vertices to be modified to get the next output needs to be executable in $O(1)$ time.

Enumeration for a particular class of graphs also has practical applications in various fields such as virtual exploration of chemical universe, and reconstruction

© Springer-Verlag Berlin Heidelberg 2010

of molecular structures from their signatures. It is known that 94.3% of chemical compounds in NCI chemical database have planar structures [2]. Hence planar graphs is an important class to be investigated. Planar graphs also plays a key role in the field of graph drawing. Tutte [6] proved that *triconnected* plane graphs is the class of plane graphs that admit convex drawings in the plane for any prescribed polygonal boundary. Thomassen [5] proved that *internally triconnected* plane graphs is the class of plane graphs that admit a convex drawings, where a suitable convex polygon for the boundary is allowed to be chosen for each plane graph.

Yamanaka and Nakano [7] gave an algorithm for generating all connected rooted plane graphs with at most m edges in $O(1)$ time per graph on average using $O(m)$ space. Li and Nakano [3] and Nakano [4] presented $O(1)$-time delay algorithms that enumerate all biconnected and triconnected rooted triangulated plane graphs, respectively. In our companion papers [10,11], we gave $O(1)$-time delay enumeration algorithms for the class $\mathcal{G}_2(n,g)$ of biconnected rooted plane graphs and the class $\mathcal{G}_3(n,g)$ of triconnected rooted plane graphs such that the number of vertices is exactly $n \geq 1$ and the size of each inner face is at most $g \geq 3$.

In this paper, we consider the class $\mathcal{G}_{\texttt{int}}(n,g)$ of all internally triconnected rooted plane graphs with exactly n vertices such that the size of each inner face is at most g, where $\mathcal{G}_3(n,g) \subseteq \mathcal{G}_{\texttt{int}}(n,g) \subseteq \mathcal{G}_2(n,g)$. The structure of internally triconnected plane graphs is more complicated than those of biconnected/triconnected plane graphs, since they are combinations of triconnected components under a planarity constraint. We present an $O(1)$-time delay algorithm that enumerates all plane graphs in $\mathcal{G}_{\texttt{int}}(n,g) - \mathcal{G}_3(n,g)$ using $O(n)$ space. Our algorithm also yields an $O(n^3)$-time delay algorithm for generating all internally triconnected *unrooted* plane graphs with exactly n vertices such that the size of each inner face is at most g.

The rest of the paper is organized as follows. After introducing basic notations in Section 2, Section 3 treats the class $\mathcal{G}_{\texttt{int}}$ of internally triconnected rooted plane graphs, where we define the parent of each graph in $\mathcal{G}_{\texttt{int}}$, and characterize the children of a graph in $\mathcal{G}_{\texttt{int}}$. Section 4 describes an algorithm for enumerating all internally triconnected rooted plane graphs in $\mathcal{G}_{\texttt{int}}(n,g)$, and analyzes the time and space complexities of the algorithm. The proofs omitted due to the space limitation can be found in a full version of the paper [12].

2 Preliminaries

A graph is denoted by a pair $G = (V, E)$ of a vertex set V and an edge set E. The set of vertices and the set of edges of a given graph G are denoted by $V(G)$ and $E(G)$, respectively.

For a subset $E' \subseteq E(G)$, $G - E'$ denotes the graph obtained from a graph G by removing the edges in E'. Let X be a subset of $V(G)$. Let $G[X]$ denote the subgraph induced from G by X, and $G - X$ denote the graph obtained from G by removing the vertices in X together with the edges incident with a vertex in X. Let $deg(v; G)$ denote the degree of a vertex v in a graph G. For a vertex $v \in V$ in a graph G, let $\Gamma(v; G)$ denote the set of *neighbours* of v (i.e., vertices adjacent to v). For a subset $X \subseteq V$, let $\Gamma(X; G) = \cup_{v \in X} \Gamma(v; G) - X$. The *vertex-connectivity* of G is denoted by $\kappa(G)$.

A *fan* is the graph obtained from a path P with at least one vertex by adding a new vertex v together with an edge incident to each vertex in the path, where the vertex v is called the *center* of a fan. A fan with n vertices is denoted by F_n. Note that F_n $(n \geq 2)$ are biconnected.

A graph is called *planar* if its vertices and edges can be drawn as points and curves on the plane so that no two curves intersect except for their endpoints. A planar graph with such a fixed embedding is called a *plane* graph, where a face is designated as the *outer face* and all other faces are called *inner faces*. Let $F(G)$ denote the set of faces in a plane graph G. For a face f in a plane graph G, let $V(f)$ and $E(f)$ denote the sets of vertices and edges on the facial cycle of f, and define the size $|f|$ of face f to be $|V(f)|$. For a vertex v and an edge e, let $F(v)$ (or $F(v; G)$) denote the set of inner faces f with $v \in V(f)$ and $F(e)$ (or $F(e; G)$) denote the set of inner faces f with $e \in E(f)$. An inner face $f \in F(v)$ is called a *v-face*.

A *rooted* plane graph is a plane graph which has a designated outer edge (u, r) with orientation from u to r, where r is called the *root*. Two rooted plane graphs G_1 and G_2 are *equivalent* if their vertex sets admit a bijection by which the designated directed edge and the incidence-relation between edges and vertices/faces in G_1 correspond to those in G_2.

3 Internally Triconnected Rooted Plane Graphs

A biconnected plane graph G is called *internally triconnected* if any cut-pair consists of outer vertices and its removal results in only components each of which contains at least one outer vertex.

For an integer $n \geq 1$, let $\mathcal{G}_{\mathtt{int}}(n)$ denote the set of all internally triconnected rooted plane graphs with exactly n vertices.

Let G be an internally triconnected plane graph with root edge $(\hat{r}, r)$. Let $\beta[u, v]$ denote the clockwise sequence between two outer vertex u and v. We denote the sequence of outer vertices in $\beta[r, \hat{r}]$ by $(v_1 = r, v_2, \ldots, v_B = \hat{r})$. The *fan factor* is defined to be the maximal subsequence $s_t = v_B, s_{t-1} = v_{B-1}, \ldots, s_1 = v_{B-t+1}$ of $\beta[r, \hat{r}]$ such that $\Gamma(s_t; G) = \{r, s_{t-1}\}$ and $\Gamma(s_i; G) = \{r, s_{i-1}, s_{i+1}\}$, $1 \leq i \leq t-1$ where we let $s_0 = v_{B-t}$. Let $\psi(G)$ denote the fan factor of G. See Fig. 1(b), where $t = 3$. If $\psi(G) \neq \emptyset$, then $\psi(G)$ induces F_t, $t = |\psi(G)|$ from G. Note that $|\psi(G)| \leq n-2$ if G with n vertices is not F_n. If G is triconnected, then $\psi(G) = \emptyset$. We say that $\psi(G)$ is *augmented by 1* in a graph G with $t = |\psi(G)| \geq 0$ when we introduce a new vertex s_{t+1} together with two new edges (s_{t+1}, r) and (s_{t+1}, s_t). Conversely, we say that $\psi(G)$ is *reduced by 1* in a graph G with $t = |\psi(G)| \geq 1$ when we remove the vertex s_t together with two incident edges (s_t, r) and (s_t, s_{t-1}).

For two outer vertices $u = v_q$ and $v = v_\ell$, $1 \le q < \ell \le B+1$ in G, the path $\beta[u, v]$ is called *pendant* if $\{u, v\}$ is a cut-pair or $\{u, v\} = \{v_1, v_B\}$. A pendant path $\beta[u, v]$ is called *reducible* if it has no proper path $\beta[u', v']$ which is pendant. For a pendant path $\beta[u, v]$, let X denote the set of vertices of the component containing $\beta[u, v] - \{u, v\}$ in $G - \{u, v\}$, and $G[u, v]$ denote the graph $G[X \cup \{u, v\}]$ together with edge (u, v), where we add a new edge (u, v) if u and v are not adjacent in G. We easily observe that path $\beta[u, v]$ is reducible if and only if $G[u, v]$ is triconnected and $\{u, v\}$ is a cut-pair or $\{u, v\} = \{v_1, v_B\}$.

A reducible path $\beta[v_q, v_\ell]$, $1 \le q < \ell \le B$ and its subgraph $G[v_q, v_\ell]$ are called the *first reducible path* and the *first reducible subgraph* if $\ell \le B - t$ and there is no reducible path $\beta[v_i, v_j]$ with $1 \le i < q$.

Let $\beta[v_q, v_\ell]$, $1 \le q < \ell \le B+1$ be the first reducible path. We denote the outer vertices in $G[v_q, v_\ell]$ by $u_1 = v_q, u_2, \ldots, u_{B'} = v_\ell$, $(B' = j - i + 1)$, and call $u_1 = v_q$ and $u_{B'} = v_\ell$ the *stating vertex* and *ending vertex* of $G[v_q, v_\ell]$, respectively. We treat graph $G[v_q, v_\ell]$ as a plane graph such that $u_1 = v_q$ and $(u_{B'}, u_1)$ are specified as the root r' and the root edge, respectively. An outer edge e is called *removable* if $G[v_q, v_\ell] - e$ remains triconnected. The *first removable element* of $G[v_q, v_\ell]$ is defined to be a removable edge $e_p = (u_p, u_{p+1})$ or a vertex u_p of degree 3 that appears for the first time along $\beta[v_q, v_\ell]$ from $r' = v_q$ to v_ℓ. We call the vertex u_p the *critical vertex*, and call the path $\beta[r', u_p]$ *active*. For each irremovable outer edge $e_i = (u_i, u_{i+1})$, $G[v_q, v_\ell]$ has a vertex cut $\{u_{i+1}, w, u_k\}$ such that w is an inner vertex and u_k is an outer vertex, and we let $\tau_{\texttt{last}}(e_i) = u_{k_i}$ denote such an outer vertex u_k $(\neq u_i, u_{i+1})$ with the largest index k. Note that $G[v_q, v_\ell]$ has two faces f and f' with $\{u_i, u_{i+1}, w\} \subseteq V(f)$ and $\{w, u_k\} \subseteq V(f')$.

3.1 Parents of Internally Triconnected Rooted Plane Graphs

As mentioned above, each reducible subgraph is a triconnected plane graph. For triconnected rooted plane graphs, we already have obtained a definition of parents to form a family tree which admits an $O(1)$-time delay enumeration algorithm [11]. In this subsection, we define the parent for an internally triconnected rooted plane graph by replacing the first reducible subgraph by its parent with respect to triconnected rooted plane graphs [11].

An internally triconnected plane graph with $n \le 3$ vertices is unique. In what follows, we assume that $n \ge 4$ and $g \ge 3$, and treat fan F_n as a plane graph rooted at an edge (v, v_c) incident to the center v_c of F_n. Let G be an internally triconnected rooted plane graph with $n \ge 4$ such that $G \neq F_n$. We define the *parent* $\mathcal{P}(G)$ of G to be the following graph with n vertices.

In $G[v_q, v_\ell]$, let z' denote the second leftmost neighbour of the starting vertex $r' = v_q$, and call the r'-face containing $(u_{B'}, r')$ and (r', z') the *leftmost* r'-face, denoted by f'_{L}. Note that $|f'_{\mathrm{L}}| = 3$ if and only if $(u_{B'}, z') \in E(G)$.

P1 $G[v_q, v_\ell]$ is a triangle: The parent $\mathcal{P}(G)$ is defined to be the graph obtained from G by removing vertex v_{q+1} and augmenting the fan factor of G by 1 (see Fig. 1(a)-(b)).

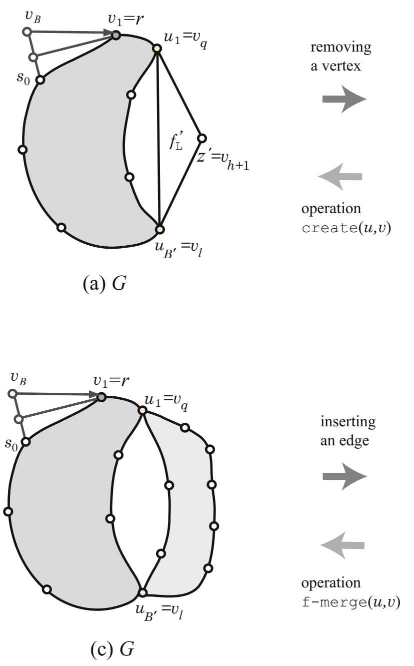


Fig. 1. (a), (b) Illustration for the parent in P1 and operation `create`; (c), (d) illustration for the parent in P2 and operation `f-merge`

P2 G has no edge joining the starting vertex $u_1 = v_q$ and the ending vertex $u_{B'} = v_\ell$: The parent $\mathcal{P}(G)$ is defined to be the graph obtained from G by inserting a new inner edge (v_q, v_ℓ) (see Fig. 1(c)-(d) and Fig. 5(a)-(b)).

P3 $(v_q, v_\ell) \in E(G)$, and the inner face f'_{L} of $G[v_q, v_\ell]$ containing edge (v_q, v_ℓ) is of length at least 4: Let z' be the second leftmost neighbour of $u_1 = v_q$ in $G[v_q, v_\ell]$ (see Fig. 2(a)). The parent $\mathcal{P}(G)$ is defined to be the graph obtained from G by splitting the face f'_{L} with a new inner edge (z', v_ℓ) (see Fig. 2(b)).

P4 $(v_q, v_\ell) \in E(G)$, $|f'_{\text{L}}| = 3$, $|V(G[v_q, v_\ell])| \geq 4$, and the first removable element in $\beta[v_q, v_\ell]$ of the triconnected graph $G[v_q, v_\ell]$ is an edge e: The parent $\mathcal{P}(G)$ is defined to be $G - e$ (see Fig. 2(c) and (d)).

P5 $(v_q, v_\ell) \in E(G)$, $|f'_{\text{L}}| = 3$, $|V(G[v_q, v_\ell])| \geq 4$, $B' = 3$, and the degree of $u_2 = v_{q+1}$ is 3, where the three neighbours of v_{q+1} are denoted by v_q, w and $v_{q+2} = v_\ell$ (see Fig. 3(a)-(d)): The parent $\mathcal{P}(G)$ is defined to be the graph obtained from G by removing vertex v_{q+1}, augmenting the fan factor of G by 1 and making w incident to v_q and v_ℓ by introducing new edges (w, v_q) and (w, v_ℓ) if necessary (see Fig. 3(e)).

P6 $(v_q, v_\ell) \in E(G)$, $|f'_{\text{L}}| = 3$, $|V(G[v_q, v_\ell])| \geq 4$, $B' \geq 4$, and the first removable element in $\beta[v_q, v_\ell]$ of the triconnected graph $G[v_q, v_\ell]$ is a vertex u_{i+1}, $1 \leq i \leq B' - 2$ (see Fig. 4(a)-(d)): Construct G/u_{i+1} from G. The parent $\mathcal{P}(G)$ is defined to be the graph obtained by augmenting fan factor $\psi(G/u_{i+1})$ by 1 (see Fig. 4(e) and Fig. 5(c)-(d)).

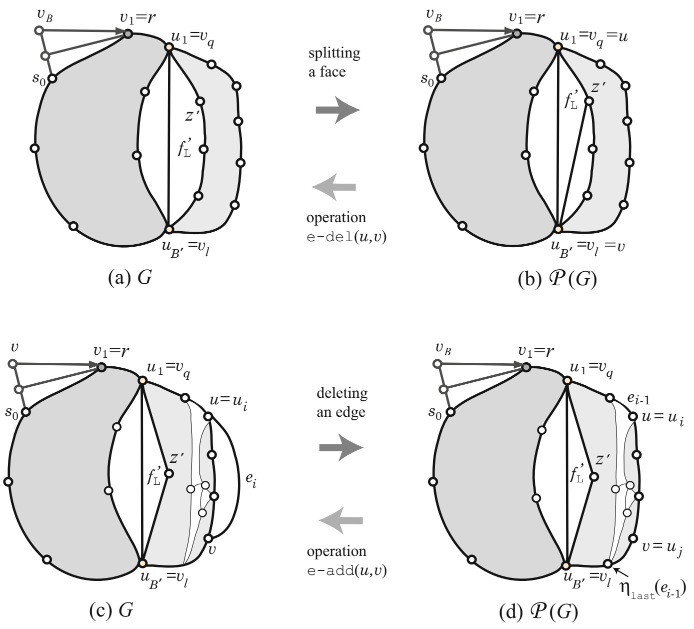


Fig. 2. (a), (b) Illustration for the parent in P3 and operation `e-del`; (c), (d) illustration for the parent in P4 and operation `e-add`

Let $f_{max}(G)$ denote the maximum size of inner faces in G, and define function $\Phi(G) = |F(G)| + 2B + |\psi(G)|$, where $F(G)$ denotes the set of faces in G.

Lemma 1. *For each rooted graph $G \in \mathcal{G}_{\mathtt{int}}(n) - \{F_n\}$ with $n \geq 4$, its parent $\mathcal{P}(G)$ is defined by one of the above six conditions, and it holds $f_{max}(G) \geq f_{max}(\mathcal{P}(G))$, $3 \leq \Phi(G) < \Phi(\mathcal{P}(G)) \leq 4n-4$ and $\kappa(G) \geq \kappa(\mathcal{P}(G))$.*

3.2 Children of Internally Triconnected Rooted Plane Graphs

Let G be an internally triconnected rooted plane graph with $n \geq 4$ vertices. A rooted plane graph G' is called a *child* of G if $G = \mathcal{P}(G')$. Let $\mathcal{C}(G)$ denote the set of all children of G, and let $\mathcal{C}^i(G)$, $i = 1, 2, \ldots, 6$ denote the set of all children G' of G such that $\mathcal{P}(G')$ is given by definition Pi of parents. Let $\beta[v_q, v_\ell]$ be the first reducible path in G, and u_p be the critical vertex of $G[v_q, v_\ell]$. Now we characterize the children in each $\mathcal{C}^i(G)$ by introducing corresponding six operations, `create`, `f-merge`, `e-del`, `e-add`, `v-insert` and `v-add` as follows.

C1 Assume that $|\psi(G)| \geq 1$ and $i \leq \ell - 1$. For the outer edge $(u = v_i, v = v_{i+1})$ $(1 \leq i \leq B' - 2 + i)$, let G_i^1 be obtained from G by `create`(u, v), an operation which adds a new outer vertex z' together with two new outer edges (z', u) and (z', v), and reduces $\psi(G)$ by 1. This creates a new reducible path $\beta[u, v]$. See Fig. 1(a)-(b).

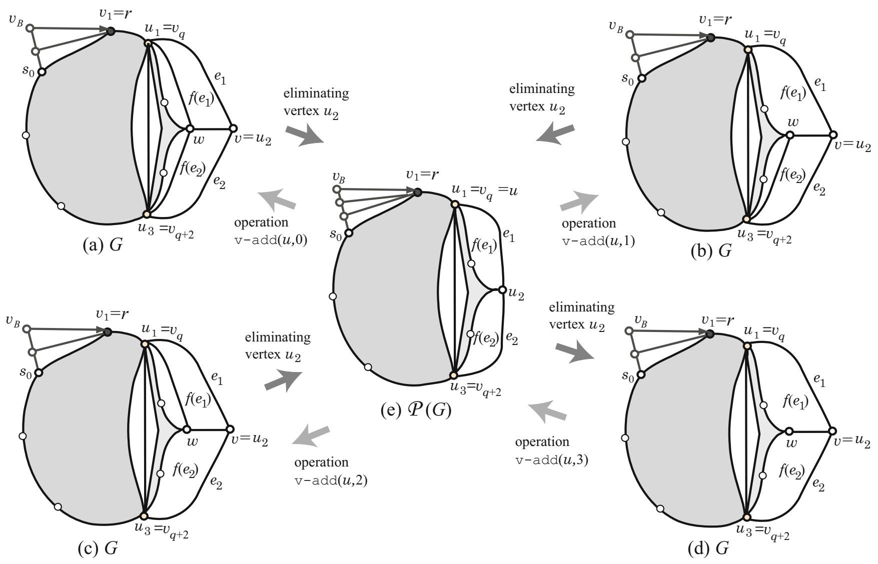


Fig. 3. Illustration for the parent in P5 and operation `v-add`

C2 Assume that $\{v_q, v_\ell\}$ is a cut-pair, and $(v_q, v_\ell) \in E(G)$. For the two outer vertices $u = v_q$ and $v = v_\ell$, let G^2 be obtained from G by `f-merge`(u, v), an operation which deletes the edge (u, v), merging the two faces incident to (u, v) in G into one. See Fig. 1(c)-(d) and Fig. 5(a)-(b).

C3 Assume that $|f'_{\mathtt{L}}| = 3$, $H = G[v_q, v_\ell]$ has no separating r'-face, and deg $(u_{B'}; H) \geq 4$, For two outer vertices $u = v_q$ and $v = v_\ell$, let G^3 be obtained from G by `e-del`(u, v), an operation which removes the edge (v, z') between $v = u_{B'}$ and the second leftmost neighbour z' of r'. See Fig. 2(a)-(b) and Fig. 5(b)-(c).

C4 Assume that $|f'_{\mathtt{L}}| = 3$, $B' \geq 4$, $i \leq p$ and $j \leq k_{i-1}$ For two outer vertices $u = u_i$ and $v = u_j$, where $i \in [1, B' - 2]$ and $j \in [i + 2, B']$, let $G^4_{i,j}$ be obtained from G by `e-add`(u, v), an operation which adds a new outer edge (u, v) as the ith outer edge e_i. See Fig. 2(c)-(d).

C5 Assume that $B' = 3$, $|f'_{\mathtt{L}}| = 3$ and $|\psi(G)| \geq 1$. Define four subsets of $\{e_1, e_2\}$ by $E_0 = \emptyset$, $E_1 = \{e_1\}$, $E_2 = \{e_2\}$, and $E_3 = \{e_1, e_2\}$. For the second outer vertex $u = v_2$ in $G[v_q, v_\ell]$ and $h \in \{0, 1, 2, 3\}$, let G^5_h be obtained from G by `v-add`(u, h), an operation which reduces $\psi(G)$ by 1 by reducing $\psi(G)$ by 1, and adds a new outer vertex v together with three edges (v, v_1), (v, v_2) and (v, v_3), deleting an edge set E_h so that $deg(x; G^5_h[v_q, v_\ell]) \geq 3$ for all $x \in \mathit{\Gamma}(v; G^5_h[v_q, v_\ell])$. See Fig. 3(a)-(e).

C6 Assume that $|\psi(G)| \geq 1$ and $|f'_{\mathtt{L}}| = 3$. Let $e_i = (a = v_i, b = v_{i+1})$ be an outer edge such that $|f(e_i)| = 3$, $V(f(e_i)) = \{a, b, w\}$ and $i \leq p$ (resp., $i \leq p - 1$) if the first removable element in $G[v_q, v_\ell]$ is an edge $e_p = (u_p, u_{p+1})$ (resp., a vertex u_p). Define four subsets of $E(f(e_i))$ by $E_0 = \emptyset$, $E_1 = \{(a, w)\}$, $E_2 = \{(b, w)\}$, and $E_3 = \{(a, w), (b, w)\}$. For $h \in \{0, 1, 2, 3\}$, let $G^6_{i,h}$ be obtained from G by v-insert(e_i, h), an operation which reduces $\psi(G)$ by 1, replaces e_i with three edges (a, v), (v, b) and (v, w), introducing a new vertex v, and delete edge set E_h, so that $deg(x; G^6_{i,h}[v_q, v_\ell]) \geq 3$ for all $x \in \Gamma(v; G^6_{i,h}[v_q, v_\ell])$ and $deg(u_i; G^6_{i,h}[v_q, v_\ell]) \geq 4$ (where $deg(u_1; G^6_{i,h}[v_q, v_\ell]) = 3$ is allowed). See Fig. 4(a)-(e) and Fig. 5(c)-(d).

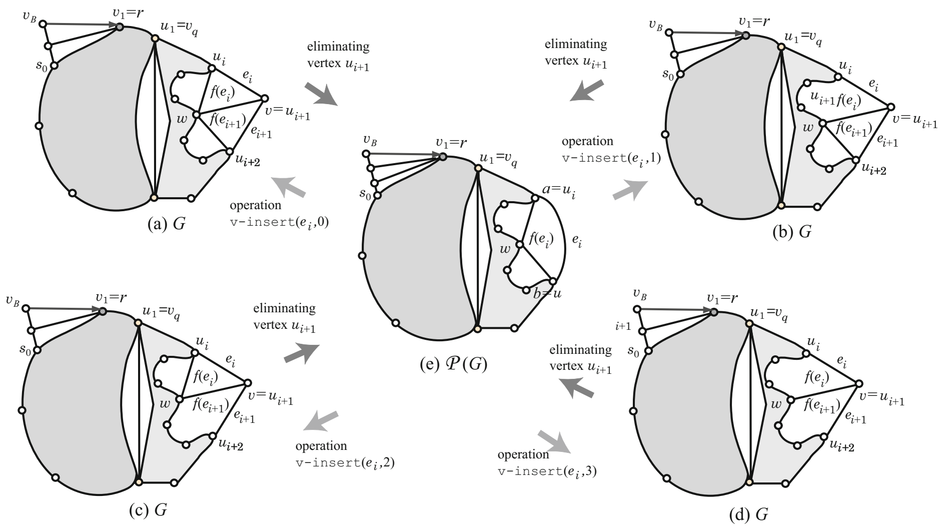


Fig. 4. Illustration for the parent in P6 and operation v-insert

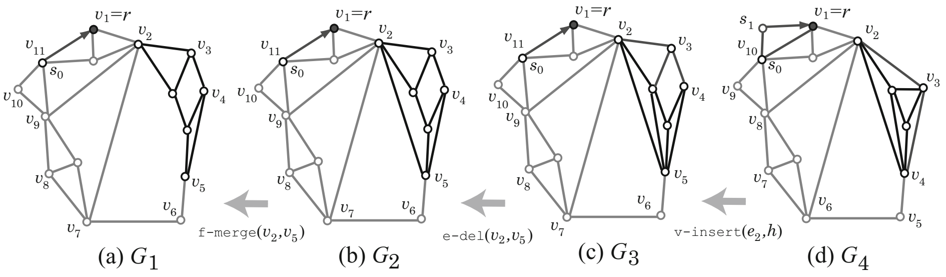


Fig. 5. Internally triconnected rooted plane graphs. (a) G_1; (b) $G_2 = \mathcal{P}(G_1)$; (c) $G_3 = \mathcal{P}(G_2)$; (d) $G_4 = \mathcal{P}(G_3)$.

4 Algorithm

To generate all plane graphs $G' \in \mathcal{C}(G) \cap \mathcal{G}_{\texttt{int}}(n, g)$, we construct only those $G' \in \mathcal{C}(G)$ such that the new face introduced by e-add and the face $f(e_i)$ and/or $f(e_{i+1})$ enlarged by f-merge, e-del, v-insert or v-add are of length at most g. For this, we execute the following recursive procedure Gen$(G, v_q, B', \varepsilon)$, with the initial setting $G := F_n$, $r' := r$, $B' := 2$ and $\varepsilon := \emptyset$, where the second and third arguments v_q and B' denote the starting vertex v_q and ending vertex $v_\ell = v_{q-1+B'}$ of the first reducible subgraph $G[v_q, v_\ell]$ of the first argument G, respectively, and and ε stands for the first removable element in $G[v_q, v_\ell]$. We set $\varepsilon = \emptyset$ if $G[v_q, v_\ell]$ is a triangle C_3 or $B' = 2$ which is used for initialize $G := F_n$.

In $\textsc{Gen}(G, r', B', \varepsilon)$, we first generate children $G^1_i, G^2_h, G^3_h \in \mathcal{C}(G)$ if any, and then generate children $G^5_i, G^6_{i,h}, G^4_{i,i+\Delta} \in \mathcal{C}(G)$ for all vertices u_i, $i = 1, 2, \ldots, p$ in the active path of G by increasing step size $\Delta \geq 2$ by 1.

```
**Procedure** $\textsc{Gen}(G, r', B', \varepsilon)$
Input: An internally triconnected rooted plane graph $G \in \mathcal{G}_{\texttt{int}}(n, g)$, the starting
vertex $r' = v_q$, the ending vertex $v_\ell$ ($\ell = q - 1 + B'$), and the first removable
element $\varepsilon$ of $G[v_q, v_\ell]$ (if any), where $\varepsilon \neq \emptyset$ is either an edge $e_p = (u_p, u_{p+1})$
or a vertex $u_p$.
Output: All descendants $G' \in \mathcal{G}_{\texttt{int}}(n, g)$ of $G$.
begin
  if the depth of the current recursive call is odd then Output $G$ endif;
  /* Let $(v_1 = r, v_2, \ldots, v_B)$ denote the boundary of $G$ in the clockwise order,
     $(u_1 = r', u_2, \ldots, u_{B'})$ denote the boundary of $G[r' = v_q, u_{B'} = v_\ell]$ in the
     clockwise order, and $e_i = (u_i, u_{i+1})$ denote the edge between $u_i$ and $u_{i+1}$ */
  if $|\psi(G)| \geq 1$ then
    for $i = 1, 2, \ldots, \ell - 1$ do
      Let $G'$ be the graph $G^1_i$ obtained from $G$ by create$(v_i, v_{i+1})$;
      $\textsc{Gen}(G', v_i, 3, \emptyset)$
    endfor
  endif;
  if $\{v_q, v_\ell\}$ is a cut-pair, $(v_q, v_\ell) \in E(G)$, $|f| + |f'| - 2 \leq g$ for the two inner
    faces $f$ and $f'$incident to edge $(v_q, v_\ell)$ then
    Let $G'$ be the graph $G^2_h$ obtained from $G$ by f-merge$(v_q, v_\ell)$;
    $\textsc{Gen}(G', v_q, B', \varepsilon)$
  endif;
  if $(v_q, v_\ell), (u_{B'}, z') \in E(G)$, $\kappa(G - (u_{B'}, z')) \geq 3$ and $|f| < g$ for the
    $u_{B'}$-face $f$ adjacent to $f'_{\mathrm{L}}$ then
    Let $G'$ be the graph $G^3_h$ obtained from $G$ by e-del$(v_q, v_\ell)$;
    $\textsc{Gen}(G', v_q, B', \emptyset)$
  endif;
  if $(v_q, v_\ell), (u_{B'}, z') \in E(G)$, $B' = 3$ and $|\psi(G)| \geq 1$ then
    for $h = 0, 1, 2, 3$ do
      Let $G'$ be the graph $G^5_h$ obtained from $G$ by v-add$(v_{k+1}, h)$, and
      let $v$ be the newly introduced vertex;
      if $deg(x; G^5_h[v_q, v_\ell]) \geq 3$ for all $x \in \Gamma(v; G')$ and $|f| \leq g$ for all $v$-faces $f$ in $G'$
      then $\textsc{Gen}(G', v_i, 3, \varepsilon := \emptyset)$ endif
    endfor
  endif;
  if $(v_q, v_\ell), (u_{B'}, z') \in E(G)$ and $|\psi(G)| \geq 1$ then
    for $i = 1, 2, \ldots, q$ do
      if $|f(e_i)| = 3$, and "$i < p$" or "$i = p$ and $\varepsilon$ is an edge" then
```

```
            for $h = 0, 1, 2, 3$ do
               Let $G'$ be the graph $G^6_{i,h}$ obtained from $G$ by $\texttt{v-insert}(e_i, h)$, and
               let $v$ be the newly introduced vertex;
               if $deg(x; G^6_{i,h}[v_q, v_\ell]) \geq 3$ for all $x \in \Gamma(v; G')$, $deg(u_i; G^6_{i,h}[v_q, v_\ell]) \geq 4$
               (when $i \neq 1$) and and $|f| \leq g$ for all $v$-faces $f \in F(v; G')$ then
                  Gen$(G', v_q, B' + 1, v)$ endif
            endfor endif endfor
   endif;
   if $(v_q, v_\ell), (u_{B'}, z') \in E(G)$ and $B' \geq 4$ then
      for $\Delta = 2, 3, \ldots, \min\{B' - 1, g - 1\}$ do
         $i := 1$;
         while  $i + \Delta \leq k_{i-1}$ and $i \leq p$ do
            /* $k_{i-1}$ be the index $k \in [i + 1, B']$ of $u_k = \tau_{\texttt{last}}(e_{i-1})$ and $k_0 \coloneqq B'$ */
            $j := i + \Delta$;
            Let $G'$ be the graph $G^4_{i,j}$ obtained from $G$ by $\texttt{e-add}(u_i, u_j)$;
            Gen$(G', v_q, B' - \Delta + 2, e_i = (u_i, u_j))$;
            $i := i + 1$
         endwhile endfor endif;
   if the depth of the current recursive call is even then Output $G$ endif;
   Return
end.
```

We omit an analysis for time and complexity of the algorithm due to space limitation (see [12] for the details), mentioning that the following lemma used to show that Gen attains $O(1)$-time delay in the worst case using $O(n)$ space.

Lemma 2. *Let $H = G[v_q, v_\ell]$ be a triconnected plane graph rooted at $r' = u_1$ such that $|f'_{\mathrm{L}}| = 3$.*

(i) *For each edge e in the active path of H, $\tau_{\texttt{last}}(e)$ can be found in $O(1)$ time and $O(n)$ space.*
(ii) *Whether H has no separating r'-face or not can be tested in $O(1)$ time and $O(n)$ space.*

This is another technical difficulty in getting an $O(1)$-time delay implementation of this algorithm. Based on an observation of how vertex cuts with 3 vertices change for each operation, a solution is found by keeping only a simple but sufficient data in $O(1)$ time and $O(n)$ space. A similar data is used to deal with triconnected rooted plane graphs [11].

Theorem 1. *For integers $n \geq 1$ and $g \geq 3$, all rooted plane graphs in $\mathcal{G}_{\texttt{int}}(n, g) - \mathcal{G}_3(n, g)$ can be enumerated without duplication in $O(n)$ space and in $O(1)$-time delay by outputting the difference from the previous output after an $O(n)$ time initialization.*

We can use our algorithm for generating unrooted plane graphs. During an execution of Gen we check in $O(n^2)$ time whether a newly generated rooted graph G is the representative among rooted graphs with the same plane graphs or not by computing its signature [1].

Corollary 1. *For a given integer $n \geq 1$ and $g \geq 3$, all internally triconnected planar graphs G with exactly n vertices such that $\kappa(G) = 2$ and the size of each inner face is at most g can be enumerated without duplication in $O(n)$ space and in $O(n^3)$-time delay on average.*

5 Concluding Remarks

In this paper, we gave an enumeration algorithm for the class of internally triconnected rooted plane graphs with exactly n vertices and bounded inner face size g. The algorithm can output only internally triconnected rooted plane graphs which are not triconnected. Since an $O(1)$-time delay algorithm for the class of triconnected rooted plane graphs is available [11], we can enumerate these two classes separately.

It is our future work to design enumeration algorithms for rooted plane graphs with a higher vertex-connectivity and to take into account the reflectional symmetry around the root, as studied in our companion paper [8,9].

References

1. Hopcroft, J.E., Wong, J.K.: Linear time algorithm for isomorphism of planar graphs. In: STOC 1974, pp. 172–184 (1974)
2. Horváth, T., Ramon, J., Wrobel, S.: Frequent subgraph mining in outerplanar graphs. In: Proc. 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 197–206 (2006)
3. Li, Z., Nakano, S.: Efficient generation of plane triangulations without repetitions. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 433–443. Springer, Heidelberg (2001)
4. Nakano, S.: Efficient generation of triconnected plane triangulations. Computational Geometry Theory and Applications 27(2), 109–122 (2004)
5. Thomassen, C.: Plane representations of graphs. In: Bondy, J.A., Murty, U.S.R. (eds.) Progress in Graph Theory, pp. 43–69. Academic Press, London (1984)
6. Tutte, W.T.: Convex representations of graphs. Proc. of London Math. Soc. 10(3), 304–320 (1960)
7. Yamanaka, K., Nakano, S.: Listing all plane graphs. In: Nakano, S.-i., Rahman, M. S. (eds.) WALCOM 2008. LNCS, vol. 4921, pp. 210–221. Springer, Heidelberg (2008)
8. Zhuang, B., Nagamochi, H.: Enumerating rooted biconnected planar graphs with internally triangulated faces, Dept. of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Technical Report 2009-018 (2009)
9. Zhuang, B., Nagamochi, H.: Efficient generation of symmetric and asymmetric biconnected rooted outerplanar graphs. In: The 3rd Annual Meeting of Asian Association for Algorithms and Computation (AAAC), POSTECH, Pohang, Korea, April 17-19 (to appear, 2010)
10. Zhuang, B., Nagamochi, H.: Enumerating biconnected rooted plane graphs, Dept. of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Technical Report 2010-001 (2010)

11. Zhuang, B., Nagamochi, H.: Listing triconnected rooted plane graphs, Dept. of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Technical Report 2010-002 (2010)
12. Zhuang, B., Nagamochi, H.: Generating internally triconnected rooted plane graphs, Dept. of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Technical Report 2010-003 (2010)

Author Index