LNCS 5344

Hajo Broersma
Thomas Erlebach
Tom Friedetzky
Daniel Paulusma (Eds.)

Graph-Theoretic Concepts in Computer Science

34th International Workshop, WG 2008
Durham, UK, June/July 2008
Revised Papers

Springer

Hajo Broersma Thomas Erlebach
Tom Friedetzky Daniel Paulusma (Eds.)

Graph-Theoretic Concepts in Computer Science

34th International Workshop, WG 2008
Durham, UK, June 30 – July 2, 2008
Revised Papers

Springer

Volume Editors

Hajo Broersma
Tom Friedetzky
Daniel Paulusma
Durham University, Department of Computer Science
South Road, Durham DH1 3LE, UK
E-mail: {hajo.broersma, tom.friedetzky, daniel.paulusma}@durham.ac.uk

Thomas Erlebach
University of Leicester, Department of Computer Science
University Road, Leicester, LE1 7RH, UK
E-mail: t.erlebach@mcs.le.ac.uk

Library of Congress Control Number: Applied for

CR Subject Classification (1998): F.2, G.2, G.1.6, G.1.2, E.1, I.3.5

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

ISSN 0302-9743
ISBN-10 3-540-92247-4 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-92247-6 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12585687 06/3180 5 4 3 2 1 0

Preface

The 34th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2008) took place in Van Mildert College at Durham University, UK, 30 June – 2 July 2008. The approximately 80 participants came from various countries all over the world, among them Australia, Brazil, Canada, Chile, Czech Republic, France, Greece, Hungary, Israel, Italy, Japan, The Netherlands, Norway, Poland, Spain, Switzerland, UK and the USA.

WG 2008 continued the series of 33 previous WG conferences. Since 1975, the WG conference has taken place 21 times in Germany, four times in The Netherlands, twice in Austria as well as once in Italy, Slovakia, Switzerland, the Czech Republic, France, Norway and now in the UK.

The WG conference traditionally aims at uniting theory and practice by demonstrating how graph-theoretic concepts can be applied to various areas in computer science, or by extracting new problems from applications. The goal is to present recent research results and to identify and explore directions of future research.

The continuing interest in the WG conferences was reflected in the number and quality of submissions; 76 papers were submitted and in an evaluation process with four reports per submission, 30 papers were accepted by the Program Committee for the conference. Due to the high number of submissions and the limited schedule of 3 days, various good papers could not be accepted.

There were excellent invited talks by Giuseppe Di Battista (Università Roma Tre, Italy) on algorithmic aspects of (un)-stable routing in the Internet, by Leszek Gąsieniec (University of Liverpool, UK) on memory-efficient graph exploration, and by Martin Grohe (Humboldt-Universität zu Berlin, Germany) on algorithmic meta theorems.

The WG 2008 Best Student Paper Award was awarded to the paper "Faster Exact Bandwidth" by Marek Cygan and Marcin Pilipczuk.

We are grateful to all those who contributed to WG 2008, especially the authors for submitting so many good papers, the numerous referees, the speakers and the Program Committee. We would also like to thank the London Mathematical Society (LMS) and the Engineering and Physical Sciences Research Council (EPSRC) for financial support and INFO.RO for sponsoring the Best Student Paper Award.

October 2008

Hajo Broersma
Thomas Erlebach
Tom Friedetzky
Daniel Paulusma

Organization

The Tradition of WG

1975	U. Pape – Berlin, Germany
1976	H. Noltemeier – Göttingen, Germany
1977	J. Mühlbacher – Linz, Austria
1978	M. Nagl, H.J. Schneider – Castle Feuerstein, Germany
1979	U. Pape – Berlin, Germany
1980	H. Noltemeier – Bad Honnef, Germany
1981	J. Mühlbacher – Linz, Austria
1982	H.J. Schneider, H. Göttler – Neuenkirchen, Germany
1983	M. Nagl, J. Perl – Haus Ohrbeck near Osnabrück, Germany
1984	U. Pape – Berlin, Germany
1985	H. Noltemeier – Castle Schwanberg near Würzburg, Germany
1986	G. Tinhofer, G. Schmidt – Bernried near Munich, Germany
1987	H. Göttler, H.J. Schneider – Kloster Banz near Bamberg, Germany
1988	J. van Leeuwen – Amsterdam, The Netherlands
1989	M. Nagl – Castle Rolduc, The Netherlands
1990	R.H. Möhring – Berlin, Germany
1991	G. Schmidt, R. Berghammer – Fischbachau near Munich, Germany
1992	E.W. Mayr – Wiesbaden-Naurod, Germany
1993	J. van Leeuwen – Utrecht, The Netherlands
1994	G. Tinhofer, E.W. Mayr, G. Schmidt – Herrsching near Munich, Germany
1995	M. Nagl – Aachen, Germany
1996	G. Ausiello, A. Marchetti-Spaccamela – Como, Italy
1997	R.H. Möhring – Berlin, Germany
1998	J. Hromkovič, O. Sýkora – Smolenice Castle, Slovak Republic
1999	P. Widmayer – Ascona, Switzerland
2000	D. Wagner – Konstanz, Germany
2001	A. Brandstädt – Boltenhagen near Rostock, Germany
2002	L. Kučera – Český Krumlov, Czech Republic
2003	H.L. Bodlaender – Elspeet, The Netherlands
2004	J. Hromkovič, M. Nagl – Bad Honnef, Germany
2005	D. Kratsch – Metz, France
2006	F.V. Fomin – Bergen, Norway
2007	A. Brandstädt, D. Kratsch, H. Müller – Dornburg near Jena, Germany
2008	H. Broersma, T. Erlebach – Durham, UK

Program Committee

Hajo Broersma	Durham University, UK (Co-chair)
Artur Czumaj	University of Warwick, UK
Thomas Erlebach	University of Leicester, UK (Co-chair)
Mike Fellows	The University of Newcastle, Australia
Fedor V. Fomin	University of Bergen, Norway
Juraj Hromkovic	ETH Zürich, Switzerland
Jan Kratochvíl	Charles University, Prague, Czech Republic
Luděk Kučera	Charles University, Prague, Czech Republic
Jan van Leeuwen	Universiteit Utrecht, The Netherlands
Alberto Marchetti-Spaccamela	Sapienza University of Rome, Italy
Hartmut Noltemeier	Universität Würzburg, Germany
Christophe Paul	CNRS, LIRMM, Montpellier, France
Andrzej Pelc	Université du Québec, Canada
Ingo Schiermeyer	TU Bergakademie Freiberg, Germany
Jan Arne Telle	University of Bergen, Norway
Takeaki Uno	National Institute of Informatics (NII), Japan
Dorothea Wagner	Universität Karlsruhe (TH), Germany
Peter Widmayer	ETH Zürich, Switzerland
Shmuel Zaks	Technion, Haifa, Israel

Local Organization

Hajo Broersma
Tom Friedetzky
Daniel Paulusma

Additional Reviewers

Gill Barequet
Reinhard Bauer
Michael Baur
Hans Bodlaender
Hans-Joachim Böckenhauer
Anthony Bonato
Magnus Bordewich
Andreas Brandstädt
Stephan Brandt
Binh-Minh Bui-Xuan
Jurek Czyzowicz
Daniel Delling
Camil Demetrescu
Stephan Dempe
Giuseppe Di Battista
Zdeněk Dvořák
Louis Esperet
Jiří Fiala
Michele Flammini
Paolo Giulio Franciosa
Karin Freiermuth
Dalibor Frončeк
Marco Gaertler
Philippe Gambette
Tomáš Gavenčiak
Robert Görke
Petr Golovach
Martin Golumbic
Fabrizio Grandoni
Gregory Gutin

Michel Habib
Pinar Heggernes
Jan van den Heuvel
Petr Hliněný
David Ilcinkas
Bill Jackson
Stanislav Jendroľ
Matthew Johnson
Mamadou Moustapha Kanté
Bastian Katz
Michael Kaufmann
Arnfried Kemnitz
Shuji Kijima
Lefteris Kirousis
Masashi Kiyomi
Ekkehard Köhler
Petr Kolman
Daniel Kráľ
Rastislav Královič
Richard Královič
Dieter Kratsch
Marcus Krug
Isabella Lari
Van Bang Le
Vincent Limouzy
Giuseppe Liotta
Martin Mareš
Euripides Markou
Steffen Mecke
Klaus Meer
Sascha Meinert
Daniel Meister
Tobias Mömke
Gianpiero Monaco
Luca Moscardelli
Haiko Müller
Nicolas Nisse
Martin Nöllenburg
Hartmut Noltemeier
Yoshio Okamoto
Sang-il Oum
Daniel Paulusma
David Peleg
Martin Pergel
Bert Randerath
Dieter Rautenbach
Udi Rotics
Ignaz Rutter
Toshiki Saitoh
Saket Saurabh
Sebastian Seibert
Brigitte Servatius
Mordechai Shalom
Joachim Spoerhase
Björn Steffen
Lorna Stewart
Iain Stewart
Jayme Szwarcfiter
Tami Tamir
Ioan Todinca
Regina Tyshkevich
Walter Unger
Ugo Vaccaro
Yngve Villanger
Margit Voigt
Hans-Christoph Wirth
David Wood
Xiao Zhou

Table of Contents

(Un)-Stable Routing in the Internet: A Survey from the Algorithmic Perspective

Luca Cittadini, Giuseppe Di Battista, and Massimo Rimondini

Roma Tre University
{ratm,gdb,rimondin}@dia.uniroma3.it

Abstract. The Internet is a huge and complex network in which Internet Service Providers (ISPs) compete for revenue. In order to support the establishment of commercial agreements, routing in the Internet must therefore allow each ISP to autonomously set up its own routing policies while ensuring global connectivity. The Border Gateway Protocol (BGP) allows to achieve both these goals, and is the currently adopted protocol for Internet routing.

It can be shown that the interaction of not-so-unlikely BGP configurations adopted at different ISPs can cause permanent oscillations of routing. Several models and algorithms have been proposed in the literature to study routing oscillations. The goal of this paper is to provide a survey of state of the art contributions in this field with an emphasis on the algorithmic aspects.

1 Introduction

Internet interdomain routing relies on the Border Gateway Protocol (BGP). BGP [1] allows each Internet Service Provider (ISP) to autonomously set up its own routing policies while ensuring global connectivity. ISPs negotiate commercial and political agreements and subsequently connect and configure BGP routers to exchange reachability information to establish paths to a set of destinations.

Each ISP owns an *Autonomous System* (*AS*) that comprises all the equipments it administrates. Roughly speaking, BGP works as follows. An AS (say AS0) tells (announces) its neighboring ASes of its existence. Its neighbors, according to the commercial agreements of their ISPs, propagate this information to other ASes, that, in turn, do the same. In this way, after a while, the entire Internet knows about the existence of AS0.

While the announcement about AS0 is propagated through the Internet, each traversed AS adds (prepends) its identifier to the announcement. In this way an announcement carries in itself the entire path to reach AS0. Each AS, according to its policies, can choose to discard some announcements and can select its favorite one.

Unfortunately, it can be shown that the interaction of not-so-unlikely BGP configurations adopted at different ISPs can cause permanent oscillations of Internet routing (see, e.g., [2,3]).

In this paper we first introduce a well known formulation of the BGP routing stability problem, based on commonly adopted models to capture BGP routing dynamics. We then propose a classification of significant instances of the stability problem according to their convergence properties. Algorithms for computing stable states are then described, together with sufficient conditions that lead to guaranteed stability. We also discuss the relationships between the classical models for stability and other variants that accommodate link costs or account for the existence of commercial relationships in the routing system. Very recent results have been obtained by studying the stability problem with a game theoretic approach. These results are also presented.

Section 2 discusses models for BGP. Section 3 considers configurations that may admit multiple stable states and conditions for guaranteed convergence. Section 4 presents algorithms for computing stable states. Section 5 shows the interplay between stability and real world constraints. Conclusions are drawn in Section 6. Most of the presented results come from the literature, while some are original contributions.

A way to read this survey is to look at Fig. 3, which provides an at-a-glance view of the relationships between classes of instances with different convergence properties.

2 Models for BGP and the Stability Problem

Several approaches have been proposed to model BGP and to study its stability. A pioneering work on this subject is in [4], that proposed the *return graph*, while algebraic models have been shown in [5,6,7]. In this paper we adopt the *Simple Path Vector Protocol* (SPVP) model [8], that is the reference point of most of the scientific contributions on BGP stability.

We now define an SPVP instance. Let $G = (V, E)$ be an undirected graph, with vertex set $V = \{0, 1, \ldots, n\}$ and edge set E. The graph G is used to encode the AS-level topology of the network under consideration, in such a way that vertices correspond to ASes and edges correspond to adjacency relationships between ASes (also called *peerings* by operators).

A *path* P in G is a sequence of $k+1$ vertices $P = (v_k\ v_{k-1}\ \ldots\ v_1\ v_0)$, $v_i \in V$, such that $(v_i, v_{i-1}) \in E$ for $0 < i \leq k$. Vertex v_{k-1} is the *next hop* of v_k in P. We denote the empty path by ϵ. The *concatenation* of two nonempty paths $P = (v_k\ v_{k-1}\ \ldots\ v_i)$, $k \geq i$, and $Q = (v_i\ v_{i-1}\ \ldots\ v_0)$, $i \geq 0$, denoted as PQ, is the path $(v_k\ v_{k-1}\ \ldots\ v_i\ v_{i-1}\ \ldots\ v_0)$. We assume that $P\epsilon = \epsilon P = \epsilon$. In the SPVP model each vertex in $V - \{0\}$ attempts to establish a path to a single vertex 0. Note that, since BGP manages each destination separately, this is not a limitation. Each vertex $u \in V$ is assigned a set of *permitted paths* $\mathcal{P}^u$. All the paths in $\mathcal{P}^u$ are simple (i.e., no repeated vertices), start from u and end in 0, and represent the paths that u can use to reach 0. The empty path represents unreachability of 0 and is permitted at each vertex $u \neq 0$. Let $\mathcal{P}^0 = \{(0)\}$ and $\mathcal{P} = \bigcup_{u \in V} \mathcal{P}^u$.

For each vertex $u \in V$, a *ranking function* $\lambda^u : \mathcal{P}^u \rightarrow \mathbb{N}$ determines the relative level of preference $\lambda^u(P)$ assigned by u to path P. If $P_1, P_2 \in \mathcal{P}^u$ and $\lambda^u(P_2) < \lambda^u(P_1)$, then P_2 is *preferred* over P_1. Let $\Lambda = \{\lambda^u | u \in V\}$. Ranking functions in Λ describe the BGP routing policies.

According to the model in [2], the following conditions hold on the paths, for each vertex $u \in V - \{0\}$:

i. $\forall P \in \mathcal{P}^u, P \neq \epsilon$: $\lambda^u(P) < \lambda^u(\epsilon)$ (unreachability is the last resort);
ii. $\forall P_1, P_2 \in \mathcal{P}^u, P_1 \neq P_2 : \lambda^u(P_1) = \lambda^u(P_2) \Rightarrow P_1 = (u\ v)P'_1, P_2 = (u\ v)P'_2$, (strict ranking is assumed on all the paths but those with the same next hop).

An instance S of SPVP is a triple $(G, \mathcal{P}, \Lambda)$. See an example in Figure 2(a). The graphical convention adopted throughout the paper is the same as in [2], namely each vertex u is equipped with a list of paths representing $\mathcal{P}^u$ sorted by increasing values of λ^u. The empty path and $\mathcal{P}^0$ are omitted for brevity. We assume that the *size* of S is the size of $\mathcal{P}$.

A *path assignment* π is a function that maps each vertex $u \in V$ to a permitted path $\pi(u) \in \mathcal{P}^u$. This represents the fact that vertex u is using path $\pi(u)$ to reach 0. We have that $\pi(0) = (0)$ and, if $\pi(u) = \epsilon$, then u cannot reach vertex 0.

In SPVP vertices asynchronously exchange messages containing paths to 0. We assume that edges introduce a finite delay on message delivery. Each vertex u keeps in a *routing information base* $\text{rib}_t(u)$ the path it adopts at time t to reach vertex 0. If a vertex u receives from a neighbor w at time t an announcement containing a path P, first of all u checks whether $(u)P$ is permitted, namely if $(u)P \in \mathcal{P}^u$. If this is the case, u puts $(u)P$ into a data structure called $\text{rib-in}_t(u \Leftarrow w)$, which is used to store the latest path received from w. If $(u)P$ is not permitted, u puts ϵ in $\text{rib-in}_t(u \Leftarrow w)$. Then, u checks whether the currently selected path, stored in $\text{rib}_{t-1}(u)$, is the currently available best path. If this is not the case, u selects the best ranked path among those in all its rib-in_t data structures and stores it in $\text{rib}_t(u)$. We refer to such a path as $\text{best}_t(u) = \underset{v|(u,v)\in E}{\arg\min}\ \lambda^u(\text{rib-in}_t(u \Leftarrow v))$. Afterwards, u announces $\text{best}_t(u)$ to all its neighbors $v \mid (u,v) \in E$.

An *activation sequence* [8] $\sigma = (A_0\ A_1\ \ldots A_i \ldots)$ is a (possibly infinite) sequence where A_t is a set containing an ordered pair $(u,v)|(u,v) \in E$ for each vertex v that processes a message from u at time t. We say that edge (u,v) is *activated* at time t. Since the delay introduced by edges is finite, messages are eventually delivered. An activation sequence is *fair* if any edge $(u,v) \in E$ is eventually activated whenever u sends a message to its neighbors. In a real world scenario, all those messages that are not filtered by explicitly configured policies are never discarded. Therefore, in the following we will be mostly considering fair activation sequences.

As SPVP operates within the network, the routing evolves through different path assignments π_t, where $\pi_t(u) = \text{rib}_t(u)$, until a stable path assignment is reached (we say that SPVP *converges* to that path assignment). A path assignment π_t is *stable* if, for each $u \in V$, $\pi_t(u) = \text{best}_t(u)$. Once a stable path assignment is reached, no further messages are generated in the network.

Obviously, the existence of a stable path assignment is a crucial requirement for a network running BGP. For this reason the *Stable Paths Problem* has been defined in [2] as follows:

Problem 1 (Stable Paths Problem). Given an instance S of SPVP, does S admit a stable path assignment?

Theorem 1. *The Stable Paths Problem is NP-complete [2].*

If a stable path assignment exists for S, it is a *solution* for the problem. We define Solvable (Unsolvable) as the set of SPVP instances that admit (do not admit) a solution.

For example, instance Good-Gadget (Fig. 2(a)) belongs to Solvable since it admits a stable path assignment. In fact, it is easy to check that the path assignment $\pi(1) = (1\ 3\ 0)$, $\pi(2) = (2\ 0)$, $\pi(3) = (3\ 0)$, $\pi(4) = (4\ 2\ 0)$ is stable. On the other hand, instance Bad-Gadget (Fig. 2(b)) belongs to Unsolvable. In fact, in any stable path assignment π there must be at least one vertex that picks the direct path to 0. Assume that $\pi(2) = (2\ 0)$. For π to be stable, we must have $\pi(3) = (3\ 2\ 0)$, which in turn implies $\pi(1) = (1\ 0)$. Note that now $(2\ 0)$ is not the best available path at vertex 2, hence no path assignment can have $\pi(2) = (2\ 0)$. The same argument applies symmetrically to the other vertices.

Throughout this paper we shall discuss other notable classes of SPVP instances besides Solvable and Unsolvable. The relationships between those classes are illustrated in Fig. 3.

3 Multiple Stable States and Guaranteed Convergence

One might ask the question whether there are SPVP instances that admit more than one solution. Let Unique be the class of SPVP instances that have exactly one solution. We have that Good-Gadget belongs to Unique.

Theorem 2. *Unique $\subset$ Solvable.*

Proof. Instance Disagree [9] (Fig. 2(c)) has two solutions. It is easy to see that both $\pi(1) = (1\ 0)$, $\pi(2) = (2\ 1\ 0)$ and $\pi'(1) = (1\ 2\ 0)$, $\pi'(2) = (2\ 0)$ are stable path assignments.

Property 1. There are instances of SPVP with an exponential number of solutions.

Proof. An instance that contains n distinct and independent Disagree structures has 2^n solutions.

Property 2. For each non-negative integer k there exists an instance of SPVP with k solutions.

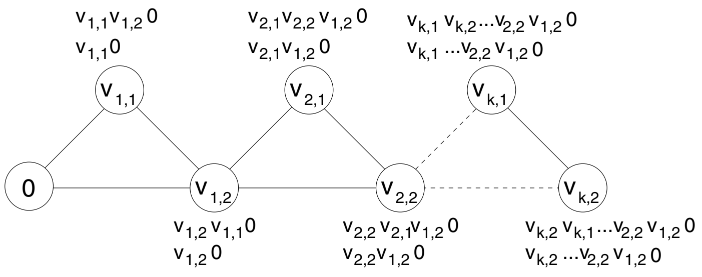


Fig. 1. An instance of SPVP with $k + 1$ solutions

Proof. This is clearly true for $k = 0$ and $k = 1$. In fact, BAD-GADGET and GOOD-GADGET are examples of instances with no solutions and with only one solution, respectively. Fig. 1 shows the structure of a generic SPVP instance with $k+1$ solutions. The idea is to stack several DISAGREE gadgets, each of which adds a new stable path assignment to the SPVP instance. In particular, each triple of vertices $(v_{i,1}, v_{i,2}, v_{i-1,2})$, with $i > 0$ and $v_{0,2} = 0$, forms a DISAGREE. Each vertex $v_{i,j}$ can reach 0 by using a direct path $(v_{i-1,2}\ v_{i-2,2}\ \ldots\ 0)$. However, $v_{i,1}$ prefers the path via $v_{i,2}$, and $v_{i,2}$ prefers the path via $v_{i,1}$. With this construction, if a DISAGREE $(v_{i,1}, v_{i,2}, v_{i-1,2})$ stabilizes on $\pi_t(v_{i,1}) = (v_{i,1}\ v_{i-1,2}\ v_{i-2,2}\ \ldots\ 0)$ and $\pi_t(v_{i,2}) = (v_{i,2}\ v_{i,1})\pi_t(v_{i,1})$ at time t, then there exists a time $t' > t$ after which the vertices of any other DISAGREE $(v_{j,1}, v_{j,2}, v_{j-1,2})$, $j > i$ permanently select the empty path ϵ. On the other hand, if a DISAGREE stabilizes on $\pi_t(v_{i,1}) = (v_{i,1}\ v_{i,2})\pi_t(v_{i,2})$ and $\pi_t(v_{i,2}) = (v_{i,2}\ v_{i-1,2}\ \ldots\ 0)$, then the following DISAGREE $(v_{i+1,1}, v_{i+1,2}, v_{i,2})$ (if any) has two stable path assignments, and a similar argument can be applied.

In this way, every DISAGREE acts as a switch that can enable or disable the subsequent DISAGREE in the stack. The last DISAGREE $(v_{k,1}, v_{k,2}, v_{k-1,2})$ can arbitrarily reach one of its two stable states without influencing further gadgets. It is easy to check that this SPVP instance has exactly $k + 1$ stable states.

Even if an instance has a stable state, there may still be activation sequences that give rise to oscillations [10]. From the point of view of a network operator, it is interesting to know whether a given BGP configuration is *guaranteed* to converge to a stable state, regardless of any possible message orderings. Hence, we define the *Safety* problem [10,11]:

Problem 2 (Safety). Given an SPVP instance S, does S admit only finite fair activation sequences?

Theorem 3. *Safe* $\subset$ *Solvable.*

Proof. DISAGREE (Fig. 2(c)) is Solvable but not Safe. An infinite fair activation sequence $\sigma = (A_0 \ldots A_i \ldots)$ can be constructed as follows. Vertices 1 and 2 first learn the direct path to zero $A_0 = \{(0, 1), (0, 2)\}$. Then they periodically exchange information in a synchronous way, i.e. $A_t = \{(1, 2), (2, 1)\}$, $\forall t \geq 1$. It is easy to see that vertex 1 keeps selecting alternately paths $\pi_t(1) = (1\ 0)$ and $\pi_{t+1}(1) = (1\ 2\ 0)$ at time $t = 2n$. Symmetric considerations apply to vertex 2.

Theorem 4. *Safe ∪ Unique ⊂ Solvable.*

Proof. DISAGREE is neither Safe nor Unique.

Theorem 5. *Safe ∩ Unique ≠ Unique.*

Proof. Instance NAUGHTY-GADGET [10] (Fig. 2(d)) has a unique stable state but a persistent oscillation. The unique solution is the same as in GOOD-GADGET, while the persistent oscillation works in a similar way as in BAD-GADGET (see [12,2]).

In [2,10] it has been shown that, when an SPVP instance has multiple solutions, then a cyclic dependence on the ranking of paths can be identified. Even though this does not imply the existence of an infinite activation sequence, we were not able to find an SPVP instance that is Safe but not Unique. We therefore state the following:

Conjecture 1. Safe ∩ Unique = Safe.

The safety of SPVP has also been studied from a game theoretic perspective [13] in the so called *convergence game*. All the vertices but 0 are players. When allowed to play, a vertex selects the best path among those offered by its neighbors (best-reply dynamics). The payoff of vertex u is $\lambda^u(P)$ if u selects path P. The game has infinite rounds. SPVP safety translates to the convergence of best-reply dynamics in the convergence game. In this framework the following result has been proved:

Theorem 6. *The safety problem is PSPACE-complete [13].*

However, interpreting Theorem 6 from the SPVP point of view requires some care. First, [13] only considers activation sequences where vertices are activated one after the other. It is unclear whether an instance that is Safe in this model is also Safe in the more general model introduced above. Second, players of the convergence game can select a neighbor instead of a path.

4 Algorithms That Search for Stable States

In this section we restate and discuss a polynomial time GREEDY algorithm that has been proposed in [2] to check if an SPVP instance has a solution. GREEDY attempts to grow a solution by iteratively building a stable path assignment. If the algorithm terminates successfully, the path assignment defines a spanning tree that is a solution for the given instance. Otherwise, GREEDY is only able to identify a stable path assignment for a subset of the vertices.

The algorithm maintains a *stable set* of vertices for which there exists one permitted path that will be in *every* stable path assignment. The stable set at iteration i of the algorithm is denoted by V_i. Vertex 0 is always in the stable set, therefore let $V_0 = \{0\}$. As the stable set grows, a path assignment π defined on the vertices in V_i is iteratively built.

We say that a path P is *compatible with a path assignment* π if $P = P_1(u\ v)P_2$, where P_1 does not contain vertices in V_i, $(u, v) \in E$, $v \in V_i$, and $P_2 = \pi(v)$.

Algorithm Greedy is as follows. At iteration i, let P_v be the path with minimum $\lambda^v(P)$ among the paths at v compatible with π. If such a path does not exist, let $P_v = \epsilon$. If there exists a vertex $v \notin V_{i-1}$ such that P_v has a next hop in V_{i-1}, then construct V_i by adding v to V_{i-1} and set $\pi(v) = P_v$. If such a vertex v does not exist, then stop.

Intuitively, at each iteration vertex v is stabilized because its best compatible path directly reaches an already stabilized vertex. Observe that the algorithm terminates after at most $|V|$ iterations. A solution to the SPVP instance exists if, after k iterations, we end with $V_k = V$. The solution is given by π.

We define Greedy-Solvable as the set of SPVP instances that have a solution that is found by Greedy. It is easy to see that Good-Gadget $\in$ Greedy-Solvable. In fact, at the first iteration vertex 3 enters the stable set V_1, and $\pi(3) = (3\ 0)$. At the second iteration, the best path at vertex 1 is then directly connected to a stable node, hence 1 enters the stable set V_2, with $\pi(1) = (1\ 3\ 0)$. Observe that this makes path $(2\ 1\ 0)$ not compatible with π. Hence, the best compatible path at 2 is directly attached to a stable vertex (0), so 2 enters V_3, and $\pi(2) = (2\ 0)$. Finally, vertex 4 has now a best path which is directly attached to 2, so it enters V_4 with $\pi(4) = (4\ 2\ 0)$. Since $V_4 = V$, the algorithm successfully terminates.

Theorem 7. *Greedy-Solvable $\subset$ Safe $\cap$ Unique.*

Proof. Part 1 ($\subseteq$): If an SPVP instance S is successfully solved by Greedy, then S has a unique solution [2] and it is safe [12].

Part 2 ($\subset$): A safe SPVP instance that has a unique solution and that Greedy is unable to solve is Di-Safe-gree (Fig. 2(e)). The instance is safe since any fair activation sequence of SPVP is finite. In fact, in any fair activation sequence vertices 1, 2, and 3 learn about the direct path to 0. After that, edge $(3, 2)$ is eventually activated, and 2 learns about $(2\ 3\ 0)$. Henceforth, vertex 2 will be permanently unable to select $(2\ 0)$, in turn preventing vertex 1 from choosing $(1\ 2\ 0)$. Finally, after edge $(1, 2)$ is activated, 2 switches to its best path $(2\ 1\ 0)$ and SPVP terminates, as no other message is further generated. Therefore any fair activation sequence is forcedly finite, and SPVP cannot oscillate on this instance.

We now walk through the execution of Greedy on Di-Safe-gree. At the first iteration, vertex 3 enters the stable set V_1, and $\pi(3) = (3\ 0)$. At the second iteration, the algorithm forcedly stops. In fact, path $(2\ 1\ 0)$ is compatible with π because $2, 1 \notin V_1$, $0 \in V_1$, $\pi(0) = (0)$, and $(1, 0) \in E$. However, even if $(2\ 1\ 0)$ is the best compatible path at vertex 2, its next hop is not in V_1. A similar argument applies to path $(1\ 2\ 0)$. Therefore, no new vertex can be added to the stable set and the algorithm stops without finding a solution, since $V_1 \neq V$.

A recent contribution [12] presents an improved GREEDY algorithm that is able to solve a larger number of instances, including DI-SAFE-GREE.

Most of the sufficient conditions [14,15,16,2,10,17] to ensure safety are based on the so called dispute wheels. A *dispute wheel* [10] $\Pi^k = (\boldsymbol{\mathcal{U}}, \boldsymbol{\mathcal{Q}}, \boldsymbol{\mathcal{R}})$ of size k is a sequence of vertices $\boldsymbol{\mathcal{U}} = u_0, u_1 \dots u_{k-1}$ and sequences of nonempty paths $\boldsymbol{\mathcal{Q}} = Q_0, Q_1 \dots Q_{k-1}$ and $\boldsymbol{\mathcal{R}} = R_0, R_1 \dots R_{k-1}$ such that:

i. R_i is a path from u_i to u_{i+1}
ii. $Q_i \in \mathcal{P}^{u_i}$
iii. $R_i Q_{i+1} \in \mathcal{P}^{u_i}$
iv. $\lambda^{u_i}(Q_i) \geq \lambda^{u_i}(R_i Q_{i+1})$

We define No-Dispute-Wheel as the set of SPVP instances that do not have a dispute wheel. In [13] it is observed that testing the absence of dispute-wheels is coNP-complete.

Theorem 8. *No-Dispute-Wheel ⊂ Greedy-Solvable.*

Proof. Part 1 (⊆): If an instance S of SPVP has no dispute wheel, then GREEDY successfully solves S [2].

Part 2 (⊂): As shown in [10], instance BAD-BACKUP (Fig. 2(f)) has a dispute wheel. However, it is easy to see that it is solved by GREEDY. In fact, at iteration 1 path (4 0) is compatible with π because $(4, 0) \in E$, $0 \in V_0$, and $\pi(0) = (0)$. Since (4 0) is also highest ranked at vertex 4, we have that $V_1 = \{0, 4\}$ and π is updated setting $\pi(4) = (4\ 0)$. At iteration 2 vertex 3 enters the stable set. In fact, (3 4 2 0) is not compatible with π since $\pi(4) \neq (4\ 2\ 0)$, while (3 0) is compatible with π. Therefore, we set $V_2 = \{0, 4, 3\}$ and $\pi(3) = (3\ 0)$. At iteration 3 vertex 1 enters the stable set, because (1 3 0) is the best ranked path compatible with π. We set $V_3 = \{0, 4, 3, 1\}$ and $\pi(1) = (1\ 3\ 0)$. Last, at iteration 4 also vertex 2 can enter the stable set, since (2 1 0) is the not compatible with π, making (2 0) the best compatible path. Hence, we have that $V_4 = \{0, 4, 3, 1, 2\} = V$ and $\pi(2) = (2\ 0)$.

5 Link Costs and Commercial Relationships

In this section we discuss the interplay between the stability of SPVP instances and several real-life constraints.

5.1 Cost-Consistent Instances

In computer networks quite often links have a cost. Hence, it is natural to rank the paths according to their cost. Consider an SPVP instance $(G, \mathcal{P}, \Lambda)$ and suppose that the edges of G have a cost. Let the cost function c be such that no cycle exists with a nonpositive cost and suppose that the paths are ranked by Λ according to their cost. We say that $(G, \mathcal{P}, \Lambda)$ is *cost-consistent* with c. We define Cost-Consistent as the set of SPVP instances that are cost-consistent with at least one cost function [10].

Theorem 9. *Cost-Consistent* $\subset$ *No-Dispute-Wheel.*

Proof. Part 1 ($\subseteq$): The absence of dispute wheels in cost-consistent instances is proved in [10].

Part 2 ($\subset$): INCOHERENT (Fig. 2(g)), a simplification of an instance presented in [10], is an example of instance that has no dispute wheel and is not cost-consistent with any cost function. Assign variables to edges according to Fig. 2(g). Since path (3 2 1 0) is preferred over (3 2 0), the cost function must be such that $a+b+d < a+c$. Also, since (2 0) is preferred over (2 1 0) we have $c < b+d$, yielding a contradiction.

5.2 Modeling Commercial Relationships

From the economic perspective, relationships between ASes can be classified as customer-provider or peer-peer. In order to implement these agreements, routing policies must obey several constraints. In [11] it has been observed that an SPVP instance $(G, \mathcal{P}, \Lambda)$ modeling these constraints must be as follows.

The neighbors of each vertex of G can be partitioned into three sets: customers, providers, and peers, such that: (i) Each path of $\mathcal{P}$ is *valley-free*: provider-customer and peer-peer edges can only be followed by provider-customer edges. (ii) Function Λ is such that, for each vertex of G, paths through customers are ranked better than paths through peers that, in turn, are ranked better than paths through providers (*prefer-customer* ranking). (iii) Relationships are acyclic.

In [18] it is shown that the valley-free condition can be tested efficiently:

Theorem 10. *Given an SPVP instance it takes polynomial time to test whether the neighbors of each vertex of G can be partitioned into three sets: customers, providers, and peers, such that each permitted path is valley-free [18].*

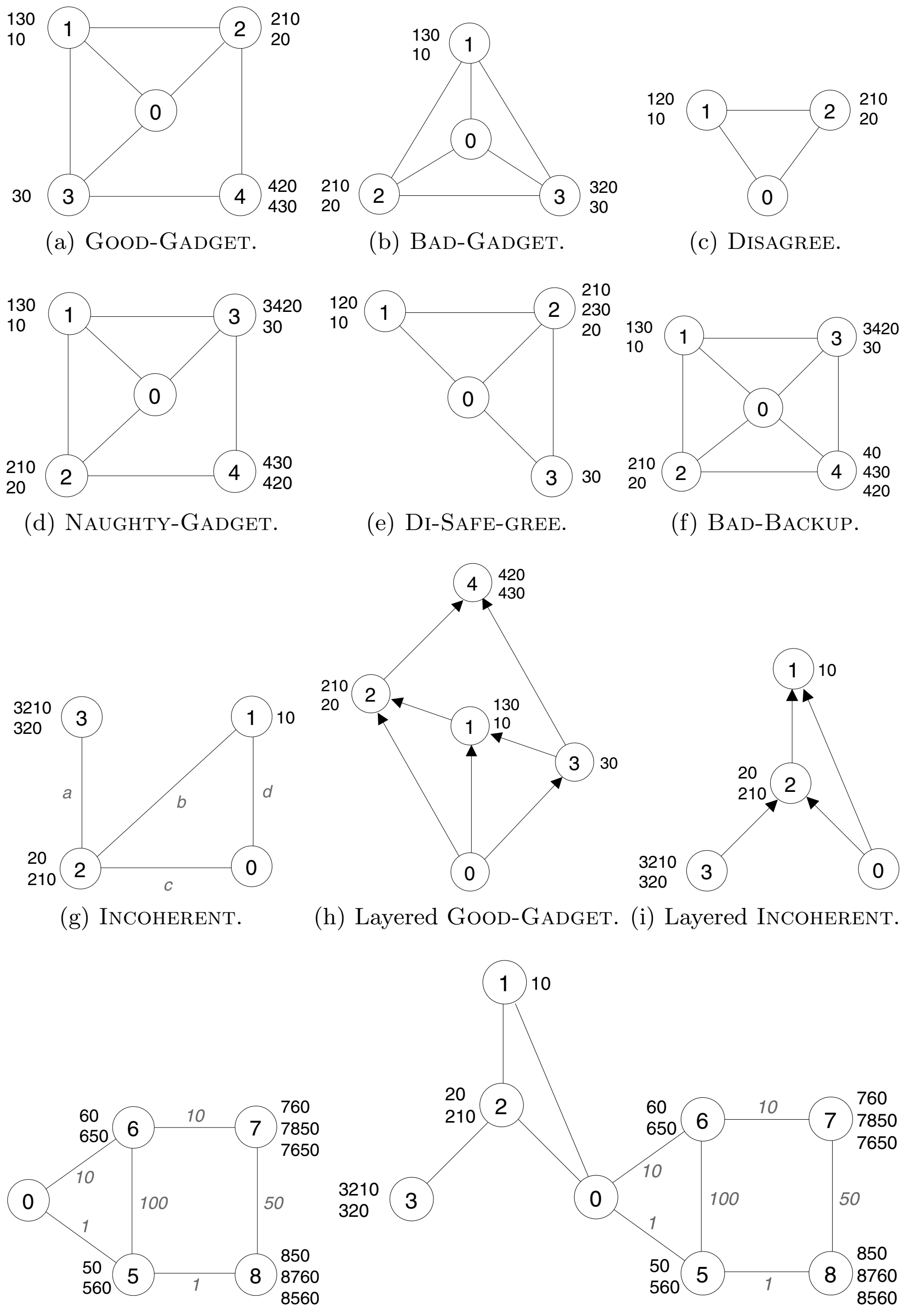
GOOD-GADGET (Fig. 2(a)) and INCOHERENT (Fig. 2(g)) are examples of instances that admit a valley-free assignment of commercial relationships such that the path rankings are prefer-customer. Two such assignments are shown in Figs. 2(h) and 2(i), where edges go from customers to providers. The layering emphasizes the customer-provider hierarchy.

Theorem 11. *Customer-Provider* $\cap$ *Cost-Consistent* $\neq$ *Cost-Consistent.*

Proof. We have already shown in the proof of Theorem 9 that INCOHERENT (Fig. 2(g)) is not cost-consistent with any cost function. On the other hand, the policies in INCOHERENT are compatible with the customer-provider hierarchy depicted in Fig. 2(i).

Theorem 12. *Customer-Provider* $\cap$ *Cost-Consistent* $\neq$ *Customer-Provider.*

Proof. It is easy to check that COSTOMER (Fig. 2(j)) is cost-consistent with the edge cost function which is presented in the figure. We now show that COSTOMER is not compatible with any customer-provider hierarchy. Consider edge $(6, 7)$. We have three possibilities:



(a) Good-Gadget. (b) Bad-Gadget. (c) Disagree.

(d) Naughty-Gadget. (e) Di-Safe-gree. (f) Bad-Backup.

(g) Incoherent. (h) Layered Good-Gadget. (i) Layered Incoherent.

(j) Costomer. (k) Nocust-Nolowest-omer.

Fig. 2. Interesting instances of SPVP

i. 6 is a provider of 7. Since $\lambda^7((7\ 6\ 5\ 0)) < \lambda^7((7\ 8\ 5\ 0)) < \lambda^7((7\ 6\ 0))$, 8 must be a provider of 7 as well. Then path (8 7 6 0) contains a valley.

ii. 6 is a peer of 7. As above, 8 is also a peer of 7, and path (8 7 6 0) contains two consecutive peer-peer edges.

iii. 6 is a customer of 7. Applying the same argument as above, we conclude that 7 is a provider of 8. By looking at the path ranking at 8 we have that $\lambda^8((8\ 7\ 6\ 0)) < \lambda^8((8\ 5\ 6\ 0))$ implies that 5 is also a provider of 8. Then path (7 8 5 0) forms a valley.

Theorem 13. *Customer-Provider* $\subset$ *No-Dispute-Wheel.*

Proof. Part 1 ($\subseteq$): Customer-provider instances cannot have a dispute wheel [19].

Part 2 ($\subset$): COSTOMER (Fig. 2(j)) has no dispute wheel, since it is cost-consistent with an edge cost function. On the other hand, from the proof of Theorem 12 we have that COSTOMER $\notin$ Customer-Provider.

Theorem 14. *Customer-Provider* $\cup$ *Cost-Consistent* $\subset$ *No-Dispute-Wheel.*

Proof. It is possible to merge COSTOMER and INCOHERENT together, obtaining NOCUST-NOLOWEST-OMER (Fig. 2(k)). Since the building blocks are completely independent and do not form a dispute wheel, then NOCUST-NOLOWEST-OMER $\in$ No-Dispute-Wheel. On the other hand, the proofs of Theorems 9 and 13 ensure that the instance is neither in Cost-Consistent nor in Customer-Provider.

6 Concluding Remarks

In this paper we have presented a survey on the state of the art knowledge about BGP stability, relating existing results by means of some new original contributions.

Many other contributions have been proposed in the literature on the BGP stability problem, but they do not fit the scope of this paper. For example, several proposals have been presented to modify BGP in such a way to improve its convergence properties [20,21,22,23,24,25,26,27,28]. Also, there are modeling contributions on I-BGP, that is the part of the BGP protocol that is run by ISPs inside the ASes [29,30]. Interestingly, the SPVP model can be applied also in this context [14,17,31]. Further, [19] studies the *robustness* of BGP defined as the guaranteed convergence under any combination of link failures.

Besides deep methodologies, the BGP stability field shows several open problems. For example, referring to the classes of Fig. 3, the following questions come naturally. Does it exist an SPVP instance that belongs to Safe but is not in Unique? Is it possible to explore the gap between the Greedy-Solvable and the Safe classes introducing further algorithms and/or conditions that better characterize safety? Is it possible to check if an SPVP instance belongs to the Customer-Provider class in polynomial time?

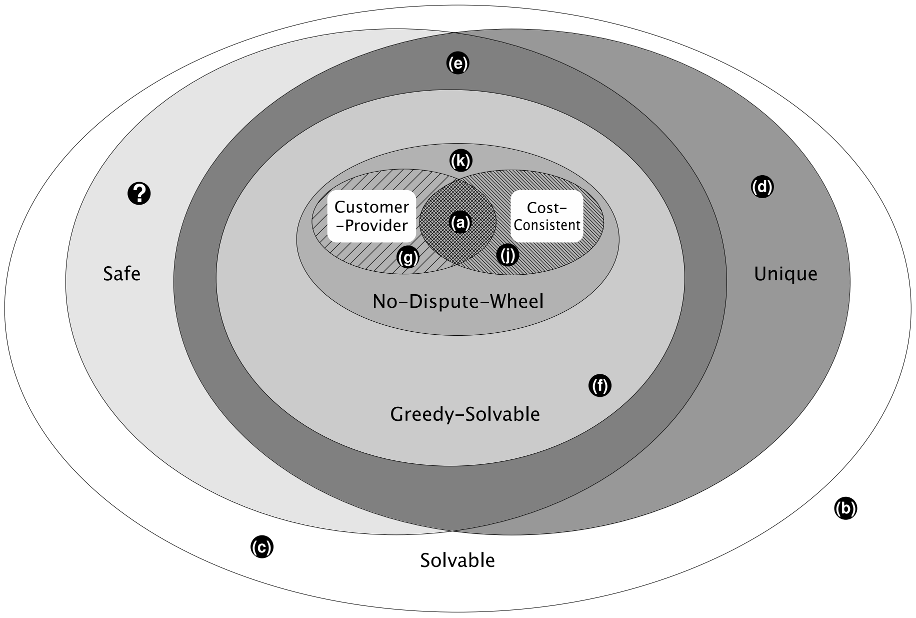


Fig. 3. Relationships between the classes of SPVP instances. Black dots with letters represent instances from Fig. 2. The black dot with a question mark indicates that an instance of that subset has not been found yet.

References

1. Rekhter, Y., Li, T., Hares, S.: A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard) (January 2006)
2. Griffin, T.G., Shepherd, F.B., Wilfong, G.: The stable paths problem and interdomain routing. IEEE/ACM Trans. on Networking 10(2), 232–243 (2002)
3. Carlzon, M.: Bgp oscillations when peering with loopback addresses. In: Proc. AINA 2006 (2006)
4. Varadhan, K., Govindan, R., Estrin, D.: Persistent route oscillations in interdomain routing. Computer Networks 32(1), 1–16 (2000)
5. Kin Chau, C.: Policy-based routing with non-strict preferences. In: Proc. SIGCOMM 2006, pp. 387–398 (2006)
6. Griffin, T.G., Sobrinho, J.L.: Metarouting. In: Proc. SIGCOMM 2005, pp. 1–12 (2005)
7. Sobrinho, J.L.: An algebraic theory of dynamic network routing. IEEE/ACM Trans. on Networking 13(5), 1160–1173 (2005)
8. Griffin, T.G., Wilfong, G.T.: A safe path vector protocol. In: Proc. INFOCOM 2000, pp. 490–499 (2000)
9. Griffin, T.G., Wilfong, G.: An analysis of bgp convergence properties. Proc. SIGCOMM 1999 29(4), 277–288 (1999)
10. Griffin, T.G., Shepherd, F.B., Wilfong, G.: Policy disputes in path-vector protocols. In: Proc. ICNP 1999, pp. 21–30 (1999)
11. Gao, L., Rexford, J.: Stable internet routing without global coordination. In: Proc. SIGMETRICS 2000, pp. 307–317 (2000)

12. Cittadini, L., Di Battista, G., Rimondini, M.: How Stable is Stable in Interdomain Routing: Efficiently Detectable Oscillation-Free Configurations. Technical Report RT-DIA-132-2008, Università Roma Tre (July 2008)
13. Fabrikant, A., Papadimitriou, C.: The complexity of game dynamics: Bgp oscillations, sink equilibria, and beyond. In: Proc. SODA, pp. 844–853 (2008)
14. Jaggard, A.D., Ramachandran, V.: Robust path-vector routing despite inconsistent route preferences. In: Proc. ICNP 2006, pp. 270–279 (2006)
15. Feamster, N., Johari, R., Balakrishnan, H.: Implications of autonomy for the expressiveness of policy routing. IEEE/ACM Trans. on Networking 15(6), 1266–1279 (2007)
16. Jaggard, A.D., Ramachandran, V.: Robustness of class-based path-vector systems. In: Proc. ICNP 2004, pp. 84–93 (October 2004)
17. Rawat, A., Shayman, M.A.: Preventing persistent oscillations and loops in ibgp configuration with route reflection. Computer Networks 50(18), 3642–3665 (2006)
18. Di Battista, G., Erlebach, T., Hall, A., Patrignani, M., Pizzonia, M., Schank, T.: Computing the types of the relationships between autonomous systems. IEEE/ACM Trans. on Networking 15(2), 267–280 (2007)
19. Gao, L., Griffin, T., Rexford, J.: Inherently safe backup routing with BGP. In: Proc. INFOCOM 2001, pp. 547–556 (2001)
20. Ahronovitz, E., Konig, J.-C., Saad, C.: A distributed method for dynamic resolution of bgp oscillations. In: Proc. IPDPS 2006 (April 2006)
21. Bremler-Barr, A., Afek, Y., Schwarz, S.: Improved bgp convergence via ghost flushing. In: Proc. INFOCOM 2003, vol. 2, pp. 927–937 (2003)
22. Cobb, J.A., Gouda, M.G., Musunuri, R.: A stabilizing solution to the stable path problem. In: Proc. Self-Stabilizing Systems, pp. 169–183 (2003)
23. Ee, C.T., Ramachandran, V., Chun, B.G., Lakshminarayanan, K., Shenker, S.: Resolving inter-domain policy disputes. Technical Report UCB/EECS-2007-27, EECS Department, University of California, Berkeley (February 2007)
24. Klockar, T., Carr-Motyčková, L.: Preventing oscillations in route reflector-based i-bgp. In: Proc. ICCCN 2004, pp. 53–58 (2004)
25. Luo, J., Xie, J., Hao, R., Li, X.: An approach to accelerate convergence for path vector protocol. In: Proc. GLOBECOM 2002, vol. 3, pp. 2390–2394 (2002)
26. Musunuri, R., Cobb, J.A.: A complete solution for ibgp stability. In: Proc. IEEE International Conference on Communications (ICC 2004), vol. 2, pp. 1177–1181 (June 2004)
27. Pei, D., Zhao, X., Wang, L., Massey, D., Mankin, A., Wu, S.F., Zhang, L.: Improving bgp convergence through consistency assertions. In: Proc. INFOCOM 2002, vol. 2, pp. 902–911 (2002)
28. Zhang, H., Arora, A., Liu, Z.: A stability-oriented approach to improving bgp convergence. In: Proc. IEEE Intl. Symposium on Reliable Distributed Systems 2004, pp. 90–99 (2004)
29. Basu, A., Ong, C.-H.L., Rasala, A., Shepherd, F.B., Wilfong, G.: Route oscillations in i-bgp with route reflection. In: Proc. SIGCOMM 2002 (2002)
30. Griffin, T.G., Wilfong, G.: On the correctness of ibgp configuration. Proc. SIGCOMM 2002 32(4), 17–29 (2002)
31. Griffin, T.G., Wilfong, G.T.: Analysis of the med oscillation problem in bgp. In: Proc. ICNP 2002, pp. 90–99 (2002)

Memory Efficient Anonymous Graph Exploration

Leszek Gąsieniec[1] and Tomasz Radzik[2]

[1] Department of Computer Science, University of Liverpool,
Liverpool L69 3BX, United Kingdom
L.A.Gasieniec@liverpool.ac.uk

[2] Department of Computer Science, King's College London, Strand,
London WC2R 2LS, United Kingdom
Tomasz.Radzik@kcl.ac.uk

Abstract. Efficient exploration of unknown or unmapped environments has become one of the fundamental problem domains in algorithm design. Its applications range from robot navigation in hazardous environments to rigorous searching, indexing and analysing digital data available on the Internet. A large number of exploration algorithms has been proposed under various assumptions about the capability of mobile (exploring) entities and various characteristics of the environment which are to be explored. This paper considers the *graph model*, where the environment is represented by a graph of connections in which discrete moves are permitted only along its edges. Designing efficient exploration algorithms in this model has been extensively studied under a diverse set of assumptions, e.g., directed vs undirected graphs, anonymous nodes vs nodes with distinct identities, deterministic vs probabilistic solutions, single vs multiple agent exploration, as well as in the context of different complexity measures including the time complexity, the memory consumption, and the use of other computational resources such as tokens and messages. In this work the emphasis is on memory efficient exploration of anonymous graphs. We discuss in more detail three approaches: *random walk*, *Propp machine* and *basic walk*, reviewing major relevant results, presenting recent developments, and commenting on directions for further research.

1 Introduction

A *graph* is a crucial combinatorial notion used for modeling complex systems in various application domains including communication, transportation and computer networks, manufacturing, scheduling, molecular biology and peer-to-peer networks. Models based on graphs often involve mobile entities which can move throughout the graph from node to node along the edges. We call such entities *agents*. An agent can be a robot servicing a hazardous environment, or a software process navigating the Internet in search for some information. *Graph exploration* refers to problems of designing algorithms (protocols) for an agent, or a group of agents, to traverse a graph in a systematic and efficient way.

In recent years the research on efficient graph exploration gathered a new momentum, generated to large extent by the theory and the applications coming ever closer together. The demand for efficient practical solutions has increased as systems of software agents moving through a large network of computers have become reality. Another example is the relevance of efficient graph exploration algorithms for efficiency of the Internet search engines. The current applications indicate that the relative importance of various aspects of graph exploration has been changing, and those changes become reflected in the theoretical research.

In the broad context of algorithmic agent design we distinguish two main models: the *geometric model* where the search environment is represented by two- or higher-dimensional space (see, e.g., [11,28,59]) and the *graph model*, considered in this paper, where the environment is represented by a finite or infinite graph supported by discrete moves permitted only along its edges. The design of efficient exploration algorithms in the graph model has been extensively studied under many different assumptions, e.g., *directed* vs *undirected graphs*, *anonymous* nodes vs nodes with *distinct identities*, or *deterministic* vs *probabilistic* solutions, as well as with different performance objectives in mind including optimal time complexity, memory consumption, or use of other resources, see [1,6,9,29,30,31,34,41,56]. Different studies may also consider different aims of exploration. The aim of exploration can be to visit each node in the network, or each edge, and terminate. Alternatively, one may drop the termination requirement and ask only for *perpetual* exploration and a guarantee that each node is visited infinitely many times, or perhaps a stronger guarantee that the nodes are being visited with similar frequencies.

If nodes have distinct identities (given as $O(\log n)$-bit words), the graph stays unchanged (a static graph), and the size of the memory available to the agent (counted in words) is linear in the number of nodes of the graph, then the depth-first search (DFS) procedure gives linear time exploration. However, in many applications one or more of these assumptions might not hold. The nodes may not have unique identities; for example, if the nodes represent very simple devices (*anonymous* graphs). The graph may keep changing; for example, if it models a growing peer-to-peer network (*dynamic* graphs). Finally, the agents may have very limited memory, which may be sufficient for storing only few node identities. For example, software agents moving through a computer network from host to host might not be allowed to carry too much data with them. This survey is mainly concerned with exploration of anonymous graphs by agents equipped with bounded memory.

In graph exploration algorithms, the memory utilisation refers actually not only to the memory of the agents ("carried" by them when they move from node to node), but also to any extra memory required in the graph environment. The latter may store some (pre-computed) additional information about the graph to guide the exploration, or may allow the agent to leave marks at nodes or to move tokens as it traverses. The demand for simple and cost effective agents as well as the desire to design exploration algorithms that are suitable for rigorous mathematical analysis imply the importance of limiting the local memory of agents and their ability to manipulate the explored environment. One of the most challenging problems in the theory of computation is to look at the far ends, *border cases*, of the considered models. In case of algorithmic agent design such a border case may refer to the size of the agent's memory, where one can limit the memory of an agent to a constant number of bits. This case is very often modeled as graph exploration by a finite state automaton and it has been extensively studied already in the 1970's [15,54,58,61]. Probably the strongest result in this setting is due to Cook and Rackoff [19]. They proved that a fixed group of finite automata, that can permanently cooperate and that can use "teleportation" to move from their current location to the location of any other automaton, cannot explore all graphs. See [42] for some recent results about limits of graph exploration with finite automata. These results imply that we either have to allow the agents to use larger memory, or to divert to randomization, or to provide the agents with extra structural information that restricts the set of graphs they have to traverse.

It has been known for some time that if we do not place strict restrictions on the local memory, then a single pebble is sufficient to explore an anonymous undirected graph. This result was extended to directed graphs by Bender *et al.* [8]. Note however that a good upper bound on the number of nodes must be known to avoid an exponential-time solution. Moreover, even if such a bound is known, the time complexity, while polynomial, remains impractically high. Another possibility is to drop determinism and to look for randomized solutions. It is known, e.g., that a random walk of length $O(n^3 \log n)$ visits all nodes of an arbitrary n-node graph with high probability [3]. Attempts to regain determinism included research on derandomization of random walks and the main approach was *universal traversal sequences* [3] that provide guidance in deterministic traversal of all graphs in a given class. Several important results have been achieved [4,7,46,60] including Reingold's [60] recent asymptotically optimal $O(\log n)$-space deterministic algorithm for the undirected *st-connectivity* problem based on a novel $O(\log n)-$bit navigation mechanism. However, note that the exploration time given by this algorithm is a polynomial of a rather high degree.

Research closely related to the setting adopted in this paper assumes some structural information about the explored environment. Such additional information allows improvements in the time or memory complexity of graph traversal. The first results, concerning exploration of a labyrinth using a compass, are due to Blum and Kozen [12]. Later, Flocchini *et al.* [39] introduced a more general notion of *sense of direction* and proved that traversal can be performed using $O(n)$ messages/agent moves in this model [38]. Fraigniaud *et al.* [40] have shown that interval routing scheme can be used to achieve the same goal. In fact, given a spanning tree, the graph can be traversed using $O(n)$ moves. Pelc and Panaite [56] studied the impact of having a map of the graph on the efficiency of graph exploration. Finally Cohen *et al.* studied efficient navigation in graphs with nodes marked with a constant number of colors, see [18].

In this paper we focus on three graph traversal methods. We start with the probabilistic *random walk* method and then discuss its recently proposed deterministic counterpart known as the *Propp machine*, which requires some (small) memory at the nodes of the graph. We conclude this survey with presentation of an alternative traversal method based on the *basic walk*, in which a small amount of memory is provided to an agent and a certain type of graph preprocessing is permitted. These three methods have several interesting combinatorial properties and one might say that they have already set certain standards in agent based anonymous graph exploration.

2 The Graph Model

In this survey we consider environments represented by undirected (symmetric) graphs. We denote by $G = (V, E)$ the graph which is to be explored, and assume that it is connected, unless stated otherwise. It will sometimes be convenient to view G as a digraph $\overleftrightarrow{G}$ obtained by replacing each undirected edge with two arcs pointing in opposite directions. We consider graphs (environments) that are anonymous, i.e., the nodes in the graphs are neither labeled nor marked in any other way. However, the ends of edges incident to each node v are ordered and often labeled by consecutive integers $1, \ldots, d_v$ called *port numbers*, where d_v is the degree of v.

If an agent is in the current step at a node v, then the standard assumptions are that it knows d_v, the label of the port through which it has entered v and any information about the graph that it might have gathered in previous steps and has been able to (has been allowed to) store in its internal limited memory. The agent decides on the basis of this knowledge which port it should take to move in the next step to a neighbour of v. Agents do not have prior knowledge about the topology of the network. The exact details about the resources and abilities of the agents as well as about the objectives of exploration may vary from problem to problem. The number of nodes and the number of edges in G are denoted by n and m, respectively, but note that these parameters might not be known to the exploring agents.

3 The Random Walk

A *random walk* on an (undirected, connected) graph G starting at a node v_0 is an (infinite) random sequence $(v_0, v_1, v_2, \ldots)$ of nodes in G such that for each $i \geq 1$, node v_i is selected randomly and uniformly from all neighbours of node v_{i-1}. Using the graph exploration terminology, we say that an agent moves in step i from node v_{i-1} to its random neighbour v_i. To implement such random exploration of an arbitrary n-node graph, the agent has to be able to select a random neighbour of the current node, so it needs $O(\log n)$-bit memory and access to $\log n$ random bits per step.

The *(node) cover time of graph G from a node v* is the expected number of steps $C_v(G)$ the random walk starting from node v takes to visit all nodes of the graph. The *cover time $C(G)$ of graph G* is defined as the maximum $C_v(C)$ over all nodes $v \in V$. Thus the cover time is the worst-case (over all starting nodes) expected time of exploring the whole graph. For graphs of some special types, the cover time can be easily estimated, or even calculated exactly. For example, it is easy to show that the cover time of the graph P_n which is an n-node simple path is equal to $C(P_n) = (n-1)^2$, by solving a simple recurrence relation for the number expected number of steps H_i required to reach the end of the path starting from its i-th node. Calculation of the cover time of the n-node clique K_n is the *coupon collector* problem: at step i select randomly and uniformly one of the n coupons/nodes and wait until all coupons have been seen, with a slight modification that the next coupon/node has to be different from the one just selected. If we have already seen exactly i distinct nodes, then the probability that in the next step we will see a new node is equal to $(n-i)/(n-1)$, so the expected number of steps before a new node is encountered is equal to $(n-1)/(n-i)$ and the cover time $C(K_n)$ is equal to $\sum_{i=1}^{n-1}(n-1)/(n-i) = (n-1)\sum_{i=1}^{n-1} 1/i = n(\ln n + O(1))$.

Random walks became an important tool in algorithm design and complexity theory when Aleliunas *et al.* [3] showed in 1979 that the cover time of *every* graph is polynomial, or more specifically, at most $2m(n-1)$. Since then general techniques for bounding the cover times have been developed and bounds for the cover times for various families of graphs have been derived. An agent using a random walk on an n-node graph with the cover time bounded by $T(n) = poly(n)$ should have an $O(\log n)$-bit counter to count $T(n)$ steps to terminate with the knowledge that the whole graph has been explored with constant probability, or to count $T(n)p \log n$ steps to terminate with the knowledge the whole graph has been explored with the high probability of at least $1 - 1/n^p$. Thus a good bound $T(n)$ on the cover time is required not to expand unnecessarily the exploration time, and we review below the main known bounds. Observe that an agent implementing a random walk needs memory for two purposes. It needs $O(\log \Delta)$ bits to implement individual moves from the current node to a random neighbour, where $\Delta \leq n$ is a bound on the degree of a node, and $O(\log n)$ bits to count the moves. If we do not require that the agent terminates, than the total of $O(\log \Delta)$ bits suffices. In particular, constant-size memory is sufficient for the randomized *perpetual* exploration of any constant degree graph.

Feige [36,35] showed the following tight bounds on the range of the cover times of n-node graphs:

$$(1 - o(1))n \ln n \leq C(G) \leq (1 + o(1))\frac{4}{27}n^3.$$

Thus K_n is an example of a graph with the cover time achieving the lower bound, while it can be shown that the n-node *lollipop* graph given in Figure 1 has the cover time achieving the upper bound. A quadratic $O(n^2)$ upper bound on the cover time of regular graphs was first shown by Kahn, Linial, Nisan and Saks in 1989, and the best know bound of $2n^2$ is due to Feige [37]. This worst-case upper bound for regular graphs should be contrasted with Rubinfeld's [62] $O(n \log n)$ bound on the cover time of regular *expander* graphs, and with Cooper and Frieze's [20] recent result showing that the cover time of a *random* d-regular graph is $(1 + o(1))\frac{d-1}{d-2}\ n \ln n$ with high probability. The cover time of the 2-dimensional $\sqrt{n} \times \sqrt{n}$ grid is $\Theta(n \log^2 n)$, with the upper bound due to Chandra *et al.* [17] and the lower bound due to Zuckerman [65]. Aldous [2] showed that the cover time of the n-node k-dimensional grid is $\Theta(n \log n)$, for $k \geq 3$.

Aleliunas' *et al.* [3] polynomial upper bound on the cover time of an arbitrary graph was an important result for the computational complexity theory as it showed that the undirected s-t connectivity problem can be solved by a randomised log-space algorithm. This started a vast body of research with the goal to settle the conjecture that

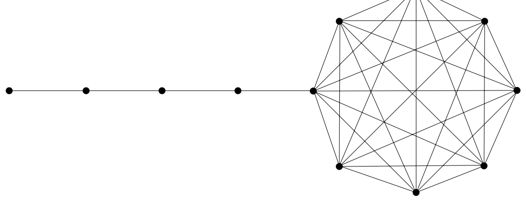


Fig. 1. The n-node lollipop graph: $(n/3)$-node path connected to n-node clique

this problem can be solved by a *deterministic* log-space algorithm, and, ultimately, the more general conjecture that *any* problem solvable by a randomised log-space algorithm can be solved by a deterministic log-space algorithm. A natural approach to settle the first conjecture was to de-randomise random walks. In this framework the objective was to produce an explicit universal traversal sequence (UTS), i.e., a sequence $p_1, \ldots, p_k$ of port labels, such that the path guided by this sequence visits all edges of any graph of a given size. It is known that, with high probability, a sequence of length $O(n^3 d^2 \log n)$, chosen uniformly at random, guides a walk in any d-regular (connected) n-node graph [3]. Unfortunately, explicit short UTS are known only for special families of graphs, including 2-regular graphs [7,13,16,50,53], 3-regular graphs [49], cliques [51], and expanders [46]. Some of these sequences can be constructed in log-space, providing exploration with $O(\log n)$-bit memory. Koucký [52] introduced the notion of a universal *exploration* sequence (UXS), i.e., a sequence $q_1, \ldots, q_k$ such that the agent leaves the current node x via port $p + q_i$ at the ith step, where p is the label of the port through which the agent entered node x. This notion allows to construct simpler and shorter sequences, for example, 1^n is a UXS for n-node cycles, and $(10)^n$ is a UXS for n-node cliques. Reingold [60] has recently showed that a UXS for general graphs is log-space constructible, providing a deterministic log-space algorithm for undirected s-t connectivity and settling the first of the above two conjectures. Note that both UTS and UXS require *a priori* knowledge of the size of the network. If an agent uses an UTS or UXS for exploration, with stopping, of graphs (of some type) of size at most n, then it does not know at the termination whether the graph has at most n nodes, or whether it is larger and possibly not fully explored.

A random walk on a graph is an example of a *finite-state Markov chain* and has often been studied within this general context, with its central issue of the *stationary distribution* and the rate of convergence to this distribution. If a random walk starts at a node s ($v_0 = s$), then the i-th node v_i on the walk is the random variable with distribution π_s^i, where $\pi_s^i(v) = \text{Prob}(v_i = v)$. The stationary distribution π is the probability distribution on the set of nodes defined by $\pi(v) = \lim_{i \to \infty} \pi_s^i(v)$, if these limits exist and are independent of the starting node s. The stationary distribution exists for every connected non-bipartite graph, and is equal to $\pi(v) = \deg(v)/(2m)$. (For a bipartite graph, since there is no odd-length cycle, a random walk can visit a given node v either only in even steps, or only in odd steps, so $\lim_{i \to \infty} \pi_s^i(v)$ is not defined. Not to exclude bipartite graphs, the inconvenience of not having an odd length cycle is usually dealt with by allowing the walk to stay at the current node with, say, probability $1/2$, or, equivalently, by adding self-loops to the graph.) The rate of convergence to the stationary distribution is usually measured by the *mixing time*, defined as the first step t when the distribution π_s^t of the random variable v_t is guaranteed to be close to the stationary distribution π. More precisely, the mixing time is the minimum t such that $\max_{s,v \in V}\{|\pi_s^t(v) - \pi(v)|\} \leq 1/n^3$, though other definitions of the distance between two distributions, and degrees of closeness other than $1/n^3$ have also been used. The random walk is *rapidly mixing*, if the mixing time is short, say $O(n^\epsilon)$ for a small constant ϵ. Among the constant degree graphs, expanders have the best possible $O(\log n)$ mixing time.

The advantage of a random walk as a strategy for exploring a graph is its simplicity and low memory requirements. Its main drawback is the time required to complete the exploration, which is given by the cover time and can be as high as $\Omega(n^3)$. Thus a natural question is to reduce the cover time, if possible, for example by considering nonuniform transition probabilities, or by allowing the walking agent to gather, and use, some limited information about the graph. Ikeda *et al.* [47] considered the nonuniform probabilities $p(v, u)$ of moving from a node v to its neighbour u such that $p(v, u')/p(v, u'') = (\deg(u')/\deg(u''))^{1/2}$, for any two neighbours u' and u'' of node v. They showed that these transition probabilities lead to an $O(n^2 \log n)$ bound on the cover time of any graph. This quite remarkable reduction from the $O(n^3)$ bound of the uniform random walk, comes unfortunately with a cost. To implement this non-standard random walk, each node of the graph has to store information about the degrees of its neighbours, or the agent has to visit first all neighbours to gather this information. Ikeda *et al.* [47] showed also that the $O(n^2 \log n)$ bound is close to the best what we can hope for, since for any transition probabilities defined on an n-node path, the cover time is $\Omega(n^2)$.

If we are looking for graph exploration with $k \geq 2$ agents, then we would like to know good bounds on the cover time by k random walks: the expected number of steps until each node has been visited by at least one random walk (assuming all agents move simultaneously in synchronised steps). Observe that it may be, and indeed is, crucial what are the relative positions of the starting nodes of the walks. Broder *et al.* [14] considered k independent random walks starting from the stationary distribution (the starting node of walk i is a node v with probability $\pi(v)$) and showed an $O((m^2 \log^3 n)/k^2)$ bound on the cover time. Thus, for example, for constant degree graphs, the *speed-up* (the ratio of the cover times of a single random walk and k random walks) can be $\Omega(k^2/\log^3 n)$. Recently Alon *et al.* [5] showed bounds on the speed-up of k independent random walks starting from the *same* node, including a $\Theta(\log k)$ speed-up for n-node cycles, if $\log k = O(n)$, and an $\Omega(k)$ speed-up for n-node expanders, if $k \leq n$. Cooper *et al.* [22] considered k independent random walks on random regular graphs, and analysed the cover time and the time required to achieve certain interaction between the agents.

Random walks have been also applied in the context of exploring *dynamic graphs*. Cooper and Frieze [21] considered a random walk on a dynamic graph growing according to some random process, and analysed the expected proportion of visited vertices. Law and Siu [55] and Cooper *et al.* [23] used graph exploration with random walks to create new, random edges in a process of building and maintaining a well connected network.

4 The Propp Machine

We consider now a type of graph exploration where the agents have no operational memory and the whole steering mechanism is provided within the environment. Thus the agents may actually be viewed as mere tokens which are being moved around the graph. We discuss the deterministic mechanism of the *rotor-router model*, which was introduced by Priezzhev in [57], further popularised by James Propp, and now known also as the *Propp machine*. In this model each node of the graph G is equipped with a small marker indicating the exit port (the edge) to be taken by an agent on the conclusion of the next visit to this node. After the agent leaves the node, the marker is moved immediately to the next port in the cyclic order. The rotor-router model has been introduced as a deterministic alternative to the random walk method. Its advantages include the balanced usage of exit ports at each node, replacing the "coupon collector" nature of the usage of ports by the random walk method.

The research on this model splits naturally into two directions associated with *finite* and *infinite* graphs. The main question for finite graphs is about properties of the periodic tour that has to be eventually adopted by the agent. For infinite graphs the Propp machine model was mainly investigated in the context of balancing schemes for even distribution of workload in networks.

Finite Graphs. Note that if an agent follows the rotor-router mechanism in the Propp machine defined on a finite graph, the agent must eventually lock itself in a tour of limited size. This is a straightforward consequence of the fact that the number of configurations based on positions of markers on exit ports and location of the agent is bounded by $n(d_{max})^n$, where d_{max} is the maximum degree of a node. However, and rather surprisingly, following the rotor-router mechanism leads to a periodic tour that corresponds to an *Euler tour* defined on $\overleftrightarrow{G}$ [10]. Moreover this periodic phenomenon starts occurring very early, namely within $O(|E|\cdot n)$ steps, independently of the original configuration of the port numbers and markers as well as the agent's location. Yanovski *et al.* [64] improved this bound by showing that in fact $2|E|\cdot D$ steps suffice to form an Euler tour, where D is the diameter of G. On the other hand a lower bound $\Omega(|E|\cdot D)$ can be obtained on a lollipop graph (of a general structure as in Figure 1) in which exit ports and markers in the clique with $\Omega(|E|)$ edges are set to form an Euler tour, and markers on the external path of length D are placed on ports leading towards the clique.

Yanovski *et al.* [64] studied also behaviour of a multi-agent system of explorers in the Propp machine where the agents cooperate via shared markers. When l agents want to exit from the same node in the same step, then they all leave this node in this step through the next l consecutive ports (according to an arbitrary assignment of the agents to these l ports, and sending multiple agents through the same ports, if l is greater than the degree of the node). They proved that for a team of k agents, the numbers of edge visits in the networks are balanced up to a factor of two within at most $2(1+\frac{1}{k})|E|D$ steps.

Infinite Graphs. In the context of infinite graphs the rotor-router has been mostly studied as a deterministic analogue of the random walk approach to balancing the workload in a network, and the main question has been how similar these two processes are. The agents are now tokens, which are initially distributed among the nodes in some, possibly uneven, way (for example, they all may be initially piled up on one node). The random walk approach balances the load of tokens among nodes by sending them along independent random walks of the same length t. Let $E_t(v)$ denote the expected number of tokens which end up at a node v. The rotor-router process starting with the same initial distribution of tokens and executing t (deterministic) steps reaches a configuration with $a_t(v)$ tokens at node v. The question of similarity of these two processes is the question of analysing the node discrepancies $|a_t(v) - E_t(v)|$. In particular, Cooper and Spencer [24] studied this question for d-dimensional grids and showed that all node discrepancies are bounded by a constant, which depends on d, but does not depend on the initial configuration, the total time t and the initial rotor settings. For example, they showed a bounding constant for $d = 1$ which is ≈ 2.3. This work was followed by a detail study of the 2-dimensional grid by Doerr and Friedrich [33]. They provided evidence that the exact (constant) value of the maximum node discrepancy depends on the choice of rotor sequences. The situation is different in the case of k-ary trees: for any number D, there exists an initial configuration of tokens, a number t and a node v such that after t steps of the rotor-router process node v has at least D tokens more than it is expected to get in the random walk process [25].

5 The Basic Walk

The basic walk method is based on an observation that one can cover a graph $G = (V, E)$, or more precisely its symmetric digraph counterpart $\overleftrightarrow{G}$, by a collection of directed cycles. The cycles are formed according to a simple rule. At any node v with the degree d_v, the incoming arc incident via a port i becomes the predecessor of the outgoing arc incident via port $(i \mod d_v) + 1$. Note that since each arc in the digraph has a unique predecessor as well as a unique successor a collection of arc-disjoint cycles containing all arcs in the digraph is formed. Figure 2 shows an example.

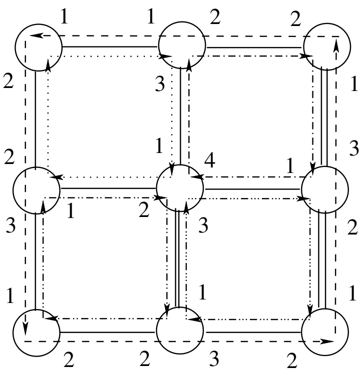


Fig. 2. The formation of cycles

Note also that a certain arrangement of the port numbers may lead to a cycle that visits all nodes in the graph (symmetric digraph), see Figure 3. We call such a cycle a *witness cycle*, see [32]. The presence of a witness cycle is very convenient in the context of graph exploration, since it can be used by a very simple mobile entity to periodically visit all nodes in the graph.

We consider two types of graph traversal based on the basic walk method. In the *oblivious model* an agent has no operational memory. It is equipped with a simplest possible mechanism that allows it to follow a single cycle in the collection of cycles

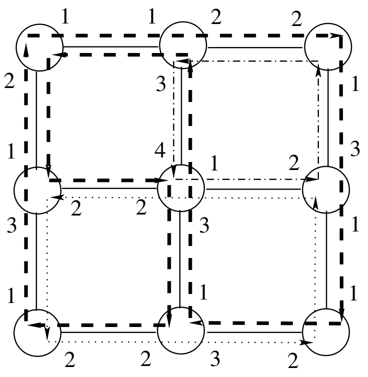


Fig. 3. A witness cycle

covering the input graph. In this model the main task is to provide port labeling to the input graph such that a witness cycle is formed. Moreover one is interested in forming a shortest possible witness cycle. Note that in this model an agent traverses indefinitely along the chosen cycle since its has no memory to make any decisions and no state to stop. In the *adaptive model* an agent is provided with small (constant number of bits) memory that allows the agent to switch between different cycles in order to shorten the route covering all nodes in the graph. Such an agent is often modeled as Mealy automaton, where the output (outgoing port number) depends on the input (incoming port number) and the state of the automaton. More formally the automaton has a transition function f and a finite number of states governing the actions of the agent. When the agent enters a node v of degree d_v through port i, it switches to state s' and exits the node through port i', where $(s', i') = f(s, i, d_v)$, see [44] for more detail description.

Oblivious Traversal. The oblivious traversal based on the basic walk method was first proposed by Dobrev *et al.* in [32]. The authors claimed that there exists a port labeling, such that the oblivious agent requires at most $10n$ steps to visit all n nodes of a graph in a periodic manner. While the stated problem and several combinatorial observations, in particular merging and exchanging contents of cycles, attracted attention in the community, the bound of $10n$ on their port labeling turned out to be incorrect. Very recently Czyzowicz *et al.* in [27] proposed a polished version of the previous argument supported by a new combinatorial structure of a *three-layer partition* of graphs. This led to the first provably correct port labeling inducing a linear tour of length $4\frac{1}{3}n$. Moreover, the labeling based on the three-layer partition can be performed in the optimal $O(|E|)-$time. The authors proposed also a non-trivial class of graphs in which one can select a spanning tree such that each node is incident to some edge outside of the spanning tree. For this class of graphs one can construct a labeling that forms a witness cycle of length $\leq 2n - 2$. An example of formation of the witness cycle in this class of graphs is presented in Figure 4.

Unfortunately the problem of deciding whether the input graph has a spanning tree with the required property is *NP-hard* since this problem corresponds to selection of a

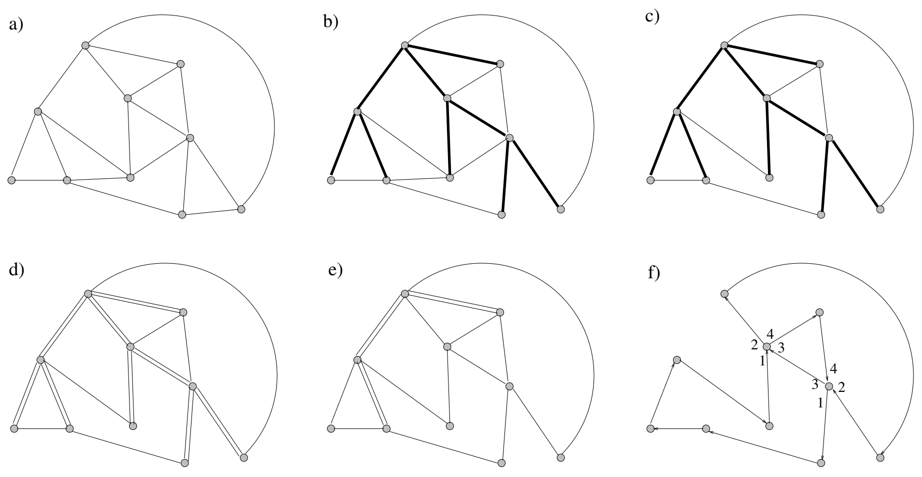


Fig. 4. Formation of a witness cycle: a) an input graph, b) a spanning tree with external edges at each node is formed, c) excessive external edges are dropped, d) the spanning tree edges are doubled, e) the parity of nodes is repaired starting from leaves in the tree, f) remaining double (non-bridging) edges are dropped and a witness cycle is formed

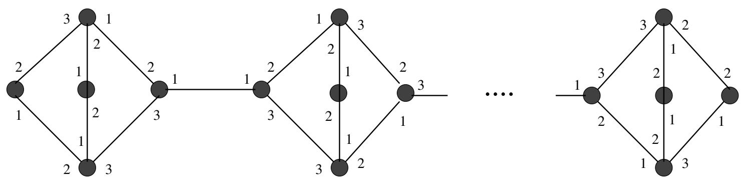


Fig. 5. A feasible solution in which each edge must be used in two directions

Hamiltonian path in 3-regular graphs, which is known to be hard. The authors in [27] give also an n-node graph, shown in Figure 5, in which the witness cycle must contain all arcs in $\overleftrightarrow{G}$ and therefore cannot be shorter than $2.8n$.

It is worth mentioning that the explicit labeling of ports in G is not essential in the oblivious graph traversal. In fact, it is enough to provide a periodic ordering of ports at each node in G. The agent when arrives at some node v via some port must know only its successor in the periodic order to continue the walk along the chosen cycle.

Adaptive Traversal. In [48] David Ilcinkas noted that the tour used by an agent to visit all nodes in the graph can be shortened to $4n - 2$ if the agent is provided with a 2-bit operational memory. The extra memory translated to a larger number of states allows the agent to perform context sensitive decisions including switching between the cycles available in the basic walk method. More precisely, Ilcinkas noticed that using a certain type of port labeling cycles and the extra memory bits, one can force the agent to follow an Euler tour defined on a chosen spanning tree T in G. The spanning tree contains a unique *root edge* $e_r = (v_r, w_r)$ that bears port number 1 at its both ends. Apart from v_r

and w_r every node u in G has a parent which is reachable from u via port 1. In fact one can also interpret that v_r is a parent of w_r and vice versa. All children of any node v in G (including root nodes) are reachable from v via ports 2 through $c_v + 1$, where c_v is the number of children of v in the spanning tree T. When the agent traverses along the Euler tour down in the tree it follows consecutive arcs of some cycle provided by the basic walk method. This process is terminated when for two consecutive nodes on the cycle v_1 and v_2, the node v_2 is entered via port different from 1, meaning that v_2 is not a child of v_1 in T. In this case the agent returns first to v_1 and then to the parent v of v_1, concluding that there are no more children of v_1 to be visited in T. The traversal is then continued along some other cycle starting at the next child (if any) of v. The edge (v_1, v_2) is called a *penalty edge* since it does not belong to T and it contributes an extra two agent moves in the tour that visits all nodes in the graph. The total length of the tour can be bounded by $2n - 2$ moves along edges in the spanning tree (each edge has to be visited in two directions) and $2n$ moves along the penalty edges (each node including root nodes may have incident edges not forming a part of the spanning tree T). Thus the total length of the tour is bounded by $4n - 2$. Ilcinkas claimed also that $4n - 2$ is the exact bound for all agents equipped with a small (constant size) memory.

This claim was later disproved by Gąsieniec *et al.* in [44] where they showed that the length of the tour could be shortened to $3.75n - 2$. The improvement was possible due to the observation that visits to penalty edges could be avoided at the fraction $\frac{1}{8}$ of the nodes. They proved that one can construct port labeling in which either a large fraction of nodes is *saturated* (all incident edges to these nodes are present in the spanning tree) or there are large clusters of sibling leaves and extended leaves (paths of length 2) located at the bottom of the spanning tree. Within each cluster the penalty edges are visited only at the first sibling, while visiting all other siblings in the same cluster is penalty free. This result was recently further improved in [27] to $3.5n - 2$ with help of the *three-layer partition* and *sham edges* that pretend to be penalty edges while serving as proper edges in the spanning tree.

Note here that while the explicit labeling of ports in G is not essential in the adaptive graph traversal at least one port at each node has to be distinguished in order to form the hierarchical structure of the spanning tree T. More precisely the marked ports play the same role as ports 1 in the explicit labeling setting.

6 Conclusion

The standard random walks are memoryless in the sense that the selection of the next node does not depend on the past. This assumption is important from the point of view of the probabilistic methods used in the theoretical analysis. Simulations show that the performance of random walks may be improved for some types of graphs, if the agents are allowed to remember, and use, some very limited information about the past. It would be very interesting to develop some theoretical analysis of such processes.

Further work on the random walk approach should include further efforts on derandomisation with limited random access memory, ideally considering also the secondary objective of minimizing the time complexity. An interesting aspects of random walks can be considered in the context of efficiency of pseudo-random number generators. A pseudo-random number generator can be seen as a deterministic steering mechanism of an agent and the efficiency of such a generator may be expressed as the efficiency of exploration in arbitrary graphs.

Research on the rotor-router mechanism started only very recently and further results on comparing this approach with random walks should be coming in near future. One possible interesting question is whether the results for infinite graphs can be used to obtain implications for their finite counterparts. Also better understanding of Euler tours formed by the rotor-router mechanism by single and multiple robots would be highly appreciated. Another direction for studies of this model is graph exploration by agents granted dynamic port labeling mechanism. This would refer to creation of various geometrical shapes and surfaces.

Finally, in the context of the basic walk approach, further understanding of witness cycles as well as tours used by agents equipped with a small memory is required. There is also very little known so far about the case when the ports at each node form a random permutation.

Acknowledgements

Many thanks go to Colin Cooper, Jurek Czyzowicz, Stefan Dobrev, David Ilcinkas, Jesper Jansson, Ralf Klasing, Adrian Kosowski, Ioannis Lignos, Russell Martin, and Alfredo Navarra for a number of valuable discussions on the main themes of this survey.

References

1. Albers, S., Henzinger, M.R.: Exploring unknown environments. SIAM Journal on Computing 29, 1164–1188 (2000)
2. Aldous, D.J.: On the time taken by random walks on finite groups to visit every state. Journal Probability Theory and Related Fields 62(3), 361–374 (1983)
3. Aleliunas, R., Karp, R.M., Lipton, R.J., Lovasz, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. In: FOCS 1979, pp. 218–223 (1979)
4. Alon, N., Azar, Y., Ravid, Y.: Universal sequences for complete graphs. Discrete Appl. Math. 27(1-2), 25–28 (1990)
5. Alon, N., Avin, C., Koucky, M., Kozma, G., Lotker, Z., Tuttle, M.R.: Many random walks are faster than one. In: SPAA 2008: Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures, pp. 119–128. ACM, New York (2008)
6. Awerbuch, B., Betke, M., Rivest, R.L., Singh, M.: Piecemeal graph exploration by a mobile robot. Information and Computation 152(2), 155–172 (1999)
7. Bar-Noy, A., Borodin, A., Karchmer, M., Linial, N., Werman, M.: Bounds on universal sequences. SIAM J. Comput. 18, 268–277 (1989)
8. Bender, M., Fernandez, A., Ron, D., Sahai, A., Vadhan, S.: The power of a pebble: Exploring and mapping directed graphs. Information and Computation 176(1), 1–21 (2002)
9. Bender, M., Slonim, D.K.: The power of team exploration: two robots can learn unlabeled directed graphs. In: Proc. of FOCS 1994, pp. 75–85 (1994)
10. Bhatt, S., Even, S., Greenberg, D., Tayar, R.: Traversing Directed Eulerian Mazes. Journal of Graph Algorithms and Applications 6(2), 157–173 (2002)

11. Blum, A., Raghavan, P., Schieber, B.: Navigating in unfamiliar geometric terrain. SIAM Journal on Computing 26, 110–137 (1997)
12. Blum, M., Kozen, D.: On the power of the compass (or, why mazes are easier to search than graphs). In: Proc. of FOCS 1978, pp. 132–142 (1978)
13. Bridgland, M.F.: Universal traversal sequences for paths and cycles. J. Algorithms 8(5), 395–404 (1987)
14. Broder, A.Z., Karlin, A.R., Raghavan, P., Upfal, E.: Trading space for time in undirected s-t connectivity. SIAM J. Comput. 23(2), 324–334 (1994)
15. Budach, L.: Automata and labyrinths. Math. Nachrichten, 195–282 (1978)
16. Buss, J.F., Tompa, M.: Lower bounds on universal traversal sequences based on chains of length five. Inf. Comput. 120(2), 326–329 (1995)
17. Chandra, A.K., Raghavan, P., Ruzzo, W.L., Smolensky, R.: The electrical resistance of a graph captures its commute and cover times. In: Proc. STOC 1989: Proceedings of the twenty-first annual ACM symposium on Theory of computing, pp. 574–586. ACM Press, New York (1989)
18. Cohen, R., Fraigniaud, P., Ilcinkas, D., Korman, A., Peleg, D.: Label-guided graph exploration by a finite automaton. In: Proc. of the 32nd International Colloquium on Automata, Languages and Programming (ICALP), pp. 335–346 (2005)
19. Cook, S.A., Rackoff, C.: Space lower bounds for maze threadability on restricted machines. SIAM Journal on Computing 9(3), 636–652 (1980)
20. Cooper, C., Frieze, A.: The cover time of random regular graphs. SIAM J. Discret. Math. 18(4), 728–740 (2005)
21. Cooper, C., Frieze, A.M.: Crawling on web graphs. In: Proc. STOC 2002, pp. 419–427 (2002)
22. Cooper, C., Frieze, A.M., Radzik, T.: Multiple random walks in random regular graphs (unpublished manuscript, 2008)
23. Cooper, C., Klasing, R., Radzik, T.: A randomized algorithm for the joining protocol in dynamic distributed networks. Technical Report RR-1432-07, LaBRI, Bordeaux, France (June 2007)
24. Cooper, J.N., Spencer, J.: Simulating a random walk with constant error. Combinatorics, Probability and Computing 15, 815–822 (2006)
25. Cooper, J., Doerr, B., Friedrich, T., Spencer, J.H.: Deterministic random walks on regular trees. In: Proc. of SODA 2008, pp. 766–772 (2008)
26. Cooper, J., Doerr, B., Spencer, J.H., Tardos, G.: Deterministic random walks on the integers. European Journal of Combinatorics 28(8), 2072–2090 (2007)
27. Czyzowicz, J., Dobrev, S., Gąsieniec, L., Ilcinkas, D., Jansson, J., Klasing, R., Lignos, I., Martin, R., Sadakane, K., Sung, W.-K.: More efficient periodic traversal in anonymous undirected graphs (unpublished manuscript, 2008)
28. Deng, X., Kameda, T., Papadimitriou, C.H.: How to learn an unknown environment: The rectilinear case. Journal of the ACM 45, 215–245 (1998)
29. Deng, X., Papadimitriou, C.H.: Exploring an unknown graph. Journal of Graph Theory 32(3), 265–297 (1999)
30. Dessmark, A., Pelc, A.: Optimal graph exploration without good maps. Theoretical Computer Science 326(1-3), 343–362 (2004)
31. Diks, K., Fraigniaud, P., Kranakis, E., Pelc, A.: Tree exploration with little memory. Journal of Algorithms 51, 38–63 (2004)
32. Dobrev, S., Jansson, J., Sadakane, K., Sung, W.-K.: Finding Short Right-Hand-on-the-Wall Walks in Graphs. In: Pelc, A., Raynal, M. (eds.) SIROCCO 2005. LNCS, vol. 3499, pp. 127–139. Springer, Heidelberg (2005)
33. Doerr, B., Friedrich, T.: Deterministic Random Walks on the Two-Dimensional Grid. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 474–483. Springer, Heidelberg (2006)

34. Duncan, C.A., Kobourov, S.G., Kumar, V.S.A.: Optimal constrained graph exploration. ACM Transaction on Algorithms 2(3), 380–402 (2006)
35. Feige, U.: A tight upper bound on the cover time for random walks on graphs. Random Structures and Algorithms 6(1), 51–54 (1995)
36. Feige, U.: A tight lower bound on the cover time for random walks on graphs. Random Struct. Algorithms 6(4), 433–438 (1995)
37. Feige, U.: Collecting coupons on trees, and the cover time of random walks. Computational Complexity 6(4), 341–356 (1996)
38. Flocchini, P., Mans, B., Santoro, N.: On the impact of sense of direction on message complexity. Information Processing Letters 63(1), 23–31 (1997)
39. Flocchini, P., Mans, B., Santoro, N.: Sense of direction: definition, properties and classes. Networks 32(3), 165–180 (1998)
40. Fraigniaud, P., Gavoille, C., Mans, B.: Interval routing schemes allow broadcasting with linear message-complexity. Distributed Computing 14(4), 217–229 (2001)
41. Fraigniaud, P., Ilcinkas, D.: Digraph exploration with little memory. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 246–257. Springer, Heidelberg (2004)
42. Fraigniaud, P., Ilcinkas, D., Peer, G., Pelc, A., Peleg, D.: Graph exploration by a finite automaton. Theoretical Computer Science 345(2-3), 331–344 (2005)
43. Garey, M.R., Johnson, D.S., Tarjan, R.E.: The planar hamiltonian circuit problem is np-complete. SIAM J. Comput. 5(4), 704–714 (1976)
44. Gąsieniec, L., Klasing, R., Martin, R., Navarra, A., Zhang, X.: Fast Periodic Graph Exploration with Constant Memory. J. Comput. Syst. Sci. 74(5), 808–822 (2007)
45. Gąsieniec, L., Pelc, A., Radzik, T., Zhang, X.: Tree exploration with logarithmic memory. In: SODA, pp. 585–594 (2007)
46. Hoory, S., Wigderson, A.: Universal traversal sequences for expander graphs. Inf. Process. Lett. 46(2), 67–69 (1993)
47. Ikeda, S., Kubo, I., Okumoto, N., Yamashita, M.: Impact of local topological information on random walks on finite graphs. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 1054–1067. Springer, Heidelberg (2003)
48. Ilcinkas, D.: Setting Port Numbers for Fast Graph Exploration. In: Flocchini, P., Gąsieniec, L. (eds.) SIROCCO 2006. LNCS, vol. 4056, pp. 59–69. Springer, Heidelberg (2006)
49. Impagliazzo, R., Nisan, N., Wigderson, A.: Pseudorandomness for network algorithms. In: Proc. STOC 1994: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing, pp. 356–364. ACM Press, New York (1994)
50. Istrail, S.: Polynomial universal traversing sequences for cycles are constructible. In: Proc. STOC 1988: Proceedings of the twentieth annual ACM symposium on Theory of computing, pp. 491–503. ACM Press, New York (1988)
51. Karloff, H.J., Paturi, R., Simon, J.: Universal traversal sequences of length $n^{O(\log n)}$ for cliques. Inf. Process. Lett. 28(5), 241–243 (1988)
52. Koucký, M.: Universal traversal sequences with backtracking. J. Comput. Syst. Sci. 65(4), 717–726 (2002)
53. Koucký, M.: Log-space constructible universal traversal sequences for cycles of length o(n4.03). Theor. Comput. Sci. 296(1), 117–144 (2003)
54. Kozen, D.: Automata and planar graphs. In: Proc. of Fundations Computatial Theory (FCT 1979), pp. 243–254 (1979)
55. Law, C., Siu, K.-Y.: Distributed construction of random expander graphs. In: Proc. 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (April 2003)
56. Panaite, P., Pelc, A.: Impact of topographic information on graph exploration efficiency. Networks 36, 96–103 (2000)
57. Priezzhev, V.B., Dhar, D., Dhar, A., Krishnamurthy, S.: Eulerian walkers as a model of self-organized criticality. Physics Review Letters 77, 5079–5082 (1996)

58. Rabin, M.O.: Maze threading automata. Technical Report Seminar Talk, University of California at Berkeley (October 1967)
59. Rao, N., Kareti, S., Shi, W., Iyengar, S.: Robot navigation in unknown terrains: Introductory survey of length, non-heuristic algorithms. Technical Report ORNL/TM12410, Oak Ridge National Lab (1993)
60. Reingold, O.: Undirected st-connectivity in log-space. In: Proc. STOC 2005, pp. 376–385 (2005)
61. Rollik, H.A.: Automaten in planaren graphen. Acta Informatica 13, 287–298 (1980)
62. Rubinfeld, R.: The cover time of a regular expander is $O(nlogn)$. Information Processing Letters 35, 49–51 (1990)
63. Winkler, P., Zuckerman, D.: Multiple cover time. Random Structures and Algorithms 9, 403–411 (1996)
64. Yanovski, V., Wagner, I.A., Bruckstein, A.M.: A Distributed Ant Algorithm for Efficiently Patrolling a Network. Algorithmica 37, 165–186 (2003)
65. Zuckerman, D.: A technique for lower bounding the cover time. SIAM J. Discret. Math. 5(1), 81–87 (1992)

Algorithmic Meta Theorems

Martin Grohe

Institut für Informatik, Humboldt-Univerität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
grohe@informatik.hu-berlin.de

Abstract. Algorithmic meta theorems are algorithmic results that apply to whole families of combinatorial problems, instead of just specific problems. These families are usually defined in terms of logic and graph theory. An archetypal algorithmic meta theorem is Courcelle's Theorem [1], which states that all graph properties definable in monadic second-order logic can be decided in linear time on graphs of bounded tree width. More recent examples of such meta theorems state that all first-order definable properties of planar graphs can be decided in linear time [2] and that all first-order definable optimisation problems on classes of graphs with excluded minors can be approximated in polynomial time to any given approximation ratio [3].

In my talk, I gave an overview of algorithmic meta theorems and the main techniques used in their proofs. Reference [4] is a comprehensive survey of the material.

References

1. Courcelle, B.: Graph rewriting: An algebraic and logic approach. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B, pp. 194–242. Elsevier Science Publishers, Amsterdam (1990)
2. Frick, M., Grohe, M.: Deciding first-order properties of locally tree-decomposable structures. Journal of the ACM 48, 1184–1206 (2001)
3. Dawar, A., Grohe, M., Kreutzer, S., Schweikardt, N.: Approximation schemes for first-order definable optimisation problems. In: Proceedings of the 21st IEEE Symposium on Logic in Computer Science, pp. 411–420 (2006)
4. Grohe, M.: Logic, graphs, and algorithms. In: Flum, J., Grädel, E., Wilke, T. (eds.) Logic and Automata – History and Perspectives. Texts in Logic and Games, vol. 2, pp. 357–422. Amsterdam University Press (2007)

A Most General Edge Elimination Polynomial[⋆]

Ilia Averbouch[⋆⋆], Benny Godlin[⋆⋆], and Johann A. Makowsky

Faculty of Computer Science
Israel Institute of Technology
Haifa, Israel
{ailia,bgodlin,janos}@cs.technion.ac.il

Abstract. We look for graph polynomials which satisfy recurrence relations on three kinds of edge elimination: edge deletion, edge contraction and edge extraction, i.e., deletion of edges together with their end points. Like in the case of deletion and contraction only (J.G. Oxley and D.J.A. Welsh 1979), it turns out that there is a most general polynomial satisfying such recurrence relations, which we call $\xi(G, x, y, z)$. We show that the new polynomial simultaneously generalizes the Tutte polynomial, the matching polynomial, and the recent generalization of the chromatic polynomial proposed by K.Dohmen, A.Pönitz and P.Tittman (2003), including also the independent set polynomial of I. Gutman and F. Harary, (1983) and the vertex-cover polynomial of F.M. Dong, M.D. Hendy, K.T. Teo and C.H.C. Little (2002). We give three definitions of the new polynomial: first, the most general recursive definition, second, an explicit one, using a set expansion formula, and finally, a partition function, using counting of weighted graph homomorphisms. We prove the equivalence of the three definitions. Finally, we discuss the complexity of computing $\xi(G, x, y, z)$.

1 Introduction

There are several well-studied graph polynomials, among them the chromatic polynomial, [Big93, GR01, DKT05], different versions of the Tutte polynomial, [Bol99, BR99, Sok05], and of the matching polynomial, [HL72, LP86, GR01], which are known to satisfy certain linear recurrence relations with respect to *deletion* of an edge, *contraction* of an edge, or deletion of an edge together with its endpoints, which we call *extraction* of an edge. The generalization of the chromatic polynomial, which was introduced by K.Dohmen, A.Pönitz and P.Tittman in [DPT03], happens to satisfy such recurrence relation as well. The question that arises is, what is the most general graph polynomial that satisfies similar linear recurrence relation.

In this paper[1] all the graphs are unlabeled ; multiple edges and self loops are allowed. We denote by $G = (V, E)$ the graph with vertex set V and edge set E.

[⋆] Partially supported by the Israel Science Foundation for the project "Model Theoretic Interpretations of Counting Functions" (2007–2010) and by a grant of the Fund for Promotion of Research of the Technion–Israel Institute of Technology.

[⋆⋆] Corresponding authors.

[1] A preliminary version of this paper has been posted as [AGM07].

Our investigation is motivated by the approach of J.G. Oxley and D.J.A. Welsh, [OW79], in defining a most general graph polynomial in five variables which satisfies a recurrence relation based on edge deletion and edge contraction, for which they show, that up to a simple prefactor, the resulting polynomial is the Tutte polynomial. Here we shall look for a most general polynomial which satisfies a recurrence relation based on three edge elimination operations: edge deletion, edge contraction and edge extraction.

1.1 Recursive Definitions of Graph Polynomials

Edge Elimination. We define three basic edge elimination operations on multigraphs:

- *Deletion.* We denote by G_{-e} the graph obtained from G by simply removing the edge e.
- *Contraction.* We denote by $G_{/e}$ the graph obtained from G by unifying the endpoints of e. Note that this operation can cause production of multiple edges and self loops.
- *Extraction.* We denote by $G_{\dagger e}$ the graph induced by $V \setminus \{u,v\}$ provided $e = \{u,v\}$. Extraction removes also all the edges adjacent to e.

Additionally, we require the polynomial to be *multiplicative* for disjoint unions, i.e., if $G_1 \oplus G_2$ denotes disjoint union of two graphs, then the polynomial $P(G_1 \oplus G_2) = P(G_1) \cdot P(G_2)$. This is justified by the fact that the polynomials occurring in the literature are usually multiplicative. The initial conditions are defined for an *empty set* (graph without vertices, usually, $P(\emptyset) = 1$) and for a single point $P(E_1)$. With respect to these operations, we recall the known recursive definitions of graph polynomials:

The Matching Polynomial. There are different versions of the matching polynomial discussed in the literature, for example *matching generating polynomial* $g(G,\lambda) = \sum_{i=0}^{n} a_i \lambda^i$ and *matching defect polynomial* $\mu(G,\lambda) = \sum_{i=0}^{n}(-1)^i a_i \lambda^{n-2i}$, where $n = |V|$ and a_i is the number of i-matchings in G. We shall use the bivariate version that incorporates the both above:

$$M(G,x,y) = \sum_{i=0}^{n} a_i x^{n-2i} y^i \tag{1}$$

The recursive definition of The matching polynomial satisfies the initial conditions $M(E_1) = x$ and $M(\emptyset) = 1$, and the recurrence relations

$$\begin{aligned} M(G) &= M(G_{-e}) + y \cdot M(G_{\dagger e}) \\ M(G_1 \oplus G_2) &= M(G_1) \cdot M(G_2) \end{aligned} \tag{2}$$

The Tutte Polynomial. We recall the definition of classical two-variable Tutte polynomial (cf. for example B.Bollobás [Bol99]):

Definition 1. Let $G = (V, E)$ be a (multi-)graph. Let $A \subseteq E$ be a subset of edges. We denote by $k(A)$ the number of connected components in the spanning subgraph (V, A). Then two-variable Tutte polynomial is defined as follows

$$T(G, x, y) = \sum_{A \subseteq E} (x-1)^{k(A)-k(E)} (y-1)^{|A|+k(A)-|V|} \tag{3}$$

The Tutte polynomial satisfies the initial conditions $T(E_1) = 1$ $T(\emptyset) = 1$ and has linear recurrence relation with respect to the operations above:

$$T(G, x, y) = \begin{cases} x \cdot T(G_{/e}, x, y) & \textit{if } e \textit{ is a bridge}, \\ y \cdot T(G_{-e}, x, y) & \textit{if } e \textit{ is a loop}, \\ T(G_{/e}, x, y) + T(G_{-e}, x, y) & \textit{otherwise} \end{cases}$$
$$T(G_1 \oplus G_2, x, y) = T(G_1, x, y) \cdot T(G_2, x, y) \tag{4}$$

However, we shall use in this paper the version of the Tutte polynomial used by A.Sokal [Sok05], known as the (bivariate) *partition function* of the Pott's model:

$$Z(G, q, v) = \sum_{A \subseteq E} q^{k(A)} v^{|A|} \tag{5}$$

The partition function of the Pott's model is co-reducible to the Tutte polynomial via

$$T(G, x, y) = (x-1)^{-k(E)} (y-1)^{-|V|} Z(G, (x-1)(y-1), y-1). \tag{6}$$

It satisfies the initial conditions $Z(E_1) = q$ and $Z(\emptyset) = 1$, and satisfies a recurrence relation which does not distinguish whether the edge e is a loop, a bridge, or none of the two:

$$Z(G, q, v) = v \cdot Z(G_{/e}, q, v) + Z(G_{-e}, q, v)$$
$$Z(G_1 \oplus G_2, q, v) = Z(G_1, q, v) \cdot Z(G_2, q, v) \tag{7}$$

The Bivariate Chromatic Polynomial. K.Dohmen, A.Pönitz and P.Tittman in [DPT03] introduced a polynomial $P(G, x, y)$ as follows: there is two disjoint sets of colors Y and Z, and a generalized proper coloring of a graph $G = (V, E)$ is a map $\phi : V \mapsto (Y \sqcup Z)$ such that for all $\{u, v\} \in E$, if $\phi(u) \in Y$ and $\phi(v) \in Y$, then $\phi(u) \neq \phi(v)$ (The set Y is called therefore "proper colors"). For two positive integers $x > y$, the value of the polynomial is the number of generalized proper colorings by x colors, y of them are proper. To make this definition meaningful for graphs with multiple edges we require that a vertex with a self-loop can be colored only by a color in $X \setminus Y$ and that a multiple edge does not affect colorings.

Proposition 1. *The polynomial $P(G, x, y)$ satisfies the initial conditions $P(E_1) = x$ and $P(\emptyset) = 1$, and the following recurrence relation:*

$$P(G, x, y) = P(G_{-e}, x, y) - P(G_{/e}, x, y) + (x-y) \cdot P(G_{\dagger e}, x, y)$$
$$P(G_1 \oplus G_2, x, y) = P(G_1, x, y) \cdot P(G_2, x, y) \tag{8}$$

Proof. Let $G = (V, E)$ be a graph, and $P(G, x, y)$ be the number of generalized colorings defined above. Let $v \in V$ be any vertex. We denote by $P^v(G, x, y)$ the number of generalized colorings of G, when v is colored by an "improper" color, i.e. $\phi(v) \in X \setminus Y$. Observing that a vertex v can have any color in $X \setminus Y$, and the coloring of $V - \{v\}$ does not depend on the color of v, we get:

Lemma 2. *$P^v(G, x, y) = (x-y) \cdot P(G_{-v}, x, y)$, where G_{-v} denotes the subgraph of G induced by $V \setminus \{v\}$.*

Let $e = \{u, v\} \in E$ be any edge of G, which is not a self-loop and not a multiple edge. Consider the number of colorings of G_{-e}. Any such coloring is either a coloring of G, or a coloring of $G_{/e}$, when the vertex $u = v$, which is produced by the contraction, is colored by a proper color. Together with Proposition 2, that raises:

$$P(G, x, y) = P(G_{-e}, x, y) - P(G_{/e}, x, y) + (x - y) \cdot P(G_{\dagger e}, x, y) \tag{9}$$

One can easily check that this equation is satisfied also for loops and multiple edges. Together with the fact that a singleton can be colored by any color, and the fact that the number of colorings is multiplicative, this completes the proof.

1.2 A Most General Edge Elimination Polynomial

Recursive Definition. Inspired by the characterization of the Tutte polynomial given in [OW79], see also [Bol99], Theorem 2 of Chapter 10, we look for the most general linear recurrence relation[2], which can be obtained on unlabeled graphs by introducing new variables, and which does not distinguish between local properties of the edge e which is to be eliminated[3]. To assure that the polynomial so defined is unique, we have to prove that its definition is not dependent on the order in which the edges are removed.

We start with the initial conditions $\xi(E_1) = x$ and $\xi(\emptyset) = 1$, and recurrence relation

$$\begin{aligned} \xi(G) &= w \cdot \xi(G_{-e}) + y \cdot \xi(G_{/e}) + z \cdot \xi(G_{\dagger e}) \\ \xi(G_1 \oplus G_2) &= \xi(G_1) \cdot \xi(G_2) \end{aligned} \tag{10}$$

We prove:

Theorem 3. *The recurrence relation (10) defines for each graph G a unique polynomial $\xi(G)$ if and only if one of the following conditions are satisfied:*

$$z = 0 \tag{11}$$

$$w = 1 \tag{12}$$

[2] The first paper to study general conditions under which linear recurrence relations define a graph invariant is D.N.Yetter [Yet90].

[3] It is conceivable that recurrence relations with various case distinctions depending on local properties of e and more variables give other "most general" polynomials. This is the reason why we speak of "a most general" edge elimination polynomial in the title of the paper.

Under condition (12), which allows a more general graph polynomial to be obtained, the recurrence relation (10) is restricted to

$$\begin{aligned} \xi(G,x,y,z) &= \xi(G_{-e},x,y,z) + y\cdot\xi(G_{/e},x,y,z) + z\cdot\xi(G_{\dagger e},x,y,z) \\ \xi(G_1 \oplus G_2,x,y,z) &= \xi(G_1,x,y,z)\cdot\xi(G_2,x,y,z) \\ \xi(E_1,x,y,z) &= x; \\ \xi(\emptyset,x,y,z) &= 1; \end{aligned} \tag{13}$$

Remark 4. *From this theorem one sees immediately that $\xi(G,x,y,z)$ gives, by choosing appropriate values for the variables and simple prefactors, the partition function of the Pott's model, the bivariate matching polynomial and the bivariate chromatic polynomial with all their respective substitution instances, including the classical chromatic polynomial, the Tutte polynomial and the independent set polynomial, [DHTL02, GH83]. The latter two polynomials are already substitution instances of the bivariate chromatic polynomial $P(G,x,y)$ of [DPT03]. The following table summarizes these observations.*

Pott's model	$Z(G,q,v) = \xi(G,q,v,0)$
Bivariate Tutte polynomial	$T(G,x,y) = (x-1)^{-k(E)}\cdot(y-1)^{-\lvert V\rvert}\cdot$ $\xi(G,(x-1)(y-1),(y-1),0)$
Bivariate matching polynomial	$M(G,x,y) = \xi(G,x,0,y)$
Bivariate chromatic polynomial	$P(G,x,y) = \xi(G,x,-1,x-y)$

Explicit definition. We now give an explicit form of the polynomial $\xi(G,x,y,z)$ using 3-partition expansion[4]:

Theorem 5. *Let $G=(V,E)$ be a (multi)graph. Then the edge elimination polynomial $\xi(G,x,y,z)$ can be calculated as*

$$\xi(G,x,y,z) = \sum_{(A\sqcup B)\subseteq E} x^{k(A\sqcup B)-k_{cov}(B)}\cdot y^{|A|+|B|-k_{cov}(B)}\cdot z^{k_{cov}(B)} \tag{14}$$

where by abuse of notation we use $(A\sqcup B)\subseteq E$ for summation over subsets $A,B\subseteq E$, such that the subsets of vertices $V(A)$ and $V(B)$, covered by respective subset of edges, are disjoint: $V(A)\cap V(B)=\emptyset$; $k(A)$ denotes the number of spanning connected components in (V,A), and $k_{cov}(B)$ denotes the number of covered connected components, i.e. the connected components of $(V(B),B)$.

Remark 6. *From Theorem 5 one can see that $\xi(G,x,y,z)$ is a polynomial definable in Monadic Second Order Logic, with quantification over sets of edges ($MSOL_2$), where an order over vertices is to be used for stating "number of connected sets", but the final result is order-independent. We shall not use logic in the sequel of the paper. For details the reader is referred to [Mak05].*

[4] A more precise name would be "Pair of two disjoint subsets expansion". We chose the name 3-partition expansion, as any two disjoint subsets induce a partition into three sets.

Counting Weighted Graph Homomorphisms. Another common way to define graph polynomials is counting weighted graph homomorphisms. Let the weighted graph $H = (V_H, E_H)$ be defined as follows:

- H is obtained by joining 3 cliques with all loops: $H = K_x^A \bowtie K_p^B \bowtie K_p^I$, such that K_x^A has x vertices, K_p^B and K_p^I have p vertices each.
- We denote by V^A the vertices of K^A, by V^B the vertices of K^B and by V^I the vertices of K^I
- The weight function $w = (\alpha, \beta)$: $\alpha : V_H \mapsto \mathbb{R}$ and $\beta : E_H \mapsto \mathbb{R}$
- $\alpha(v) = \begin{cases} 1 & if\ v \in (V^A \cup V^B) \\ -1 & otherwise \end{cases}$
- $\beta(u,v) = \begin{cases} y+1 & if\ (u = v) \wedge (u, v \in (V^A \cup V^B)) \\ 1 & otherwise \end{cases}$

In [AGM08] the following is shown:

Theorem 7. *Let $Z_H(G)$ be a homomorphism function of a graph $G = (V, E)$ into a weighted graph H above.*

$$Z_H(G) = \sum_{\substack{h\,:\,V \mapsto V_H \\ homomorphism}} \prod_{v \in V} \alpha(h(v)) \prod_{(u,v) \in E} \beta(h(u), h(v))$$

Then, for all nonnegative integers x and p and all $y \in \mathbb{R}$, we have

$$\xi(G, x, y, p \cdot y) = Z_H(G)$$

Remark 8. *A general characterization of graph parameters which can be obtained from homomorphism functions by choosing appropriate weights is given in [FLS07]. This characterization requires that the weights α are positive reals. However, in Theorem 7, we use negative values for α.*

2 The Most General Recurrence Relation

We are looking for the most general linear recurrence relation with respect to edge deletion, edge contraction and edge extraction operation that can be obtained by introducing new variables. Recall that we are interested in a *graph invariant*, e.i. the resulting function should not depend on the order of graph deconstruction. Moreover, this invariant should be a multiplicative graph polynomial.

From this consideration alone we obtain the initial condition $\xi(\emptyset) = 1$; and the product rule:

$$\xi(G_1 \oplus G_2) = \xi(G_1) \cdot \xi(G_2)$$

Indeed, the disjoint union with an empty set gives the same graph, so the resulting function should also remain the same.

We now formulate the edge elimination rule introducing a new variable wherever we can. We set

$$\begin{aligned}\xi(G,x,y,z,t) &= t\cdot\xi(G_{-e},x,y,z,t)+y\cdot\xi(G_{/e},x,y,z,t)+z\cdot\xi(G_{\dagger e},x,y,z,t)\\ \xi(E_1,x,y,z,t) &= x; \end{aligned}\tag{15}$$

Let G be a graph as presented on Fig. 1. Note that the subgraphs H_1, H_{1-u}, H_2 and H_{2-w} can be different and have (in general) different ξ. Since we are

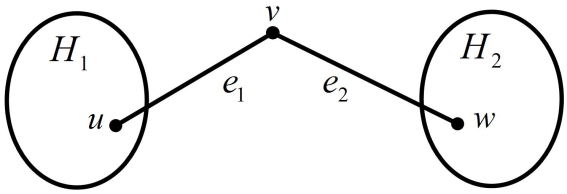


Fig. 1. Testing order invariance of edge removal

looking for a graph invariant, we must obtain the same result by applying the edge elimination rule first on the edge e_1 and then on the edge e_2, as in case when we apply the edge elimination rule first on the edge e_2 and then on the edge e_1.

$$\begin{aligned}\xi(G) &= t\cdot\xi(G_{-e_1})+y\cdot\xi(G_{/e_1})+z\cdot\xi(G_{\dagger e_1}) =\\ &= t\cdot\xi(H_1)\cdot[x\cdot t\cdot\xi(H_2)+y\cdot\xi(H_2)+z\cdot\xi(H_{2-w})]+\\ &\quad y\cdot\left[t\cdot\xi(H_1)\xi(H_2)+y\cdot\xi(G_{/e_1/e_2})+z\cdot\xi(H_{1-u})\xi(H_{2-w})\right]+\\ &\quad z\cdot\xi(H_{1-u})\xi(H_2)\end{aligned}\tag{16}$$

On the other hand,

$$\begin{aligned}\xi(G) &= t\cdot\xi(G_{-e_2})+y\cdot\xi(G_{/e_2})+z\cdot\xi(G_{\dagger e_2}) =\\ &= t\cdot\xi(H_2)\cdot[x\cdot t\cdot\xi(H_1)+y\cdot\xi(H_1)+z\cdot\xi(H_{1-u})]+\\ &\quad y\cdot\left[t\cdot\xi(H_1)\xi(H_2)+y\cdot\xi(G_{/e_1/e_2})+z\cdot\xi(H_{1-u})\xi(H_{2-w})\right]+\\ &\quad z\cdot\xi(H_{2-w})\xi(H_1)\end{aligned}\tag{17}$$

Solving the above two equations, we get:

$$tz\cdot\xi(H_1)\xi(H_{2-w})+z\cdot\xi(H_{1-u})\xi(H_2)=tz\cdot\xi(H_{1-u})\xi(H_2)+z\cdot\xi(H_1)\xi(H_{2-w})$$

Hence, we have the following necessary condition:

$$z=0 \quad \textit{or} \tag{18}$$

$$t=1 \quad \textit{or} \tag{19}$$

$$\xi(H_1)\xi(H_{2-w})=\xi(H_{1-u})\xi(H_2) \quad \textit{for any } H_1 \textit{ and } H_2 \tag{20}$$

Under condition (20) we get the polynomial $\xi(G)=x^{|V(G)|}$ which we also get under the assumption $z=0$ or $t=1$. In case of $z=0$ it can be easily seen that the resulting function is a substitution instance of the Pott's model:

$$\xi(G, x, y, 0, t) = t^{|E|} \cdot Z(G, x, \frac{y}{t}) \tag{21}$$

Since the partition function of the Pott's model can be also obtained when $t = 1$, the latter case is considered more general. That brings us back to the recurrence relation (13). To complete the proof of Theorem 3, we need now to show that any two steps of the graph decomposition using (13) are interchangeable. This involves two parts,

- Edge elimination and disjoint union, and
- Decomposition of a graph by elimination of any two edges in different order.

The proof of the first part is simple. Let G be a disjoint union of two graphs: $G = H_1 \oplus H_2$. Without loss of generality, assume that the edge e, which is being eliminated, is in $E(H_1)$. Then use the linearity of our recurrence relation to show that

$$\begin{aligned}\xi(G) &= \big(\xi(H_{1-e}) + y \cdot \xi(H_{1/e}) + z \cdot \xi(H_{1\dagger e})\big) \cdot \xi(H_2) = \\ &= \xi(H_{1-e}) \cdot \xi(H_2) + y \cdot \xi(H_{1/e}) \cdot \xi(H_2) + z \cdot \xi(H_{1\dagger e}) \cdot \xi(H_2)\end{aligned}$$

The second part requires analyzing of three possible cases:

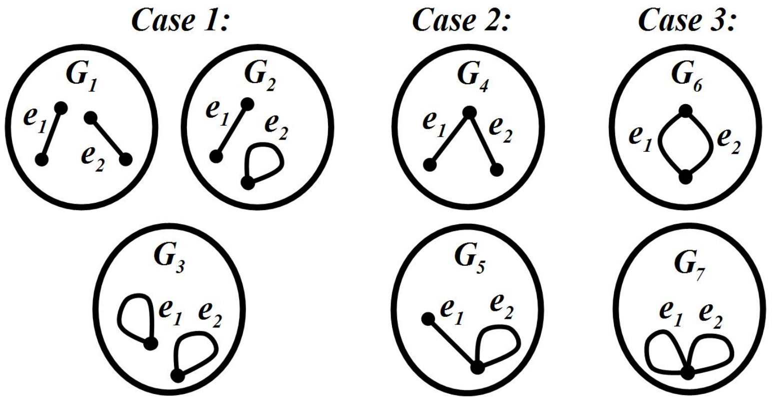


Fig. 2. Different cases to check order invariance of edge removal

- Case 1: Two the edges have no common vertices (graphs G_1, G_2, G_3);
- Case 2: Two the edges have one common vertex, and at least one exclusive vertex (graphs G_4, G_5);
- Case 3: Two the edges have no exclusive vertices (graphs G_6, G_7).

The detailed analysis of these cases is straightforward.

3 The Explicit Form of $\xi(G)$

We need to show that the expression (14) satisfies the *initial conditions* of (13), that it is *multiplicative* and that it also satisfies the *edge elimination rule* of (13). Then by induction on the number of edges in G the theorem holds.

The first fact is trivial; the second one can be easily checked by reader. Indeed, the summation over subsets of edges of $G(V,E) = G_1(V_1,E_1) \oplus G_2(V_2,E_2)$ can be regarded as a summation over the subsets of E_1, and then independently over the subsets of E_2. Therefore, we just need to prove that

Lemma 9. *The explicit expression given by (14) satisfies the edge elimination rule of (13).*

Proof. Let $G = (V,E)$ be the (multi)graph of interest. Let $N(G)$ be defined as

$$N(G,x,y,z) = \sum_{(A \sqcup B) \subseteq E} x^{k(A \sqcup B) - k_{cov}(B)} \cdot y^{|A|+|B|-k_{cov}(B)} \cdot z^{k_{cov}(B)} \qquad (22)$$

where $k(A)$ denotes the number of connected components in (V,A), and $k_{cov}(B)$ denotes the number of the connected components of $(V(B),B)$, where $V(B) \subseteq V$ are the vertices covered by the edges of B. Let e be the edge we have chosen to reduce. Any particular choice of A and B can be regarded as a vertex-disjoint edge coloring in 2 colors A and B, when part of the edges remains uncolored. We divide all the coloring into three disjoint cases:

- Case 1: e is uncolored;
- Case 2: e is colored by B, and it is the only edge of a colored connected component;
- Case 3: All the rest. That means, e is colored by A, or e is colored by B but it is not the only edge of a colored connected component.

In case 1, we just sum over colorings of G_{-e}:

$$N_1(G) = \sum_{(A \sqcup B) \models \ Case\ 1} x^{k(A \sqcup B) - k_{cov}(B)} \cdot y^{|A|+|B|-k_{cov}(B)} \cdot z^{k_{cov}(B)} = N(G_{-e})$$

In case 2, the edge e is a connected component of $(V(B),B)$. Therefore, if we analyze now $N(G_{\dagger e})$, we will get

- The number of edges colored by A is the same;
- The number of edges colored by B is reduced by one;
- The total number of colored connected components is reduced by one;
- The number of covered connected components colored B is reduced by one;

This gives us

$$N_2(G) = \sum_{(A \sqcup B) \models \ Case\ 2} x^{k(A \sqcup B) - k_{cov}(B)} \cdot y^{|A|+|B|-k_{cov}(B)} \cdot z^{k_{cov}(B)} = z \cdot N(G_{\dagger e})$$

And finally, in case 3, e is a part of a bigger colored connected component, or it is alone a connected component colored by A. In this case, we analyze the colorings of $G_{/e}$:

– Either $|A|$ or $|B|$ is reduced by 1, the other remained the same;
– The total number of colored connected components remained the same;
– The number of covered connected components colored B remained the same.

According to the above,

$$N_3(G) = \sum_{(A \sqcup B) \models \ Case\ 3} x^{k(A \sqcup B) - k_{cov}(B)} \cdot y^{|A|+|B|-k_{cov}(B)} \cdot z^{k_{cov}(B)} = y \cdot N(G_{/e})$$

which together with $N(G) = N_1(G) + N_2(G) + N_3(G)$ completes the proof.

4 Computational Complexity of $\xi(G)$

In this section we consider the complexity of computation of $\xi(G)$. In general, this polynomial is $\sharp$**P**-hard to compute, as every instance stated in the Remark 4 is $\sharp$**P**-hard. Moreover, C. Hoffmann proves in [Hof08] that at every point $(x, y, z) \in \mathbb{Q}$, with $x \neq 0$, $z \neq -xy$, $(x, z) \notin \{(1, 0), (2, 0)\}$, $y \notin \{-2, -1, 0\}$, evaluating $\xi(G, x, y, z)$ is $\sharp$**P**-hard.

Recall that, according to Remark 6, the formula (14) can be used to give an order invariant definition in Monadic Second Order Logic, with quantification over sets of edges, and an auxiliary order. Hence, from the general theorem from [Mak05, Mak04], we immediately get that $\xi(G)$ is polynomial time computable on graphs of tree-width at most k where the exponent of the run time is independent of k. The drawback of the general method of [Mak05, Mak04] lies in the huge hidden constants, which make it practically unusable. However, an explicit dynamic algorithm for computing the polynomial $\xi(G)$ on graphs of bounded tree-width, given the tree decomposition of the graph, where the constants are simply exponential in k, can be constructed along the same ideas as presented in [Tra06, FMR08, MRAG06].

5 Conclusions and Open Questions

We have introduced a new graph polynomial $\xi(G, x, y, z)$ from which the Tutte polynomial, the matching polynomial and the bivariate chromatic polynomial can be obtained by simple evaluations. We have given three equivalent definitions of this polynomial, and we have shown that it is the most general graph polynomial satisfying a recurrence relation (without case distinctions) with respect to three edge elimination operations.

There are still some challenging open questions.

Recursive Definitions with Case Distinctions. Contrary to the approach in [OW79], we have avoided case distinctions in the recurrence relation. This was justified because it still gives the Tutte polynomials as special cases. Alternatively we could have introduced a polynomial in more variables which does incorporate a case distinction with respect to some local properties of an edge, such as being a bridge or a loop, or we could have allowed the deletion of single vertices, and distinguish between cases where they are isolated with or without loops, etc.

Question 1. Does one get essentially stronger polynomials if one allows also deletion of single vertices and takes into account case distinctions?

Distinctive Power. We know that the polynomial $\xi(G)$ has at least the same distinctive power as the Tutte polynomial and the bivariate chromatic polynomial together, but more than every one of them individually. Indeed, since $T(G, x, y)$ and $P(G, x, y)$ are both substitution instances of $\xi(G)$, if $\xi(G)$ coincides for two graphs, so do $T(G, x, y)$ and $P(G, x, y)$. On the other hand, we do not know whether $\xi(G)$ has more distinctive power.

Question 2. Are there two graphs G_1, G_2 such that for all x, y we have $T(G_1, x, y) = T(G_2, x, y)$ and $P(G_1, x, y) = P(G_2, x, y)$, but such that for some $x, y, z \;\; \xi(G, x, y, z) \neq \xi(G_2, x, y, z)$?

Complexity on Graph Classes of Bounded Clique-Width. We have noted that for graphs of tree-width at most k computing the edge reduction polynomial $\xi(G)$ is fixed parameter tractable (FPT) in the sense of [DF99, FG06]. Another graph parameter, introduced in [CO00] and discussed there is the clique-width. It was stated as an open problem whether the Tutte polynomial is fixed parameter tractable for graphs of clique-width at most k, [GHN05, MRAG06]. Very recently, F. Fomin, P. Golovach, D. Lokshtanov and S. Saurabh [FGLS08] showed that computing the chromatic number of graphs of clique-width at most k is $W[1]$-hard, and therefore not fixed parameter tractable. It follows from this that it is also true for evaluating the Tutte polynomial and our polynomial $\xi(G)$. Further more, this shows that the results on the complexity of evaluating the chromatic polynomial in [MRAG06] are optimal.

Acknowledgments. We would like to thank B. Courcelle, T. Kotek and to the anonymous referees, for their comments on early versions of this paper.

References

[AGM07] Averbouch, I., Godlin, B., Makowsky, J.A.: The most general edge elimination polynomial. arXiv (2007), `http://uk.arxiv.org/pdf/0712.3112.pdf`

[AGM08] Averbouch, I., Godlin, B., Makowsky, J.A.: An extension of the bivariate chromatic polynomial (submitted, 2008)

[Big93] Biggs, N.: Algebraic Graph Theory, 2nd edn. Cambridge University Press, Cambridge (1993)

[Bol99] Bollobás, B.: Modern Graph Theory. Springer, Heidelberg (1999)

[BR99] Bollobás, B., Riordan, O.: A Tutte polynomial for coloured graphs. Combinatorics, Probability and Computing 8, 45–94 (1999)

[CO00] Courcelle, B., Olariu, S.: Upper bounds to the clique–width of graphs. Discrete Applied Mathematics 101, 77–114 (2000)

[DF99] Downey, R.G., Fellows, M.F.: Parametrized Complexity. Springer, Heidelberg (1999)

[DHTL02] Dong, F.M., Hendy, M.D., Teo, K.L., Little, C.H.C.: The vertex-cover polynomial of a graph. Discrete Mathematics 250, 71–78 (2002)

[DKT05] Dong, F.M., Koh, K.M., Teo, K.L.: Chromatic Polynomials and Chromaticity of Graphs. World Scientific, Singapore (2005)

[DPT03] Dohmen, K., Pönitz, A., Tittmann, P.: A new two-variable generalization of the chromatic polynomial. Discrete Mathematics and Theoretical Computer Science 6, 69–90 (2003)

[FG06] Flum, J., Grohe, M.: Parameterized complexity theory. Springer, Heidelberg (2006)

[FGLS08] Fomin, F., Golovach, P., Lokshtanov, D., Saurabh, S.: Clique-width: On the price of generality. Department of Informatics, University of Bergen, Norway (preprint, 2008)

[FLS07] Freedman, M., Lovász, L., Schrijver, A.: Reflection positivity, rank connectivity, and homomorphisms of graphs. Journal of AMS 20, 37–51 (2007)

[FMR08] Fischer, E., Makowsky, J.A., Ravve, E.V.: Counting truth assignments of formulas of bounded tree width and clique-width. Discrete Applied Mathematics 156, 511–529 (2008)

[GH83] Gutman, I., Harary, F.: Generalizations of the matching polynomial. Utilitas Mathematicae 24, 97–106 (1983)

[GHN05] Giménez, O., Hliněný, P., Noy, M.: Computing the Tutte polynomial on graphs of bounded clique-width. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 59–68. Springer, Heidelberg (2005)

[GR01] Godsil, C., Royle, G.: Algebraic Graph Theory. Graduate Texts in Mathematics. Springer, Heidelberg (2001)

[HL72] Heilmann, C.J., Lieb, E.H.: Theory of monomer-dymer systems. Comm. Math. Phys 28, 190–232 (1972)

[Hof08] Hoffmann, C.: A most general edge elimination polynomial–thickening of edges (2008) arXiv:0801.1600v1 [math.CO]

[LP86] Lovasz, L., Plummer, M.D.: Matching Theory. Annals of Discrete Mathematics, vol. 29. North-Holland, Amsterdam (1986)

[Mak04] Makowsky, J.A.: Algorithmic uses of the Feferman-Vaught theorem. Annals of Pure and Applied Logic 126(1-3), 159–213 (2004)

[Mak05] Makowsky, J.A.: Colored Tutte polynomials and Kauffman brackets on graphs of bounded tree width. Disc. Appl. Math. 145(2), 276–290 (2005)

[MRAG06] Makowsky, J.A., Rotics, U., Averbouch, I., Godlin, B.: Computing graph polynomials on graphs of bounded clique-width. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 191–204. Springer, Heidelberg (2006)

[OW79] Oxley, J.G., Welsh, D.J.A.: The Tutte polynomial and percolation. In: Bundy, J.A., Murty, U.S.R. (eds.) Graph Theory and Related Topics, pp. 329–339. Academic Press, London (1979)

[Sok05] Sokal, A.: The multivariate Tutte polynomial (alias Potts model) for graphs and matroids. In: Survey in Combinatorics, 2005. London Mathematical Society Lecture Notes, vol. 327, pp. 173–226 (2005)

[Tra06] Traldi, L.: On the colored Tutte polynomial of a graph of bounded treewidth. Discrete Applied Mathematics 154(6), 1032–1036 (2006)

[Yet90] Yetter, D.N.: On graph invariants given by linear recurrence relations. Journal of Combinatorial Theory, Series B 48(1), 6–18 (1990)

Approximating the Metric TSP in Linear Time

Davide Bilò[1], Luca Forlizzi[2], and Guido Proietti[2,3]

[1] Institut für Theoretische Informatik, ETH, Zürich, Switzerland
[2] Dipartimento di Informatica, Università di L'Aquila, Italy
[3] Istituto di Analisi dei Sistemi ed Informatica, CNR, Roma, Italy
{davide.bilo,forlizzi,proietti}@di.univaq.it

Abstract. Given a metric graph $G = (V, E)$ of n vertices, i.e., a complete graph with an edge cost function $c : V \times V \to \mathbb{R}_{\geq 0}$ satisfying the triangle inequality, the *metricity degree* of G is defined as $\beta = \max_{x,y,z \in V} \left\{ \frac{c(x,y)}{c(x,z)+c(y,z)} \right\} \in \left[\frac{1}{2}, 1\right]$. This value is instrumental to establish the approximability of several NP-hard optimization problems definable on G, like for instance the prominent *traveling salesman problem*, which asks for finding a Hamiltonian cycle of G of minimum total cost. In fact, this problem can be approximated quite accurately depending on the metricity degree of G, namely by a ratio of either $\frac{2-\beta}{3(1-\beta)}$ or $\frac{3\beta^2}{3\beta^2-2\beta+1}$, for $\beta < \frac{2}{3}$ or $\beta \geq \frac{2}{3}$, respectively. Nevertheless, these approximation algorithms have $O(n^3)$ and $O(n^{2.5} \log^{1.5} n)$ running time, respectively, and therefore they are superlinear in the $\Theta(n^2)$ input size. Thus, since many real-world problems are modeled by graphs of huge size, their use might turn out to be unfeasible in the practice, and alternative approaches requiring only $O(n^2)$ time are sought. However, with this restriction, all the currently available approaches can only guarantee a 2-approximation ratio for the case $\beta = 1$, which means a $\frac{2\beta^2}{2\beta^2-2\beta+1}$-approximation ratio for general $\beta < 1$. In this paper, we show how to enhance –without affecting the space and time complexity– one of these approaches, namely the classic *double-MST* heuristic, in order to obtain a 2β-approximate solution. This improvement is effective, since we show that the double-MST heuristic has in general a performance ratio strictly larger that 2β, and we further show that *any* re-elaboration of the shortcutting phase therein provided, cannot lead to a performance ratio better than 2β.

Keywords: Traveling Salesman Problem, Metric Graphs, NP-hardness, Linear-time Approximation Algorithms.

1 Introduction

The *Traveling Salesman Problem* (TSP, for short) is one of the most prominent and studied combinatorial optimization problems. It is defined as follows: Given a complete, undirected graph $G = (V, E)$ of n vertices, with an edge cost function $c : V \times V \to \mathbb{R}_{\geq 0}$, find a Hamiltonian cycle $H = (V, E(H))$ of G (i.e., a simple cycle that spans all the vertices of G) of minimum total *cost* $c(H) = \sum_{e \in E(H)} c(e)$.

The TSP is intractable and does not admit any polynomial-time constant-ratio approximation algorithm, unless $\mathsf{P} = \mathsf{NP}$. On the other hand, when the edge cost function induces a metric on G (i.e., for any $x, y, z \in V, c(x,z) \leq c(x,y) + c(y,z)$), things improve sensibly. More specifically, in such a case the problem is known as the *metric* TSP, and it can be approximated by the Christofides algorithm [4] within an approximation factor of 3/2, but still it remains NP-hard to find an approximate solution for it within a factor better than 220/219 [9]. In some special metric cases, however, the approximability of the problem becomes really effective. For instance, when vertices of G are actually points in a fixed-dimension Euclidean space, and the edge cost function is their Euclidean distance, then the problem (also known as *geometric* TSP) admits a polynomial-time approximation scheme [1].

Getting back to the general metric case, it is interesting to notice that one can define suitable approximation algorithms depending on the *metricity degree* of G, which is defined as $\beta = \max_{x,y,z \in V} \left\{ \frac{c(x,y)}{c(x,z)+c(y,z)} \right\} \in \left[\frac{1}{2}, 1\right]$. More precisely, for $\frac{1}{2} \leq \beta \leq \frac{2}{3}$, there exists a $\frac{2-\beta}{3(1-\beta)}$-approximation algorithm [2],[1] while for $\frac{2}{3} < \beta \leq 1$, the best approximation algorithm is still the Christofides' one, with a ratio of $\frac{3\beta^2}{3\beta^2-2\beta+1}$. In the following, we will therefore assume that the input graph is *β-metric* (i.e., its metricity degree is equal to β), and the metric TSP with this additional parameter will be correspondingly named the *β-metric traveling salesman problem* (β-MTSP for short, also known as *strengthened-metric* TSP).

Both the aforementioned best-known approximation algorithms for the β-MTSP have a shortcoming, however: their time complexity is superlinear in the input size. Indeed, the former runs in $O(n^3)$ time,[2] while the latter runs in $O(n^{2.5} \log^{1.5} n)$ time [8]. From a practical point of view, this might result in a severe drawback, since often the input graph is massive. For instance, the well-known benchmark TSPLIB of TSP instances [10] contains several input graphs with order of 10^5 vertices (most of them coming from roadmaps, and then most likely metric), for which these algorithms become computationally expensive. In such a case, a linear-time approximation algorithm would be desirable, even if a (possibly small) price in terms of performance ratio has to be paid. In this paper, we exactly aim at this goal. We emphasize that insisting on this trade-off is not completely novel, and linear-time constrained approximation algorithms have been already developed in the past, but as far as we know, this was done only for high-degree polynomial-time solvable problems (e.g., the *weighted matching problem* [6] and the *watchman route problem* [11]).

[1] Notice that for $\beta = 1/2$, all the graph edges have the same cost, and the problem becomes trivial.

[2] This bound can be obtained after conditioning the costs of the input graph G in such a way that the problem of determining a minimum-cost cycle cover of G, which represents the core procedure of the algorithm in [2], reduces to that of finding a maximum-cost simple 2-matching in the transformed graph, which can be solved in $O(n^3)$ time [7].

Actually, designing a linear-time approximation algorithm for the β-MTSP starting from the efficient implementation given in [8] of the Christofides algorithm sounds prohibitive, since this uses the very efficient $O(n^{2.5}\log^{1.5} n)$ time approximate minimum-cost perfect matching algorithm as a brick for constructing a feasible solution (and in the worst case such an algorithm runs on a graph having $\Theta(n)$ vertices). The same holds for the algorithm of Böckenhauer *et al.* [2], which makes use of the long-standing $O(n^3)$ time procedure for computing a maximum-cost perfect 2-matching. Thus, a different approach needs to be used.

A well-known approximation algorithm for the metric TSP is the *double-tree shortcutting* algorithm, which works as follows: first, construct a *Minimum Spanning Tree* (MST, for short) T of G; after, construct a Eulerian tour D on the multigraph $G' = T \cup T$, and finally return a Hamiltonian cycle H from D by shortcutting in G repeated vertices in the Eulerian tour. This algorithm (which we will call `Double-MST shortcut` in the rest of the paper) is easily seen to guarantee a 2-approximation ratio, and then it turns out to be a $\frac{2\beta^2}{2\beta^2-2\beta+1}$-approximation algorithm for the β-MTSP [2].[3] As a matter of fact, it is then always outperformed by its super-linear counterparts. In an effort of improving this gap, by maintaining the linear-time constraint, we therefore develop an easy modification of such an algorithm, which allows us to obtain, by means of an accurate analysis, a 2β-approximate solution, thus beating the `Double-MST shortcut` for any $1/2 < \beta < 1$. This improvement is effective, since as a side result, we show that for any $1/2 < \beta < 1$, the performance ratio of the `Double-MST shortcut` is strictly larger than 2β. As a matter of fact, the reduction of the theoretical gap with respect to the superlinear approximation algorithm is significant: for instance, for all $1/2 < \beta \leq 3/4$, our algorithm is only about 5% away from it in the worst case, while for the `Double-MST shortcut` this gap raises to about 26%.

It is worth noticing that the `Double-MST shortcut` approach is at the basis of one of the best-performing heuristics available for the TSP, namely that developed by Deineko and Tiskin[3,5]. This heuristic computes in $O(4^d n^2)$ time (where d denotes the maximum node-degree in T) the minimum cost Hamiltonian cycle that can be obtained using the `Double-MST shortcut` approach. From now on, we will call this heuristic with `Double-MST Min-weight shortcut`. Extensive computational experiments have shown that this strategy allows very good approximations, often better that those obtained with heuristics derived by the Christofides algorithm, although it requires a running time which might be exponential. However, despite its high performances registered in practice, no theoretical approximation guarantee better than that of `Double-MST shortcut` was exhibited in [5]. Our analysis immediately shows that the algorithm given in [5] has actually an approximation ratio of 2β, for any $\beta < 1$. Moreover, we also prove that this ratio is asymptotically tight. Due to the practical relevance

[3] This ratio results from the fact that if $\mathcal{A}$ is an α-approximation algorithm for the metric TSP, then $\mathcal{A}$ is an $\frac{\alpha\cdot\beta^2}{\beta^2+(\alpha-1)(1-\beta)^2}$ -approximation algorithm for the β-MTSP (see [2] for further details).

of heuristics based on the minimum-weight shortcutting of D, we see this as a noteworthy consequence of our results.

The rest of the paper is organized as follows: in Section 2 we show that the `Double-MST shortcut` algorithm does not return a 2β-approximate solution; in Section 3 we present an enhanced version of the previous algorithm and prove it computes a 2β-approximate solution; in Section 4 we prove that `Double-MST Min-weight shortcut` does not compute a solution within a constant factor smaller than 2β, thus proving that no algorithm based on the `Double-MST shortcut` approach can be significantly better than the one presented in the previous section. Finally, in Section 5, we provide a graphical comparison (in terms of approximation ratio) between our algorithm and its counterparts.

2 `Double-MST Shortcut` Is Not a 2β-Apx Algorithm

From now on, we will assume that the input graph $G = (V, E)$ is an instance of the β-MTSP. Moreover, paths and cycles will be expressed explicitly throughout the sequence of constituting vertices. The `Double-MST shortcut` algorithm does not prescribe how to shortcut repeated vertices in the Eulerian cycle obtained by merging the two copies of the MST. An easy to state and widely used rule, is to perform shortcut according to the visiting order of vertices in a *depth-first search* of the MST. Applying this rule, the `Double-MST shortcut` algorithm can be stated as follows:

Algorithm 1. An implementation of the `Double-MST shortcut` algorithm

```
Input: A $\beta$-metric graph $G = (V, E)$.
Step 1. Construct an MST $T$ of $G$.
Step 2. Perform a depth-first search on $T$ starting from an arbitrary vertex $v_0$. Let
    $v_0, v_1, \ldots, v_{n-1}$ be the sequence of vertices in the order they are visited.
Output: Return the cycle $H = v_0 v_1 \ldots v_{n-1} v_0$ of $G$.
```

As said in the introduction, this algorithm turns out to be a $\frac{2\beta^2}{2\beta^2-2\beta+1}$-approximation algorithm for the β-MTSP. Although this ratio has not been proven to be tight, in the following we provide an input instance showing that the cost of the Hamiltonian cycle computed by `Double-MST shortcut` can be strictly larger than 2β times the optimal one.

For a given $\frac{1}{2} < \beta < 1$, let $G = (V, E)$ be a β-metric graph with vertex set $V = \{v_0, x_1, y_1, z_1, \ldots, x_h, y_h, z_h\}$, and such that the edge costs are defined as follows:

- $c(v_0, x_i) = 1, c(v_0, y_i) = \frac{\beta}{1-\beta}, c(v_0, z_i) = 2\beta, \forall 1 \le i \le h$;
- $c(y_i, v) = \frac{\beta}{1-\beta}, \forall 1 \le i \le h$ and $\forall v \in V \setminus \{y_i\}$;
- $c(x_i, x_j) = 2\beta, \forall 1 \le i, j \le h$;
- $c(z_i, z_j) = 2\beta, \forall 1 \le i, j \le h$;

– $c(x_i, z_j) = 1, \forall 1 \le i \le j \le h$;
– $c(x_i, z_j) = 2\beta^2 + \beta, \forall 1 \le j < i \le h$.

An MST T of G is formed by edges $(v_0, x_i), (x_i, y_i), (x_i, z_i)$, for each $1 \le i \le h$.In Figure 1 the MST for the case $h = 3$ is depicted. Assume now that the `Double-MST shortcut` algorithm takes v_0 as root of T, and performs a depth-first search visiting y_i before z_i, for all $1 \le i \le h$. Then, the algorithm builds the solution $H = v_0 x_1 y_1 z_1 x_2 y_2 z_2 \dots x_h y_h z_h v_0$, having cost (see Figure 2 for the case $h = 3$)

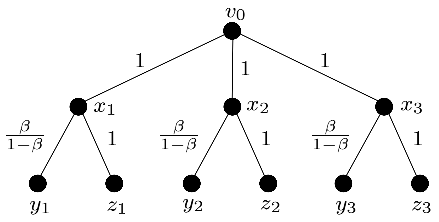


Fig. 1. An MST of G

$$c(H) = \frac{1}{1-\beta}\left(h(3\beta + \beta^2 - 2\beta^3) + 1 - 3\beta^2 + 2\beta^3\right).$$

Consider the solution $H^* = y_1 y_2 \dots y_h v_0 x_h z_h x_{h-1} z_{h-1} \dots x_1 z_1 y_1$ which has cost (see Figure 2 for the case $h = 3$)

$$c(H^*) = \frac{1}{1-\beta}\left(h(2-\beta) + \beta\right).$$

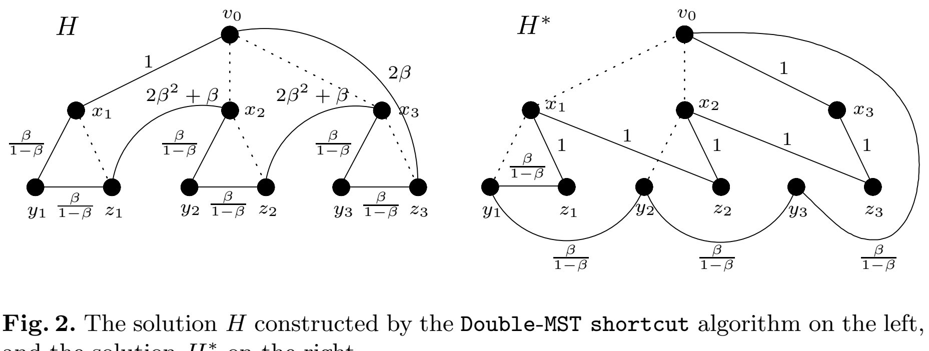


Fig. 2. The solution H constructed by the `Double-MST shortcut` algorithm on the left, and the solution H^* on the right

Therefore, if OPT denotes an optimal solution, we have

$$\frac{c(H)}{c(\text{OPT})} \ge \frac{c(H)}{c(H^*)} = \frac{h(3\beta + \beta^2 - 2\beta^3) + 1 - 3\beta^2 + 2\beta^3}{h(2-\beta) + \beta} > 2\beta,$$

where the last inequality holds whenever $h > \frac{2}{\beta(1-\beta)} - \frac{1-\beta}{\beta}$.

3 The 2β-Approximation Algorithm

In this section we provide a refinement of `Double-MST shortcut` which returns a 2β-approximate solution for the β-MTSP (its pseudo-code is given in Algorithm 2). In the following, given a vertex v of the considered rooted tree T, we denote by T_v the subgraph of T induced by v and by its children in T. We say that $v' \in T_v$ is a *closest child* of v if $c(v,v') \leq c(v,v''), \forall v'' \in T_v$.

Algorithm 2. The `Refined Double-MST shortcut` algorithm

```
Input: A $\beta$-metric graph $G = (V, E)$.
Step 1. Construct an MST $T$ of $G$.
Step 2. Root $T$ at any arbitrary vertex of degree 1. Starting from the root, do a careful
    depth-first search of $T$, i.e., a depth-first search with the following additional rule:
    when the visit of an internal vertex of $T$ begins, choose one of its closest children
    as the next vertex to visit. Let $v_0, v_1, \ldots, v_{n-1}$ be the sequence of vertices in the
    order they are visited.
Output: Return the cycle $H = v_0 v_1 \ldots v_{n-1} v_0$ of $G$.
```

3.1 Algorithm Analysis

In this section we prove that `Refined Double-MST shortcut` returns a 2β-approximate solution. The core idea of the proof is to use properties of a careful depth first search together with the fact that in β-metric graphs, the cost of a direct edge between two vertices u and v is never larger than β times the cost of any other path between u and v. More formally, a simple proof by induction shows that

Lemma 1. *Let $P = v_0 v_1 \ldots v_k$ be a simple path in G of length $k \geq 3$. Then*

$$c(v_0, v_k) \leq \beta\, c(P) - (1-\beta) \sum_{j=1}^{k-2} \left(\beta^j \sum_{i=0}^{k-1-j} c(v_i, v_{i+1}) \right).$$

□

Let us denote by A the set of edges of T belonging to H, and by $B = E(H) \setminus A$ the remaining edges in H. The following lemma holds:

Lemma 2. *For every non-leaf vertex v_j of T, we have that $|A \cap E(T_{v_j})| = 1$.*

Proof. Clearly, the lemma holds for $j = 0$. Hence, assume that $j > 0$. There are exactly two edges in $E(H)$ incident to v_j. One of them connects v_j to v_{j-1}, so it does not belong to T_{v_j} (indeed, for the depth-first search properties, v_{j-1} cannot be a descendent of v_j). If v_j is not a leaf, one of its children is visited immediately after v_j, hence $(v_j, v_{j+1}) \in E(T_{v_j})$. □

Observe that edges in A are exactly edges of the form (u, v), where u is an internal node of T while v is a closest child of u. Moreover, observe that since the root v_0 has degree 1, then edges $(v_0, v_1), (v_1, v_2)$ always belong to A. For any edge $e \in E(H)$, let P_e denote the unique path in T joining the two endpoints of e (notice that if $e \in A$, then P_e consists of e). Because of the depth-first search properties, it is not hard to see that, for any $e \in E(T)$, there are exactly two edges of H, say e_1, e_2, that *cover* e, i.e., $e \in E(P_{e_1})$ and $e \in E(P_{e_2})$.

Let us denote by Opt an optimal solution and by e^* an edge of Opt of maximum cost. As Opt minus e^* is a spanning tree of G, we can claim that $c(\text{Opt}) \geq c(T) + c(e^*)$. In the remaining part of this section we will prove that $c(H) \leq 2\beta\big(c(T) + c(e^*)\big)$, thus proving that $c(H) \leq 2\beta\, c(\text{Opt})$. The idea of the proof is to charge the cost of H to the edges in T plus e^* in such a way that each of these edges will be charged with at most 2β times its cost. This can be done in a simple way because every edge f in H is associated with P_f, a path in T. Therefore, it is quite natural to start charging the cost of f to the edges of P_f according to the formula in Lemma 1. Let e be an edge of T, and let $e_1, e_2 \in E(H)$ be such that P_{e_1}, P_{e_2} cover e. Using the formula in Lemma 1, we can observe the following

– if $e \notin A$ then e is charged with at most 2β times its cost, as both P_{e_1}, P_{e_2} are different from e;
– if $e \in A$ then e is charged with at most $1 + \beta$ times its cost, as either P_{e_1} or P_{e_2} must be different from e.

Therefore, the unique problem to solve is how to uncharge edges in A and still keep every other edge with a charge of 2β times its cost. By the careful depth-first search, we know that A contains only edges of the form (u, v), where u is an internal node of T while v is a closest child of u. By definition of closest child, we have that $c(u,v) \leq c(u,v'), \forall (u,v') \in E(T_u)$. Moreover, it is possible to prove that $c(e) \leq c(e^*), \forall e \in E(T)$. Finally, in β-metric graphs, with $\frac{1}{2} \leq \beta < 1$, two adjacent edges f, f' satisfies $c(f) \leq \frac{\beta}{1-\beta}\, c(f')$ (see [2]). Since from Lemma 1, some edges may be charged with less than 2β times their cost, we can therefore uncharge (u, v) using the quantities we are saving on its adjacent edges in T plus e^* using the above rules. In what follows, we will prove that this can always be done in such a way that each edge of T plus e^* will be charged by at most 2β times its cost. The quality of the solution computed by `Refined Double-MST shortcut` is better than the one computed by `Double-MST shortcut` because, whenever an edge $(u, v') \in E(T_u)$ has been charged with $(2\beta - \delta)c(u, v')$, thanks to the careful depth-first search, we can uncharge (u, v) by $\delta\, c(u, v)$ instead of uncharging it by the smaller value $\frac{1-\beta}{\beta}\, \delta\, c(u, v)$. Before providing a formal proof of the above idea, we point out that the restriction of having a vertex of degree 1 as the root of T is only for the purpose of avoiding technicalities. All the results contained in this section can be extended to the general case in which T is rooted at any arbitrary vertex.

For any $e \in E(H)$, with $P_e = x_0 x_1 \ldots x_k$, we define V_e as the set of vertices $\{x_1, x_2, \ldots, x_{k-2}\}$, and $V' = \bigcup_{e \in E(H), |E(P_e)| \geq 3} V_e$. The following holds:

Lemma 3. *Let $e = (v_j, v_{j+1}) \in E(H)$ be such that $P_e = x_0 x_1 \ldots x_k$, with $x_0 = v_j$, $x_k = v_{j+1}$, and assume that $k \geq 3$. Then, for $1 \leq i \leq k-1$, x_i is the parent in T of x_{i-1}.*

Proof. Since P_e is a simple path in T, all the vertices in P_e but one, say x_h, have their parent in T contained in P_e. Suppose by contradiction that $h < k-1$. Then x_k is a descendent of x_{h+1} in T. It follows that x_{h+1} is visited after x_0 and before x_k. But this is not possible since $(x_0, x_k) \in E(H)$ implies that x_k is visited immediately after x_0. Therefore, it must be either $h = k-1$ or $h = k$. In both cases, this implies that x_{k-1} is the parent of x_{k-2} in T, which in its turn is the parent of x_{k-3} in T, and so on. From this, the claim follows. □

Let $\mathcal{L}$ denote the set of leaves of T. We have that

Lemma 4. $V' = V \setminus (\mathcal{L} \cup \{v_0, v_1\})$.

Proof. Lemma 3 implies that V' cannot contain any of the vertices in $\mathcal{L} \cup \{v_0, v_1\}$. Hence, in order to prove the claim, it is enough to show that any internal vertex v_j of T but v_0 and v_1 belongs to V_e, for some $e \in E(H)$. As v_j is an internal vertex and because of the depth-first search properties, there is an edge in H, say $e = (v_{h-1}, v_{h \bmod n})$, with $h-1 > j$, such that v_{h-1} is a proper descendent of v_j while $v_{h \bmod n}$ is not a descendent of v_j. Then, in order to conclude that $v_j \in V_e$, it suffices to observe that from the depth-first search properties, $v_{h \bmod n}$ cannot be the parent of v_j in T. □

From now on, we will denote by A' the set of edges in A but $(v_0, v_1), (v_1, v_2)$. Moreover, we define a function $\sigma : A' \to V'$ which maps each edge $e \in A'$ into the endpoint of e which is the parent in T of the other endpoint. From the above lemma and from Lemma 2, it is easy to derive that σ is a bijective function. We can now prove the following

Lemma 5. *For every $e \in B$ such that $P_e = x_0 x_1 \ldots x_k$ contains $k \geq 3$ edges, we have that*

$$\sum_{j=1}^{k-2} \left(\beta^j \sum_{i=0}^{k-1-j} c(x_i, x_{i+1}) \right) \geq \sum_{x \in V_e} c(\sigma^{-1}(x)). \tag{1}$$

Proof. Grouping together terms with respect to the edge costs, the left-hand side of (1) becomes

$$\sum_{j=1}^{k-2} \beta^j c(x_0, x_1) + \sum_{i=1}^{k-2} \sum_{j=1}^{k-1-i} \beta^j c(x_i, x_{i+1}). \tag{2}$$

By suitably rearranging its terms, (2) can be rewritten as

$$\sum_{i=1}^{k-2} \left(\sum_{j=1}^{k-1-i} \beta^j c(x_{i-1}, x_i) + \beta^{k-1-i} c(x_i, x_{i+1}) \right). \tag{3}$$

Now we bound the i-th term of the external summation in (3) with respect to the cost of edge $\sigma^{-1}(x_i)$. Recall that (see [2]) for any two adjacent edges e_1, e_2 of G, it is $c(e_1) \leq \frac{\beta}{1-\beta} c(e_2)$. Then, for each $x_i \in V_e$ it is $c(\sigma^{-1}(x_i)) \leq$

$\frac{\beta}{1-\beta} c(x_i, x_{i+1})$. Moreover, $c(\sigma^{-1}(x_i)) \leq c(x_{i-1}, x_i)$, since $\sigma^{-1}(x_i)$ and (x_{i-1}, x_i) are both edges of T_{x_i}. Then we have

$$\begin{aligned}(3) \geq& \sum_{i=1}^{k-2} \left(\sum_{j=1}^{k-1-i} \beta^j c(\sigma^{-1}(x_i)) + \beta^{k-2-i}(1-\beta)\, c(\sigma^{-1}(x_i)) \right) \\ =& \sum_{i=1}^{k-2} \left(\sum_{j=1}^{k-2-i} \beta^j + \beta^{k-2-i} \right) c(\sigma^{-1}(x_i)) \\ \geq& \sum_{i=1}^{k-2} \left(\sum_{j=1}^{k-2-i} \frac{1}{2^j} + \frac{1}{2^{k-2-i}} \right) c(\sigma^{-1}(x_i)) = \sum_{i=1}^{k-2} c(\sigma^{-1}(x_i)) = \sum_{x \in V_e} c(\sigma^{-1}(x))\end{aligned}$$

where the last inequality holds because $\beta \geq \frac{1}{2}$. □

We are now ready to give our main result:

Theorem 1. *The* `Refined Double-MST shortcut` *algorithm is a 2β-approximation algorithm for the β-MTSP.*

Proof. Let us start by setting $B = \bigcup_{2 \leq i \leq n} B_i$, where $B_i = \{e \in B \text{ s.t. } |E(P_e)| = i\}$. For any $e \in E(H)$, let $P_e = x_0^e x_1^e \dots x_k^e$. We can bound the total cost of H with

$$\begin{aligned}c(H) =& \sum_{e \in A} c(e) + \sum_{e \in B_2} c(e) + \sum_{k=3}^{n} \sum_{e \in B_k} c(e) \leq \sum_{e \in A} c(e) + \beta \sum_{e \in B_2} c(P_e) + \\ &+ \sum_{k=3}^{n} \sum_{e \in B_k} \left(\beta\, c(P_e) - (1-\beta) \sum_{j=1}^{k-2} \left(\beta^j \sum_{i=0}^{k-1-j} c(x_i^e, x_{i+1}^e) \right) \right) \\ \leq& \sum_{e \in A} c(e) + \beta \sum_{k=2}^{n} \sum_{e \in B_k} c(P_e) - (1-\beta) \sum_{k=3}^{n} \sum_{e \in B_k} \sum_{x \in V_e} c(\sigma^{-1}(x))\end{aligned}$$

where the first inequality holds from the β-metricity of G and from Lemma 1, while the last inequality holds from Lemma 5. Since each edge of T is covered by exactly two edges of H, then

$$\sum_{k=2}^{n} \sum_{e \in B_k} c(P_e) = \sum_{e \in A} c(e) + 2 \sum_{e \in E(T) \setminus A} c(e).$$

By definition, observe that each $e \in E(H)$ for which $V_e \subseteq V'$ is in B_j for some $j \geq 3$. From this fact and because σ is bijective we can derive

$$\sum_{k=3}^{n} \sum_{e \in B_k} \sum_{x \in V_e} c(\sigma^{-1}(x)) \geq \sum_{x \in V'} c(\sigma^{-1}(x)) = \sum_{f \in A'} c(f)$$

(in fact, here we could prove that equality holds, since $e \neq e' \Rightarrow V_e \cap V_{e'} = \emptyset$).

To conclude the proof, let e^* be a maximum-cost edge of any optimal solution Opt. Observe that, $c(e) \le c(e^*)$ for every $e \in E(T)$. Indeed, let us consider the cut induced by the removal of e from T, and let $e' \in E(H)$ be an edge traversing that cut. Then, since T is an MST, we have $c(e) \le c(e') \le c(e^*)$. Using previous observations we have

$$\begin{aligned}
c(H) &\le c(v_0,v_1) + c(v_1,v_2) + \sum_{e \in A'} c(e) \\
&\quad + \beta\left(c(v_0,v_1) + c(v_1,v_2) + \sum_{e\in A'} c(e) + 2\sum_{e\in E(T)\setminus A} c(e)\right) - (1-\beta)\sum_{e\in A'} c(e) \\
&= 2\beta\left(\sum_{e\in A'} c(e) + \sum_{e\in E(T)\setminus A} c(e)\right) + (1+\beta)\Big(c(v_0,v_1)+c(v_1,v_2)\Big) \\
&\le 2\beta\left(\sum_{e\in A'} c(e) + \sum_{e\in E(T)\setminus A} c(e)\right) + c(v_0,v_1)+c(v_1,v_2) + 2\beta\, c(e^*) \\
&\le 2\beta\, c(T) + 2\beta\, c(e^*) \le 2\beta\, c(\textsc{Opt})
\end{aligned}$$

where the last but two inequality holds because $c(e^*) \ge c(e)$, $\forall e \in E(T)$, the last but one inequality holds because $\beta \ge \frac{1}{2}$, and the last inequality follows from the fact that by removing e^* from Opt, one obtains a spanning tree. □

4 Lower Bound for `Double-MST Min-Weight Shortcut`

In the introduction we have mentioned one of the best heuristics available for the TSP, i.e., the heuristic we called `Double-MST Min-weight shortcut` [3,5]. We said that this heuristic computes in $O(4^d n^2)$ time (where d denotes the maximum node-degree in T) the minimum cost Hamiltonian cycle that can be obtained using the `Double-MST shortcut` approach. It is worth noticing that the result proved for the algorithm we proposed in Section 3 immediately implies that `Double-MST Min-weight shortcut` is a 2β-approximation algorithm. In this section we prove that `Double-MST Min-weight shortcut` cannot compute an approximate solution within a factor which is significantly better than 2β. As a consequence, 2β is an asymptotic lower bound for the approximation ratio of all the algorithms based on the `Double-MST shortcut` approach. Hence, the `Refined Double-MST shortcut` is one of the best approximation algorithms based on this approach.

Consider the β-metric graph of $n+1$ vertices given in Figure 3. All the internal vertices of the MST T represented with solid edges have degree $\sqrt{n}$. A feasible solution is given by the $\sqrt{n}$ dotted paths arbitrarily linked one another and with x to form a Hamiltonian cycle. As the cost of each dotted path is $\sqrt{n}-1$ and as the cost of any other edge is at most 2β, we have that the cost of an optimal solution Opt is upper bounded by $c(\textsc{Opt}) \le \sqrt{n}(\sqrt{n}-1) + 2\beta(\sqrt{n}+1) \le n + 2\beta\sqrt{n}$.

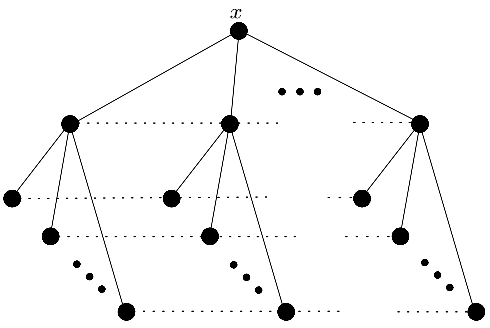


Fig. 3. A β-metric graph of $n+1$ vertices showing the asymptotic lower bound of 2β on the approximation ratio of `Double-MST Min-weight shortcut`. Both solid and dotted edges have cost 1, while the cost of missing edges is 2β. An MST T is given by the set of solid edges The degree of all the internal vertices of the MST is $\sqrt{n}$.

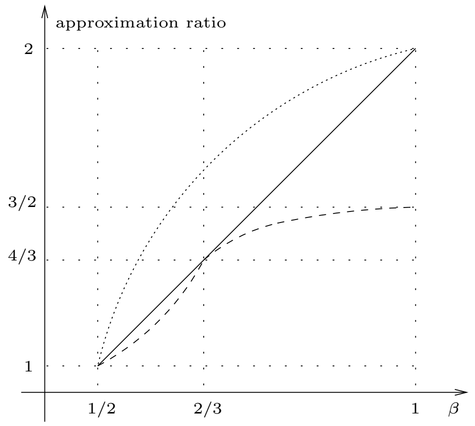


Fig. 4. The approximation ratio of our algorithm (solid line), as compared to that provided by the `Double-MST shortcut` (dotted) and the currently best-known approximation algorithm (dashed)

Concerning the solution built by `Double-MST Min-weight shortcut`, first of all notice that no Hamiltonian cycle contains more than $2\sqrt{n}$ solid edges, as these edges are all incident to the $\sqrt{n}$ internal vertices of T but x, and the degree of every vertex in a Hamiltonian cycle is 2. Now, observe that every dotted edge is a shortcut of some path in $D = T \cup T$ containing 2 solid edges which are incident to x. As the degree of x in D is $2\sqrt{n}$, we have that every solution built by `Double-MST Min-weight shortcut` contains at most $\sqrt{n}$ dotted edges. As a consequence, the solution computed by `Double-MST Min-weight shortcut` contains at least $n + 1 - 3\sqrt{n}$ edges of cost 2β. As $c(\text{OPT}) \leq n + 2\beta\sqrt{n}$, we therefore have that `Double-MST Min-weight shortcut` does not return a $(2\beta-\epsilon)$-approximate solution, for every $\epsilon > 0$.

5 Conclusion

In Figure 4, we provide a comparison between the approximation algorithms for the β-MTSP discussed in this paper, namely the `Double-MST shortcut`, that obtained by composing the algorithms given in [4,2], and finally our one. It is worth noticing that our algorithm induces a significant improvement in the gap with respect to the superlinear approximation algorithm: for instance, for all $1/2 < \beta \leq 3/4$, our algorithm is only about 5% away from it in the worst case, while for the `Double-MST shortcut` this gap raises to about 26%. Thus, especially for this range of values of β, from a practical point of view one can make use of our very simple linear-time algorithm—instead of pursuing a very complicate implementation of the superlinear algorithm—by only paying a little bit more in terms of approximation ratio.

References

1. Arora, S.: Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J. ACM 45(5), 753–782 (1998)
2. Böckenhauer, H.-J., Hromkovič, J., Klasing, R., Seibert, S., Unger, W.: Approximation algorithms for the TSP with sharpened triangle inequality. Information Processing Letters 75, 133–138 (2000)
3. Burkard, R.E., Deineko, V.G., Woeginger, G.J.: The travelling salesman and the pq-tree. Mathematics of Operations Research 23(3), 613–623 (1998)
4. Christofides, N.: Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegy–Mellon University (1976)
5. Deineko, V.G., Tiskin, A.: Fast minimum-weight double-tree shortcutting for metric TSP. In: Demetrescu, C. (ed.) WEA 2007. LNCS, vol. 4525, pp. 136–149. Springer, Heidelberg (2007)
6. Drake Vinkemeier, D.E., Hougardy, S.: A linear-time approximation algorithm for weighted matchings in graphs. ACM Transactions on Algorithms 1(1), 107–122 (2005)
7. Gabow, H.N.: An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In: Proc. of the 15th Annual ACM Symp. on Theory of Computing (STOC), pp. 448–456 (1983)
8. Gabow, H.N., Tarjan, R.E.: Faster scaling algorithms for general graph-matching problems. J. ACM 38(4), 815–853 (1991)
9. Papadimitriou, C.H., Vempala, S.: On the approximability of the traveling salesman problem. Combinatorica 26(1), 101–120 (2006)
10. Reinelt, G.: TSPLIB library, University of Heidelberg, `http://www.iwr.uni-heidelberg.de/groups/comopt/software/tsplib95/`
11. Tan, X.: Approximation algorithms for the watchman route and zookeeper's problems. Discrete Applied Mathematics 136(2-3), 363–376 (2004)

The Valve Location Problem in Simple Network Topologies

Hans L. Bodlaender[1], Alexander Grigoriev[2], Nadejda V. Grigorieva[3], and Albert Hendriks[1]

[1] Institute of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands
hansb@cs.uu.nl, alberthendriks@gmail.com

[2] Department of Quantitative Economics, Maastricht University, P.O. Box 616, 6200 MD Maastricht, The Netherlands
a.grigoriev@ke.unimaas.nl

[3] Institute of Power Resources Transport (IPTER), 144/3, pr. Octyabrya, Ufa-450055, Russia
n_grigorieva@yahoo.com

Abstract. To control possible spills in liquid or gas transporting pipe systems, the systems are usually equipped with shutoff valves. In case of an accidental leak these valves separate the system into a number of pieces limiting the spill effect. In this paper, we consider the problem, for a given edge-weighted network representing a pipe system and for a given number of valves, to place the valves in the network in such a way that the maximum possible spill, i.e. the maximum total weight of a piece, is minimized. We show that the problem is NP-hard even if restricted to any of the following settings: (i) for series-parallel graphs and hence for graphs of treewidth two; (ii) if all edge weights equal one. If the network is a simple path, a cycle, or a tree, the problem can be solved in polynomial time. We also give a pseudo-polynomial time algorithm and a fully polynomial approximation scheme for networks of bounded treewidth.

Keywords: Valve location problem; computational complexity; bounded treewidth; dynamic programming; binary search.

1 Introduction

In this paper, we consider a combinatorial problem that arose from a number of applications connected to operations and maintenance of a broad variety of transportation systems; for applications related to the long oil and gas pipelines see e.g. [10]; for applications in water supply engineering see [15]; for applications in electrical grid maintenance see [7]. Let us briefly discuss the related problem arising in oil and gas transportation. A pipeline is the most efficient and environmentally friendly way to transport hazardous liquids and gases, e.g. crude oil or natural gas, over land. In normal daily operations, pipelines do not produce any pollution. However, due to external factors or pipe corrosion, accidents on pipelines sometimes happen and the accidental damage can be substantial. To control possible spills, every pipe system is usually equipped with special shutoff valves. Whenever the pipe system is depressurized, the valves automatically and instantly separate the pipe system into *pieces*. Therefore, the quantity of hazardous liquid or gas potentially leaving the system equals the total length of the pipes in the damaged piece of the system separated by shutoff valves. In the application at hand, there is a given edge-weighted network representing a pipe system and a given number of valves that can be placed in the vertices of the network. We want to solve the following problem: find a valve location in the network that minimizes the maximum total weight of a piece separated by shutoff valves.

This paper is organized as follows. In Section 2, we give a precise graph theoretic formulation of the problem. In Section 3, we show that using dynamic programming the problem can be solved in polynomial time on simple network topologies: paths, cycles and trees. In Section 4, we consider a more general case, namely the graphs of bounded treewidth. For these graphs, we give a pseudo-polynomial time dynamic programming algorithm, and then we turn this algorithm into a fully polynomial approximation scheme (FPTAS). Finally, in Section 5, we discuss the complexity of the problem. Here, we show that the problem is NP-hard even for series-parallel graphs and hence for graphs of treewidth at most two. We also show that the unweighted version of the problem, i.e. the problem where all edge weights equal one, is also NP-hard.

2 Graph Theoretic Formulation

The problem can be formulated in graph theoretic terms in a natural way. Let $G = (V, E)$ be an undirected graph representing a pipe network. Edges of the graph represent pipes. Let $\omega_e \in \mathbb{Z}^+$ denote the length of pipe $e \in E$. Vertices of the graph represent connection points between the pipes. Let k be a number of valves to be installed. We assume that a valve can be located in any vertex $v \in V$.

Consider a set of vertices $V' \subseteq V$. If we use V' as valve locations, we use $|V'|$ valves, and partition G into *pieces* as follows. The set of edges E is partitioned into sets with two edges in the same set of the partition if and only if they are on a path in G that does not contain a valve. Thus, E is partitioned into subsets $E_1, E_2, \ldots, E_S$ where edges in E_s, $1 \le s \le S$, form a connected component in G called a piece, and for any two subsets, E_s and E_t, the set of endpoints in E_s intersects the set of endpoints in E_t only in elements of V'. The *cost* of V', denoted $W_{\max}(V')$, is

$$W_{\max}(V') = \max_{1 \le s \le S} \sum_{e \in E_s} \omega_e,$$

i.e., the maximum total length of a piece or the maximum *spill*.

The VALVE LOCATION problem then is to find a subset V' of vertices in G such that $|V'| \le k$ and the cost $W_{\max}(V')$ is minimized. In other words, we have to find a k-elementary separator in G such that the maximum length connected component is minimized. We consider also the unweighted version of the problem where $\omega_e = 1$ for all $e \in E$.

Throughout the paper, n denotes the number of vertices in G, $\omega_{\max}$ the maximum length of an edge:

$$\omega_{\max} = \max_{e \in E} \omega_e,$$

and ω_Σ the total length of all edges:

$$\omega_\Sigma = \sum_{e \in E} \omega_e.$$

Clearly, the maximum spill is yet another network vulnerability measure. This concept is very close to many other known vulnerability measures, e.g. *vertex integrity* of a graph defined as $I(G) = \min\{|S| + m(G - S) : \ S \subset V\}$, where $m(H)$ denotes the maximum order of a component of H, see [2,3]; *minimum balanced separator* defined as a minimum order separator S such that the maximum component in $G - S$ contains at most βn vertices for a given $0 < \beta < 1$, see [1,8,14]; and some other, see e.g. [13]. The key difference between the maximum spill and the known vulnerability measures is that the maximum spill measures vulnerability of a graph in terms of the total edge weight (or length) of a component when all other measures are related to the maximum order (number of vertices) in a component. Of course, practical suitability of a certain measure depends heavily on applications.

Throughout the paper, we measure run time of the algorithms using the widely accepted convention that we can do an addition or multiplication of two integers in $O(1)$ time. If we want to count bit operations, we must multiply run times by a factor $\log \omega_\Sigma$.

3 Simple Networks: Paths, Cycles and Trees

In this section, we give dynamic programming algorithms to solve the problem in simple network topologies: paths, cycles and trees.

3.1 The Valve Location Problem on a Path

We first consider the VALVE LOCATION problem on a path. This simple case appears frequently in the practical settings of the long oil pipelines, and thus is of practical relevance; see [10]. We have two different exact algorithms. One uses 'text book' dynamic programming.

Proposition 1. *The* VALVE LOCATION *problem on a path can be solved by dynamic programming in $O(kn^2)$ time.*

The other algorithm is obtained by using a binary search for the optimal spill value and by checking feasibility of each spill value with a greedy algorithm.

Proposition 2. *Given a path and a value L, we can decide in $O(n)$ time if there is a solution to the* VALVE LOCATION *problem with k valves with cost at most L.*

The same idea also gives the minimum number of valves needed to guarantee a cost that is at most L. Using binary search for the optimal value in the range of integers between 0 and ω_Σ, we directly obtain the following result.

Corollary 1. *For a given path, we can solve the* VALVE LOCATION *problem in $O(n \log \omega_\Sigma)$ time.*

It is also possible to construct a fast 2-approximation algorithm using a greedy strategy for paths.

Proposition 3. *The* VALVE LOCATION *problem on a path admits a 2-approximation algorithm that uses $O(n)$ time.*

Finally, we can sharpen Proposition 3 when $\omega_{\max} \geq 3A_p$.

Proposition 4. *Consider the* VALVE LOCATION *problem on a path. Let k be the number of valves. If $\omega_{\max} \geq 3\omega_\Sigma/(k+1)$, then the optimal solution has cost $\omega_{\max}$. An optimal solution can be found in this case in $O(n)$ time.*

3.2 Cycles

If G is a cycle, then we can obtain exact and approximate solutions for the VALVE LOCATION by using variants to the algorithms for paths.

Proposition 5. *The* VALVE LOCATION *problem on a cycle admits a 2-approximation algorithm that uses $O(n)$ time.*

Proposition 6. *Consider the* VALVE LOCATION *problem on a cycle. Let k be the number of valves. If $\omega_{\max} \geq 3\omega_\Sigma/k$, then the optimal spill equals $\omega_{\max}$. An optimal valve location can be found in this case in $O(n)$ time.*

Theorem 1. *The* VALVE LOCATION *problem on a cycle can be solved by solving $O(n/k)$* VALVE LOCATION *problems on paths of length at most n.*

Corollary 2. *The* VALVE LOCATION *problem on a cycle can be solved in $O(n \min \{\log \omega_\Sigma, n/k\})$ time.*

3.3 Trees

Very recently, see [7], we became aware of the fact that the VALVE LOCATION problem on trees is an important modern research topic in electrical engineering. Whenever a regional power supply network (a tree) should be maintained, the engineers are shutting down some small subtree and they are interested in an optimal location of switchers. This application brings us to the VALVE LOCATION in trees. Surprisingly enough, already this special case of the problem is quite complicated: according to [7], a typical modern approach to the problem is a genetic algorithm. In this section, we present a nontrivial algorithm that solves the problem on trees in polynomial time. More specifically, we show:

Theorem 2. *The* VALVE LOCATION *problem on a tree can be solved in* $O(nk^2 \log(n\omega_{\max}))$ *time.*

The global structure of the algorithm is a binary search on the optimal value in the range of integers between 0 and $n\omega_{\max}$. Thus, we directly obtain Theorem 2 as a corollary of the next result.

Proposition 7. *Given a tree, and an integer* L*, we can decide in* $O(nk^2)$ *time if we can place* k *valves with maximum piece size at most* L*.*

Proof. We choose an arbitrary vertex v_r as root of the tree. For rooted subtrees T', and integers i, $0 \leq i \leq k$, we define

$A_{T',L}(i)$ = the minimum over all possible ways to put at most i valves in T' such that no piece in T' has a total length of more than L, of the total length of the piece that contains the root node of T'.

$A_{T',L}(i) = 0$, if there is a way to put at most i valves in T' such that no piece in T' has a total length of more than L, such that there is a valve in the root node of T'.

$A_{T',L}(i) = \infty$, if there is no possible way to put at most i valves in T' such that no piece in T' has a total length of more than L.

$P_{T',L}(i) =$ **true** if and only if $A_{T',L}(i) = 0$, i.e. if we can put at most i valves in T' such that no piece in T' has a total length of more than L, such that there is a valve in the root node of T'.

We will compute tables $A_{T',L}$ and $P_{T',L}$ for several subtrees of T:

- For each vertex v in T except v_r, we compute a table for the subtree, consisting of the parent of v in T, v, and all the descendants of v. The root of this subtree is the parent of v. Call this subtree T_v^+.
- For each vertex v in T: if v has i children $w_1, w_2, \ldots, w_i$, then for each j, $0 \leq j \leq i$, we compute a table for the subtree, consisting of v, $w_1, \ldots, w_j$, and all descendants of $w_1, w_2, \ldots, w_j$. Vertex v is the root of this subtree. Call this subtree $T_{v,j}$. For the case $j = i$, write $T_v = T_{v,i}$; this is the tree consisting of v and all its descendants.

The following two lemmas give recursive formulations that show how to compute these tables.

Lemma 1. *Let* T *be obtained by taking the union of trees* T' *and* T'' *such that the root* r *of* T' *and* T'' *is the only vertex that belongs to both trees. Let* $0 \leq i \leq k$*.*

1. $P_{T,L}(i)$*, if and only if there are* i'*,* i'' *with* $i' + i'' = i - 1$*,* $0 \leq i' \leq k$*,* $0 \leq i'' \leq k$*, such that* $P_{T',L}(i')$ *and* $P_{T'',L}(i'')$*.*
2. *If* $P_{T,L}(i)$*, then* $A_{T,L}(i) = 0$*.*

3. *If not $P_{T,L}(i)$, then*

$$A_{T,L}(i) = \min_{i',i'',i'+i''=i,0\leq i,i'} A_{T',L}(i') + A_{T'',L}(i'')$$

if this term is at most L, otherwise $A_{T,L}(i) = \infty$.

Lemma 2. *Let T be a tree with root r, and let T^+ be obtained by adding an edge $\{r, r'\}$ to a new vertex r' with length ℓ. Let r' be the root of T^+. Let $0 \leq i \leq k$, and L be an integer.*

1. *$P_{T^+,L}(i)$ holds if and only if $i > 0$ and $A_{T,L}(i-1) + \ell \leq L$.*
2. *If $P_{T^+,L}(i)$, then $A_{T^+,L}(i) = 0$.*
3. *If not $P_{T^+,L}(i)$, then $A_{T^+,L}(i) = A_{T^+,L}(i) + \ell$, if this term is at most L, and $A_{T^+,L}(i) = \infty$ otherwise.*

Using Lemmas 1 and 2, we can compute all desired tables. Recall that L is fixed during the computation. Now, for all vertices in the tree, in postorder, we compute all values $A_{T_v,L}(i)$, and $P_{T_v,L}(i)$, for all i, $0 \leq i \leq k$. This is done in the following way. If v is a leaf of T, then computing these values is trivial. Otherwise, suppose v has s children, say $w_1, w_2, \ldots, w_s$. For all j, $1 \leq j \leq s$, we compute all values $A_{T_{v,j},L}(i)$, and $P_{T_{v,j},L}(i)$ for all i, $0 \leq i \leq k$. In case $j = 1$, we note that $T_{v,1}$ is the same subtree as $T^+_{w_1}$. Thus, using Lemma 2, we can compute the values $A_{T_{v,1},L}(i)$, and $P_{T_{v,1},L}(i)$ from the already earlier computed tables $A_{T_{w_1},L}$ and $P_{T_{w_1},L}$. For $2 \leq j \leq s$, we note that $T_{v,j}$ is the union of $T_{v,j-1}$ and $T^+_{w_j}$. Thus, we first compute the tables $A_{T^+_{w_j}}, L$ and $P_{T^+_{w_j}}, L$ given the tables $A_{T_{w_j},L}$ and $P_{T_{w_j},L}$ using Lemma 2. Then, we compute the tables $A_{T_{v,j},L}$ and $P_{T_{v,j},L}$ from the tables $A_{T^+_{w_j},L}$, $P_{T^+_{w_j},L}$, $A_{T_{v,j-1},L}$ and $P_{T_{v,j-1},L}$, using Lemma 1. Finally, note that $T_{v,s} = T_v$. When we have the tables $A_{T_{v_r},L}$ and $P_{T_{v_r},L}$, we can easily decide whether we can place k valves in T with maximum piece size at most L, using the following simple observation.

Proposition 8. *Let v_r be the root of T. There is a solution to the* VALVE LOCATION *problem with k valves and cost at most L if and only if $A_{T_{v_r},L}(k) < \infty$.*

If T has n vertices, then we compute $O(n)$ tables: $O(1)$ per edge in T. Each table can be computed in $O(k^2)$ time. This can be easily observed from Lemmas 1 and 2. Simply, iterate over all possible values of k and k', and compute the necessary value of k''. Each step involves $O(1)$ computations. Actually, the step that uses Lemma 2 needs only $O(k)$ time. This finishes the proof of Proposition 7. □

Notice that for trees, a result similar to Propositions 4 and 6 holds. However, in this case, this does not lead to an approximation algorithm with constant performance guarantee.

Proposition 9. *Consider the* VALVE LOCATION *problem on a tree. Let k be the number of valves. If $\omega_{\max} \geq 3\omega_{\Sigma}/k$, then the optimal spill equals $\omega_{\max}$.*

4 Algorithms for Graphs of Bounded Treewidth

In practice, most of the transportation systems are more complicated than trees. This makes the problem more difficult from algorithmic perspective. Fortunately, real-life networks in majority of applications (e.g., for oil and gas pipeline transportation) are outerplanar or, taking this more generally, the corresponding graphs have bounded treewidth; see e.g. [5,6,11]. For this type of networks we have the following results.

Theorem 3. *The* VALVE LOCATION *problem on graphs of treewidth q admits a dynamic programming algorithm running in time $(n\omega_{\max})^{O(q)}$.*

This dynamic programming algorithm follows the lines of several algorithms for other problems on graphs of bounded treewidth. For easier description, we use a *nice* tree decomposition of width at most q; for definition see below.

As a first step, we must find a tree decomposition of width at most q. This can be done in $O(n)$ time for fixed q; see [4]. At this point, we would like to make a remark concerning practical implementations. The algorithm in [4] has such a large hidden constant, that it is not of use in a practical setting. Fortunately, there are several heuristics that often give good bounds. Also, there are fast algorithms that construct tree decompositions of optimal width for graphs of treewidth at most three (including outerplanar graphs), see e.g. [6] for a discussion.

Given a tree decomposition, in $O(n)$ time one can transform it to a *nice* tree decomposition [12] with the same width. We now give the definition of a nice tree decomposition.

A *nice tree decomposition* of a graph $G = (V, E)$ is a rooted binary tree $T = (I, F)$, where each node $i \in I$ is a subset $X_i \subseteq V$, called *bag*, such that

1. $\bigcup_{i \in I} X_i = V$.
2. For all $\{v, w\} \in E$, there exists an $i \in I$, with $v, w \in X_i$.
3. For all $v \in V$, the set $\{i \in I \mid v \in X_i\}$ forms a subtree of T.
4. If $i \in I$ has two children j_1, j_2, then $X_i = X_{j_1} = X_{j_2}$ (JOIN NODE).
5. If $i \in I$ has one child j, then either there is a $v \in X_i$ with $X_j \cup \{v\} = X_i$ (INTRODUCE NODE) or there is a $v \in X_j$ with $X_i \cup \{v\} = X_j$ (FORGET NODE).
6. If $i \in I$ is a leaf in T, then $|X_i| = 1$ (LEAF NODE).

The *width* of a nice tree decomposition is $\max_{i \in I} |X_i| - 1$.

In our dynamic programming algorithm, we compute in postorder for each node of T a table. Associate to node $i \in I$ the subgraph $G_i = G[V_i]$, induced by the set of vertices in X_i or a bag X_j with j a descendant of i: $V_i = \bigcup X_j$, with the union taken over all j in the subtree of T rooted at j.

A placement of valves on the vertices of G_i has a *characteristic*, which is a 5-tuple $(j, Z, L, f, \sim)$, consisting of

- The number j of used valves in G_i.
- The subset $Z \subset X_i$ of the vertices in X_i that contain a valve.
- The maximum length of a piece in G_i.
- A function $f : X_i \to \mathbf{N}$, giving for each vertex $v \in V_i$ the total length of the piece that contains v; if there is a valve on v, then $f(v) = 0$.
- An equivalence relation $\sim$ on X_i, with for all $v, w \in X_i$, $v \sim w$, if and only if there is a path from v to w in G_i that does not contain a vertex with a valve.

In the table of i, we store all possible characteristics of all placements of valves in G_i. Note that in this way, tables have a size that is bounded by $(n\omega_{\max})^{O(q)}$.

A somewhat tedious case analysis, typical for dynamic programming algorithms on graphs of bounded treewidth, shows that we can compute for each of the four types of nodes the table of all characteristics for a node, given such a table for each of the children of the node, in time polynomial in the table size.

Then, computing these tables for all nodes in postorder gives an algorithm computing the table for the root node, and as G_r for the root node r equals G, we obtain the optimal valve location from this table.

We remark that the described dynamic programming is only a pseudo-polynomial time algorithm for the weighted version of the VALVE LOCATION problem on graphs of bounded treewidth. Using standard scaling arguments, we derive the following corollary.

Corollary 3. *The* VALVE LOCATION *problem on graphs of bounded treewidth admits a fully polynomial approximation scheme.*

5 Complexity Results

In this section, we show that two restricted versions of the VALVE LOCATION problem are NP-hard. For general networks it is strongly NP-hard as even the unweighted version of the problem is NP-hard, while for series-parallel graphs (a special case of graphs of treewidth at most two) the problem is weakly NP-hard. Note that this complements the result that the problem is solvable in pseudo-polynomial time on graphs of bounded treewidth.

Theorem 4. *The* VALVE LOCATION *problem is NP-hard even if $\omega_e = 1$ for all $e \in E$.*

The proof of Theorem 4 is quite straightforward and it is based on a reduction from the strongly NP-hard problem 3-PARTITION.

The second complexity result is less trivial. For this result let us remind a definition of a series-parallel graph. A *series-parallel graph* is a graph $G = (V, E)$ with two special vertices, called its terminals, often denoted s and t, that can be formed with the following operations:

- A graph consisting of a single edge $\{s, t\}$ between its terminals is a series-parallel graph.

- If G and H are terminal graphs, with terminals s_G, t_G, and s_H and t_H, then the *series composition* of G and H is a series-parallel graph. In the series composition, we take the disjoint union, then identify t_G and s_H, and take s_G and t_H as terminals of the resulting graph.
- If G and H are terminal graphs, with terminals s_G, t_G, and s_H and t_H, then the *parallel composition* of G and H is a series-parallel graph. In the parallel composition, we take the disjoint union, then identify s_G and s_H and identify t_G and t_H. The two vertices obtained by identification are the terminals of the resulting graph.

Theorem 5. *The* VALVE LOCATION *problem is weakly NP-hard for series-parallel graphs.*

Proof. We show that the VALVE LOCATION problem is weakly NP-hard for the following graphs: we have two vertices s and t, and a number of internally disjoint paths from s to t of length exactly five.

We use a reduction from PARTITION; see e.g. [9]. Suppose we are given positive integers $a_1, a_2, \ldots, a_n$. The PARTITION problem asks if these integers can be partitioned into two sets with equal sum, i.e. we look for two sets, each of sum $B = \sum_{i=1}^{n} a_i/2$. We may assume B is integer, as if $\sum_{i=1}^{n} a_i$ is odd, the PARTITION problem trivially has no solution.

As the corresponding instance for the VALVE LOCATION problem, we take n disjoint paths from s to t. Each of these paths has length five, i.e., four intermediate vertices, which we call $v_{i,1}, v_{i,2}, \ldots, v_{i,4}$. The successive lengths of the edges on the ith path are 1, a_i, $B - a_i + n$, a_i, 1. Call the resulting graph G, see Figure 1.

Proposition 10. *Set $A = \{a_1, a_2, \ldots, a_n\}$ can be partitioned into two sets, both of sum B, if and only if we can place at most $2n$ valves in G such that each part has total length at most $B + n$.*

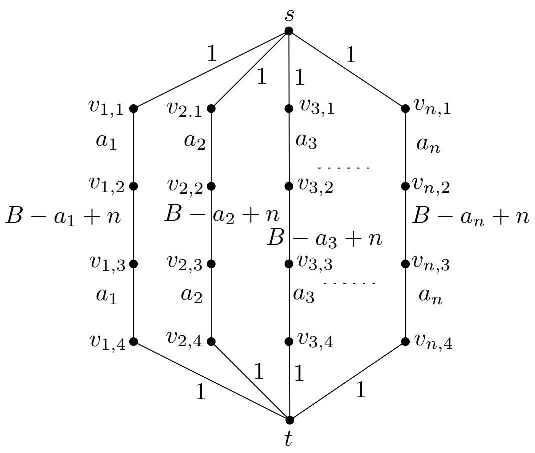


Fig. 1. The series-parallel graph constructed in Theorem 5

The NP-hardness of the VALVE LOCATION problem on series-parallel graphs now follows, by noting that G is series-parallel: a path can be constructed by a sequence of series compositions, and by parallel compositions, we can identify the endpoints of the paths. □

As series-parallel graphs have treewidth two, the results of the previous section show that (unless P=NP), the problem on series-parallel graphs cannot be strongly NP-hard. Moreover, Theorem 5 excludes the possibility for fixed-parameter tractability of the problem with respect to the parameter "graph treewidth".

6 Conclusions

In this paper we presented fast algorithms for several practically relevant classes of instances of the VALVE LOCATION problem. Moreover, applying literally the same techniques to the vertex integrity problem, we can tackle this later problem as well.

Acknowledgments

We thank Alexandr Kostochka for pointing on several very useful references on graph integrity.

References

1. Alon, N., Seymour, P., Thomas, R.: A separator theorem for graphs with an excluded minor and its applications. In: Proc. of the 22nd Symposium on Theory of Computing, STOC 1980, pp. 293–299. ACM Press, New York (1980)
2. Barefoot, C.A., Entringer, R., Swart, H.C.: Vulnerability in graphs: a comparative survey. J. Comb. Math. Comb. Comput. 1, 12–22 (1987)
3. Barefoot, C.A., Entringer, R., Swart, H.C.: Integrity of trees and the diameter of a graphs. Congressus Numerantium 58, 103–114 (1987)
4. Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput. 25, 1305–1317 (1996)
5. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theor. Comp. Sc. 209, 1–45 (1998)
6. Bodlaender, H.L.: Treewidth: Characterizations, applications, and computations. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 1–14. Springer, Heidelberg (2006)
7. Bouwman, S.: A survey of OR models and techniques for electrical grid companies. In: Proceedings of the 33rd Conference on the Mathematics of Operations Research, Lunteren 2008, Landelijk Netwerk Mathematische Besliskunde, The Netherlands (2008)
8. Feige, U., Mahdian, M.: Finding small balanced separators. In: Proceedings of the 37th Annual Symposium on Theory of Computing, STOC 2006, pp. 375–384. ACM Press, New York (2006)

9. Garey, M.R., Johnson, D.S.: Computers and intractability: A guide to the theory of NP-completeness. W. H. Freeman, San Francisco (1979)
10. Grigorieva, N.V., Grigoriev, A.: Optimal valve location in long oil pipelines. Research Memorandum RM/07/007, Maastricht Research School of Economics of Technology and Organizations (METEOR), Maastricht University, Maastricht, The Netherlands (2007), `http://ideas.repec.org/p/dgr/umamet/2007007.html`
11. Hicks, I.V., Koster, A.M.C.A., Kolotoglu, E.: Branch and tree decomposition techniques for discrete optimization. In: Smith, J.C. (ed.) TutORials 2005, INFORMS Tutorials in Operations Research Series, ch. 1, pp. 1–29. INFORMS Annual Meeting (2005)
12. Kloks, T.: Treewidth. Computations and Approximations. LNCS, vol. 842. Springer, Berlin (1994)
13. Kratsch, D., Kloks, T., Müller, H.: Measuring the vulnerability for classes of intersection graphs. Discrete Applied Mathematics 77(3), 259–270 (1997)
14. Marx, D.: Parameterized Graph Separation Problems. In: Downey, R.G., Fellows, M.R., Dehne, F. (eds.) IWPEC 2004. LNCS, vol. 3162, pp. 71–82. Springer, Heidelberg (2004)
15. Ozger, S., Mays, L.W.: Optimal location of isolation valves: A reliability approach. In: Water Supply Systems Security, Digital Engineering Library. McGraw-Hill, New York (2004), `http://dx.doi.org/10.1036/0071455663.CH13`

A 3/2-Approximation Algorithm for Finding Spanning Trees with Many Leaves in Cubic Graphs

Paul Bonsma[⋆] and Florian Zickfeld[⋆⋆]

Institut für Mathematik, Technische Universität Berlin,
Straße des 17. Juni 136, 10623 Berlin, Germany
{bonsma,zickfeld}@math.tu-berlin.de

Abstract. We consider the problem of finding a spanning tree that maximizes the number of leaves (MaxLeaf). We provide a 3/2-approximation algorithm for this problem when restricted to cubic graphs, improving on the previous 5/3-approximation for this class. To obtain this approximation we define a graph parameter $x(G)$, and construct a tree with at least $(n - x(G) + 4)/3$ leaves, and prove that no tree with more than $(n - x(G) + 2)/2$ leaves exists. In contrast to previous approximation algorithms for MaxLeaf, our algorithm works with *connected dominating sets* instead of constructing a tree directly. The algorithm also yields a 4/3-approximation for Minimum Connected Dominating Set in cubic graphs.

Keywords: approximation algorithm, maximum leaf, connected dominating set, cubic graph.

1 Introduction

The problem MaxLeaf is defined as follows: given a connected graph G, find a spanning tree of G that maximizes the number of leaves. This problem is NP-hard, even for cubic graphs [14]. *Cubic* graphs are graphs in which every vertex has degree 3. This problem is closely related to the problem MinCD-Set, which asks for a smallest possible connected dominating set or *CD-set*, which is a set $S \subseteq V(G)$ such that $G[S]$ is *connected*, and every vertex of G is either in S or adjacent to S (a *dominating set*). Observing that the non-leaves of a spanning tree form a CD-set, it is easily seen that G has a spanning tree with at least k leaves if and only if G has a CD-set of size at most $|V(G)| - k$ (provided that $G \neq K_2$), and that these can be constructed from each other in polynomial time.

These problems have many theoretical and practical applications; in particular, recently they have received a lot of attention due to their importance in wireless networks [3]. Therefore it is not surprising that they have been considered from many different viewpoints, such as purely combinatorial settings

[⋆] Supported by the Graduate School "Methods for Discrete Structures" in Berlin, DFG grant GRK 1408.

[⋆⋆] Supported by the Studienstiftung des deutschen Volkes.

(see below), and using most of the different algorithmic paradigms for solving hard problems, such as approximation algorithms (see below), fixed parameter tractable algorithms [2,7] and fast exact algorithms [8]. Generalizations such as to directed graphs have been studied [6]. Restrictions to different graph classes have also been considered. Motivated by the wireless networking applications, unit disk graphs have been widely studied [3]. In this paper, we study the two problems when restricted to cubic graphs, which was done before in [4,10,14,15]. Let $n(G)$ denote $|V(G)|$, and let $\delta(G)$ and $\Delta(G)$ denote the minimum and maximum degree of G, respectively. If there is no cause for confusion we will simply write n, δ and Δ.

With regard to the approximability of MaxLeaf, it is known that a polynomial time approximation scheme is unlikely to exist, since the problem is known to be MAX SNP-complete [9]. A 3-approximation was given by Lu and Ravi [16], and later a 2-approximation was given by Solis-Oba [18], which is the current best approximation ratio for general graphs. Loryś and Zwoźniak initiated the study of approximation algorithms for MaxLeaf in cubic graphs, and gave a 7/4-approximation for this class [15]. This was recently improved to 5/3 by Correa et al [4]. In this paper, we will give a 3/2-approximation for MaxLeaf for cubic graphs. From an approximation viewpoint MaxLeaf and MinCD-Set behave quite differently: Guha and Khuller [12] showed that it is unlikely that constant factor approximation algorithms exist for MinCD-Set for general graphs. The current best approximation is $2 + \ln \Delta(G)$, given by Ruan et al [17]. For cubic graphs our algorithm will approximate MinCD-Set with a guarantee of 4/3.

In another branch of research, a number of tight lower bounds is given for the maximum number of leaves that can be obtained, for (connected) graphs from different classes. Linial and Sturtevant first proved that every graph with $\delta \geq 3$ has a spanning tree with at least $n/4 + 2$ leaves (unpublished). A short proof appears in [13], where it is also shown that in graphs with $\delta \geq 4$, $2n/5 + 8/5$ leaves can be obtained. For graphs with $\delta \geq 5$, $n/2 + 2$ leaves are possible [11]. The $n/4+2$ bound is also tight for cubic graphs, but when in addition *diamonds* are forbidden as subgraphs, Griggs et al [10] showed that $n/3 + 4/3$ leaves can be obtained. A *diamond* is a K_4 minus one edge. Recently it was shown that when in addition to diamonds, a certain subgraph on seven vertices is forbidden, the $n/3 + 4/3$ bound also holds for graphs with $\delta \geq 3$ [2,19]. This generalizes an earlier result [1] that proves the same bound for graphs with $\delta \geq 3$ without triangles. However, because of some useful features (see Section 2), it is this earlier result that we will apply in this paper. In [19], a number of these bounds and similar bounds have been generalized to graphs with arbitrary degrees. Even though the algorithmic viewpoint is not stressed in the results mentioned above, all proofs easily give polynomial time algorithms that construct a tree satisfying the bounds.

This leads us back to approximation algorithms. For instance, note that the $n/4 + 2$ lower bound trivially gives a 4-approximation when MaxLeaf is restricted to graphs with $\delta \geq 3$. It is straightforward to show that spanning trees in cubic graphs have at most $n/2 + 1$ leaves. Combined with the $n/4 + 2$ lower bound, this then gives a 2-approximation for cubic graphs. (Note that these bounds in terms of CD-sets show that any CD-set in a cubic graph contains at least $n/2 - 1$ vertices, and that a CD-set with at most $3n/4 - 2$ vertices can be constructed. So for CD-sets in cubic graphs these bounds give a 3/2-approximation.) The 5/3-approximation given by Correa et al [4] is based on a more sophisticated version of this idea which we will now treat in more detail, since we use a similar strategy.

The goal of [4] was to match the upper bound $n/2 + 1$ for the number of leaves in cubic graphs with the lower bound $n/3 + 4/3$ for cubic graphs *without diamonds*. To make this work, only diamonds have to be treated in some way. This was done by defining a graph parameter $c(G)$ that depends on the number and positions of diamonds in G: for a subgraph H of G, the *internal vertices* of H are those with only neighbors in H. Note that diamonds in cubic graphs always have two internal vertices. The parameter $c(G)$ now denotes the number of components obtained when removing all internal vertices of diamonds from G. Using the bound from [10], it was shown that a spanning tree with at least $(3n - 2c(G) + 17)/10 > \frac{3}{10}(n - 2c(G) + 4)$ leaves can be constructed, and it was shown that any spanning tree has at most $(n - 2c(G) + 4)/2$ leaves. Together this gives a $\frac{1}{2}/\frac{3}{10} = \frac{5}{3}$ approximation. It was also conjectured in [4] that a 3/2-approximation algorithm is possible for MaxLeaf.

Our contribution. We prove this conjecture by providing a 3/2-approximation for MaxLeaf in cubic graphs. The algorithm itself is very simple, although the analysis is more involved. It is necessary to extend the study of problematic structures further, beyond only diamonds. This is indicated by the examples from [4]: there it is shown that graphs G with $c(G) = 0$ exist that have no spanning tree with more than $\lceil (3n + 17)/10 \rceil$ leaves, hence considering only the parameter $c(G)$ is not good enough. We consider all triangles of G, identify different types of them, and use their positions in G to define a number of graph parameters. This is done in Section 3. Then in Section 4, we state the algorithm, and prove it yields at least $(n - x(G) + 4)/3$ leaves, where $x(G)$ is a function of the defined graph parameters. In Section 5 we prove that spanning trees in cubic graphs have at most $(n - x(G) + 2)/2$ leaves. Together, this yields the 3/2-approximation for MaxLeaf, and the 4/3-approximation for MinCD-Set (see Section 6). We believe that the two bounds we prove, and the identified graph parameters are interesting in their own right, showing exactly in which cases diamonds and triangles make it hard or even impossible to find spanning trees with at least $n/3 + 4/3$ leaves in graphs with $\delta \geq 3$.

Unlike the previous algorithms, our algorithm is in fact an algorithm that constructs a CD-set instead of a tree. In [1] it was proved that in graphs with $\delta \geq 3$ without triangles, any CD-set S that satisfies some simple properties has $|S| \leq 2n/3 - 4/3$. We observe that this bound can be generalized to graphs with triangles, but with a correction term that (sloppily speaking) depends on the number of triangles in $G[S]$. Our algorithm constructs S such that it contains at most $x(G)$ triangles, such that the bound yields $|S| \leq (2n + x(G) - 4)/3$. This in turn gives the bound for spanning trees mentioned above. More details on the bounds for CD-sets are given in Section 2, together with some basic definitions. We end in Section 7 with a discussion. The easier proofs are omitted because of space constraints.

2 Preliminaries

Basic notations and terminology. For basic graph theoretic notions we refer to [5]. We assume all graphs to be simple (i.e. without parallel edges and loops), with the exception that we allow edge contractions to yield parallel edges and loops. When applying edge contractions, edges are assumed to be labeled, that is, edge identities are preserved even if the labels of their end vertices change.

The set resulting from removing element v from S is denoted by $S - v$, and adding an element is denoted by $S + v$. The number of components of a graph G is denoted by $cc(G)$. *Internal vertices* of a subgraph H of G are those vertices that only have neighbors in H. Let $\textsc{Int}(H)$ denote the set of internal vertices of H (with respect to its supergraph G). We say that a subgraph H of G is a *block* of G if it is a maximal 2-connected subgraph. The vertex degree of $v \in V(G)$ is denoted by $d_G(v)$, or $d(v)$ if possible.

Minimal CD-sets. A CD-set S is called a *2-CD-set* if every vertex in $V(G)\backslash S$ has at most two neighbors in S. Formulated just for cubic graphs, the results in [1] yield the following bound.

Theorem 1. *Let $G = (V, E)$ be a connected cubic graph. Let S be a minimal 2-CD-set of G where $G[S]$ contains no triangles, and let $S' \subseteq S$ be a minimal CD-set of G. Then $|S'| \leq (2n(G) - 4)/3$.*

This theorem can be applied to graphs without triangles, in that case it holds for any minimal 2-CD-set. But when considering the proof in [1], it can be seen that actually the following stronger statement is proved.

Theorem 2. *Let $G = (V, E)$ be a connected cubic graph. Let S be a minimal 2-CD-set of G where $G[S]$ has b^{Δ} blocks that contain triangles, and let $S' \subseteq S$ be a minimal CD-set of G. Then $|S'| \leq (2n(G) + b^{\Delta}(S') - 4)/3$.*

The strength of Theorem 2 lies not in the combination of the bound and the graph class itself (as we remarked in the introduction, in that sense it is strengthened and generalized for instance by the bound from [2]), but in the fact that it holds for *any minimal 2-CD-set*. This allows us to first construct a 2-CD-set S^* that satisfies some useful properties, namely that it contains few triangles, and then consider a minimal 2-CD-set $S \subseteq S^*$.

3 Subgraphs Obtained by Removing Triangles

The operations and notions defined in this section are illustrated in Figure 1. This figure introduces the example of G that we will use to illustrate most proofs

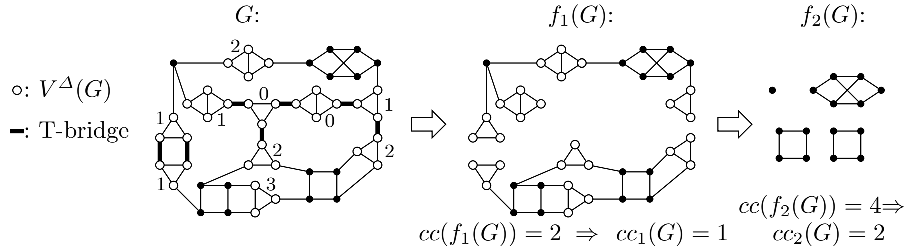

G:
$f_1(G)$:
$f_2(G)$:
○: $V^{\Delta}(G)$
▬: T-bridge
$cc(f_1(G)) = 2 \Rightarrow cc_1(G) = 1$
$cc(f_2(G)) = 4 \Rightarrow cc_2(G) = 2$

Fig. 1. An example of G, $f_1(G)$ and $f_2(G)$

in the paper. For a graph G with $\Delta(G) \leq 3$, let $V^{\Delta}(G) \subseteq V(G)$ be those vertices of G that are part of a triangle.

We distinguish a number of triangle types of G. First we distinguish between triangles of G that are part of diamonds, and those that are not. **From now on, when we talk about *triangles of* G, we mean those triangles that are not part of diamonds**, except when explicitly noted otherwise. Note that since $\Delta(G) \leq 3$, all diamonds of G are pairwise vertex disjoint, and all triangles of G are pairwise vertex disjoint (since they are not part of diamonds). We say that a triangle or diamond H of G is of *type* i if it is incident with i edges that have an end vertex not in $V^{\Delta}(G)$. So triangles can be of type i for $i \in \{0, 1, 2, 3\}$, and diamonds can be of type i for $i \in \{0, 1, 2\}$. Let $T_i(G)$ ($D_i(G)$) denote the number of triangles (diamonds) of type i in G. In Figure 1, the numbers next to the triangles and diamonds of G indicate their types.

An edge uv of G is a triangle bridge or *T-bridge* if $u, v \in V^{\Delta}(G)$ but u and v are not part of the same triangle or diamond. The graph $f_1(G)$ is obtained by deleting all vertices of G that are part of type 0 triangles or type 0 diamonds, and in addition deleting all T-bridges. The graph $f_2(G)$ is obtained from G by deleting all vertices in $V^{\Delta}(G)$, so $f_2(G)$ is a subgraph of $f_1(G)$.

Let $cc_1(G) = cc(f_1(G)) - cc(G)$, and let $cc_2(G) = cc(f_2(G)) - cc(f_1(G))$. We will always consider G to be connected, so $cc_1(G) \geq -1$, and this is only an equality when $V^{\Delta}(G) = V(G)$. Since every component of $f_1(G)$ contains a vertex not in $V^{\Delta}(G)$, we have $cc_2(G) \geq 0$. We remark that the graph parameter $x(G)$ that we mentioned in the introduction can now be defined as $x(G) = 2cc_1(G) + cc_2(G) + D_0(G) + T_0(G)$. If the graph in question is clear, we will also write cc_1 and T_0 etc. instead of $cc_1(G)$, $T_0(G)$, etc.

4 Constructing and Bounding the CD-Set

In this section we present our algorithm to construct a minimal 2-CD-set S, and prove an upper bound for the number of triangles in $G[S]$ (Theorem 3), which gives a suitable upper bound for the size of any minimal CD-set $S' \subseteq S$ using Theorem 2. The algorithm is shown in Algorithm 1. Note that the algorithm is non-deterministic in the sense that in every step many choices are possible. For any of these choices the bound we prove will hold.

Algorithm 1. An algorithm for finding small CD-sets in cubic graphs

```
INPUT: A connected, cubic graph $G$.
OUTPUT: A minimal CD-set $S'$ (which is a subset of a minimal 2-CD-set $S$)

(Stage 1:)
$S_1 := V(G)$.
while $\exists$ T-bridge $uv$ in $G[S_1]$ s.t. $S_1 - u - v$ is a CD-set of $G$ do
    $S_1 := S_1 - u - v$.
(Stage 2:)
$S_2 := S_1$.
repeat
    if $\exists$ type 3 triangle $T$ with $V(T) \subseteq S_2$ s.t. $S_2 \backslash V(T)$ is a CD-set of $G$ then
        $S_2 := S_2 \backslash V(T)$.
    if $\exists$ type 2 diamond $D$ with $V(D) \subseteq S_2$ s.t. $S_2 \backslash \textsc{Int}(D)$ is a CD-set of $G$ then
        $S_2 := S_2 \backslash \textsc{Int}(D)$.
until no change was made.
(Stage 3:)
$S := S_2$.
Remove vertices (possibly pairwise) from $S$ until it is a minimal 2-CD-set of $G$.
$S' := S$.
Remove vertices from $S'$ until it is a minimal CD-set of $G$.
```

The idea of the algorithm is that we try to minimize the number of triangles and diamonds that are present in the final CD-set S, since this makes the bound from Theorem 2 stronger. The stages reflect the priorities; a single change in Stage 1 can remove two triangles or diamonds from the current CD-set. In Stage 2, we can only remove one triangle or diamond per change. After these stages no further gain can be made, and we simply find minimal sets S and S' in order to apply Theorem 2.

Lemma 1. *Algorithm 1 has a polynomial time implementation, and yields a minimal 2-CD-set S of G and minimal CD-set $S' \subseteq S$ of G.*

Theorem 3. *Let S be a minimal 2-CD-set of G constructed by Algorithm 1. Then the number of triangles plus the number of diamonds in $G[S]$ is at most $2cc_1(G) + T_0(G) + D_0(G) + cc_2(G)$.*

Proof: First we analyze Stage 1. Let S_1 denote the set S_1 as it is when Stage 1 has finished. We will show that the number of triangles of type 0, 1 and 2 plus the number of diamonds of type 0 and 1 with all vertices still in S_1 is bounded by $2cc_1 + T_0 + D_0$. In particular, if $V^{\Delta} = V(G)$, the number of triangles and diamonds with all vertices in S_1 is bounded by $T_0 + D_0 - 2$.

The following definitions are illustrated in Figure 2. A T-bridge uv of G is called an S_1*-bridge* if $u, v \in S_1$. Let $n_B(S_1)$ denote the number of S_1-bridges in $G[S_1]$. A *bridge* is an edge for which the removal increases the number of components. We can observe the following.

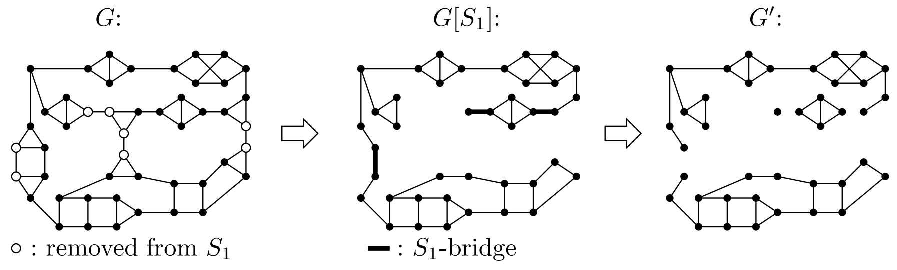


Fig. 2. The set S_1 after Stage 1 and G' obtained from $G[S_1]$ by deleting S_1-bridges

Claim 1. *An S_1-bridge uv is a bridge of $G[S_1]$.*

Let G' be the subgraph of $G[S_1]$ obtained by removing all S_1-bridges. The components of G' are of two types: those that are part of components of $f_1(G)$, and those that are part of the type 0 triangles and diamonds of G. Hence G' has $1+cc_1+T_0+D_0$ components. (Note that because of the way vertices are removed from S_1 during Stage 1, for every $f_1(G)$ component and every type 0 triangle or diamond, at least one vertex remains in S_1.) Since all S_1-bridges are bridges of $G[S_1]$ (Claim 1), contracting every edge of $G[S_1]$ that is not an S_1-bridge gives a tree on $cc(G')$ vertices, so the number of S_1-bridges is

$$n_B(S_1) = cc_1 + T_0 + D_0.$$

Let $T_{i,j}$ ($D_{i,j}$) denote the number of type i triangles (diamonds) of G that contain j vertices of S_1. By counting the number of S_1-bridges that every such triangle or diamond is incident with, we obtain

$$T_{2,3} + T_{1,2} + 2T_{1,3} + T_{0,1} + 2T_{0,2} + 3T_{0,3} + 2D_{0,4} + D_{0,3} + D_{1,4} \le 2n_B(S_1).$$

Using the above two inequalities, we can bound the number of type 0, 1, 2 triangles and type 0, 1 diamonds of G that are still fully part of $G[S_1]$:

$$T_{0,3} + T_{1,3} + T_{2,3} + D_{0,4} + D_{1,4} \le$$

$$2n_B(S_1) - 2T_{0,3} - T_{0,1} - 2T_{0,2} - T_{1,2} - T_{1,3} - D_{0,4} - D_{0,3} =$$

$$2cc_1 + 2T_0 + 2D_0 - 2T_{0,3} - T_{0,1} - 2T_{0,2} - T_{1,2} - T_{1,3} - D_{0,4} - D_{0,3} =$$

$$2cc_1 + T_0 + D_0 - T_{0,3} - T_{0,2} - T_{1,2} - T_{1,3} \le 2cc_1 + T_0 + D_0.$$

Here we used $T_0 = T_{0,1} + T_{0,2} + T_{0,3}$ and $D_0 = D_{0,3} + D_{0,4}$.

Now we will analyze Stage 2. Let S_2 denote the set S_2 as it is when Stage 2 has finished. Type 3 triangles T with $V(T) \subseteq S_2$ are called S_2*-triangles*, and type 2 diamonds D with $V(D) \subseteq S_2$ are called S_2*-diamonds*. We will show that the number of S_2-triangles plus the number of S_2-diamonds is at most cc_2.

The following arguments are illustrated in Figure 3 (which is not based on the graph G that we have used as example earlier). For a component C of $f_1(G)$, the number of components of $f_2(G)$ that are part of C is $cc_2(C) + 1$. We will first show that the number of S_2-triangles plus the number of S_2-diamonds in C is at most $cc_2(C)$. In the example of Figure 3, $cc(f_2(C)) = 5$, so $cc_2(C) = 4$, and there are two S_2-triangles and one S_2-diamond.

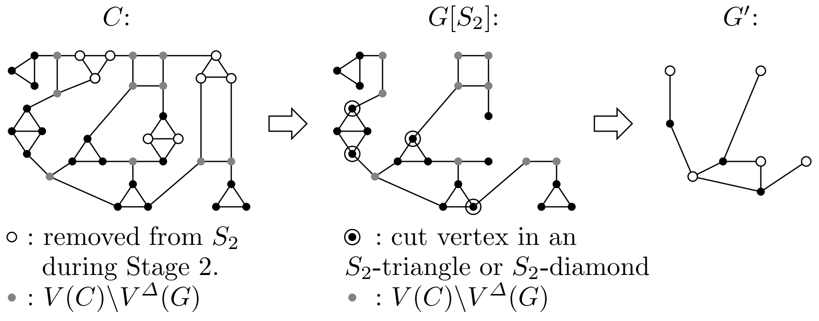

C:
G[S2]:
G':
○ : removed from S2 during Stage 2.
• : V(C)\V^Δ(G)
⦿ : cut vertex in an S2-triangle or S2-diamond
• : V(C)\V^Δ(G)

Fig. 3. A component C of $f_1(G)$, the set S_2 after Stage 2, and G'

Consider an S_2-triangle T in C. The set S_2 still contains all vertices of $V(G)\backslash V^{\Delta}$, so since T is of type 3, after removing $V(T)$ from S_2, the set would still be a dominating set of G. Therefore, since $V(T)$ was not removed during Stage 2, $G[S_2\backslash V(T)]$ is not connected. Since T is a triangle and G is cubic, this implies that $V(T)$ contains at least one cut vertex of C. Similarly, for S_2-diamonds it also holds that they contain a cut vertex of C (two actually).

Consider the graph G' defined as follows. For every S_2-triangle and every S_2-diamond in C, we add a black vertex to G', and for every $f_2(G)$ component in C we add a white vertex to G'. When an S_2-triangle or S_2-diamond is adjacent to an $f_2(G)$ component, we add an edge between the corresponding vertices. This gives a bipartite graph G' with $cc_2(C) + 1$ white vertices. Note that since every S_2-triangle and every S_2-diamond contains a cut vertex of C, all black vertices are cut vertices of G'. Using a simple induction argument it then follows that the number of black vertices of G' is at most the number of white vertices minus one, hence is bounded by $cc_2(C)$.

Every S_2-triangle and S_2-diamond of G is part of some component C of $f_1(G)$. In addition, if $\mathcal{C}$ is the set of components of $f_1(G)$, then $cc_2(G) = \sum_{C\in\mathcal{C}} cc_2(C)$. It follows that the number of type 3 triangles plus the number of type 2 diamonds in S_2 is at most $cc_2(G)$.

Since S is a subset of S_1 and of S_2, we know that the number of type 0, 1, 2 triangles plus the number of type 0, 1 diamonds of G that are fully in S is also bounded by $2cc_1 + T_0 + D_0$, and that the number of type 3 triangles plus the number of type 2 diamonds of G that are fully in S is bounded by cc_2. Now all types of triangles and diamonds of G have been considered, which proves the statement. □

The number of triangles and diamonds in $G[S]$ is an upper bound for the number of blocks of $G[S]$ that contain triangles (here we do also mean triangles that are part of diamonds): since S is a minimal 2-CD-set, it is not possible that a diamond of G is not fully part of S but three of its vertices that together form a triangle are. So by combining Lemma 1, Theorem 2 and Theorem 3 we obtain:

Theorem 4. *In polynomial time, Algorithm 1 returns a CD-set S' of G with $|S'| \le (2n(G) + 2cc_1(G) + cc_2(G) + T_0(G) + D_0(G) - 4)/3$.*

5 An Upper Bound for the Number of Leaves

We only have to prove the upper bound for trees of the following form.

Proposition 1. *A spanning tree T of G with maximum number of leaves exists that contains two edges of every type 0, 1 and 2 triangle of G, and contains either zero or two edges of every type 3 triangle of G.*

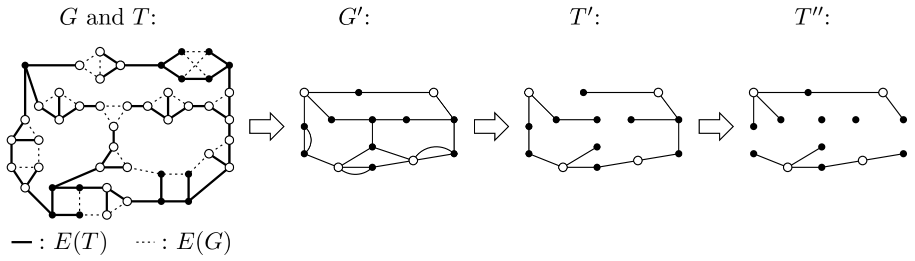


Fig. 4. The graphs G, T, G', T' and T'' from the proof of Theorem 5

Theorem 5. *Let T be a spanning tree of a cubic graph G. Then T has at most $(n(G) - 2cc_1(G) - cc_2(G) - D_0(G) - T_0(G) + 2)/2$ leaves.*

Proof: We will assume T has the properties stated in Proposition 1. The following constructions are illustrated in Figure 4. Let G' be the graph obtained from G by contracting every diamond and every triangle into a single *black* vertex, and by contracting every component of $f_2(G)$ into a single *white* vertex. Let B (W) denote the set of black (white) vertices of G'. We construct the following spanning subgraph T' of G'. An edge of G' is added to T' if and only if for the corresponding edge $e \in E(G)$:

- $e \in E(T)$, and
- e is not incident with a leaf of T that is part of a triangle or diamond.

Type 3 triangles that contain three leaves of T will correspond to isolated vertices v of T' at this point. To ensure that T' is connected, in addition we add one arbitrary edge of G' incident with v to T'. Using the assumptions from Proposition 1, the following claims about T' can be proved.

Claim 2. *T' is connected.*

Claim 3. *If a black vertex v has degree i in T', then the corresponding subgraph H_v of G contains at least $i-1$ vertices that have degree 2 in T.*

Since T' is connected (Claim 2), we have $|E(T')| \geq |V(T')| - 1 = |B| + |W| - 1$. Note that all edges of G' are incident with at least one black vertex. Consider the subgraph T'' of T' that has vertex set $V(G')$ again, but only contains those edges of T' that are incident with at least one white vertex. Note that $cc(T'') \geq D_0(G) + T_0(G) + cc_1 + 1$. When we add the edges of T' one by one until T' is obtained, clearly every edge addition can only decrease the number of components by at most one. Hence $cc(T'') - 1$ is a lower bound for the number of edges of T' that are incident with two black vertices. The degree sum of black vertices in T' is then at least the number of edges of T' plus the number of edges of T' incident with two black vertices. This yields the following bound.

$$\sum_{v\in B}(d_{T'}(v) - 1) \geq |E(T')| + (cc(T'') - 1) - |B| \geq$$

$$|B| + |W| - 1 + D_0 + T_0 + cc_1 - |B| = 2cc_1 + cc_2 + D_0 + T_0.$$

For the last equality we used $|W| = cc_1 + cc_2 + 1$. Since vertices of T' with degree 2 account for at least one vertex of degree 2 in T, and vertices of degree 3 account for at least two such vertices (Claim 3), the above number is also a lower bound for the number of degree 2 vertices in T.

Now let d_i denote the number of vertices of T with degree i, and $n = V(T)$. So $d_1 + d_3 = n - d_2$. For trees with $\Delta \leq 3$ it is easy to see that $d_3 = d_1 - 2$. This yields $2d_1 = n - d_2 + 2 \leq n - (2cc_1 + cc_2 + D_0 + T_0) + 2$, which gives the stated bound. □

6 The Approximation Guarantee

Theorem 6. *Algorithm 1 is a 4/3-approximation for* MinCD-Set *in cubic graphs, and gives a 3/2-approximation for* MaxLeaf *in cubic graphs.*

Proof: Let $x(G) = 2cc_1(G) + cc_2(G) + T_0(G) + D_0(G)$ and $n = n(G)$. Let S' be the minimal CD-set returned by the algorithm, and let S^* be a *minimum* CD-set of G. By Theorem 4, $|S'| \leq (2n + x(G) - 4)/3 \leq 2(n + x(G) - 2)/3$. By Theorem 5, any spanning tree of G has at most $(n - x(G) + 2)/2$ leaves, so any CD-set of G, in particular S^*, has $|S^*| \geq n - (n - x(G) + 2)/2 = (n + x(G) - 2)/2$. It follows that

$$|S'|/|S^*| \leq \frac{2(n + x(G) - 2)}{3} / \frac{(n + x(G) - 2)}{2} = 4/3.$$

Similarly, using S', a spanning tree T with $l^A \geq n - (2n + x(G) - 4)/3 = (n - x(G) + 4)/3$ leaves can easily be constructed in polynomial time. Since an optimal spanning tree has at most $l^* \leq (n - x(G) + 2)/2 < (n - x(G) + 4)/2$ leaves, the approximation guarantee for MaxLeaf is

$$l^*/l^A < \frac{(n - x(G) + 4)}{2} / \frac{(n - x(G) + 4)}{3} = 3/2.$$

7 Discussion

It would be interesting to know if the 3/2-approximation ratio can be improved. We expect however that this will be hard to do by extending our method: the upper bound is tight for many values of cc_1 and cc_2, and it also seems hard to improve the lower bound (see also [10]). Many additional graph structures would need to be taken into account beyond those that we considered. A more promising and important goal is to establish the MAX SNP-completeness of MaxLeaf for cubic graphs, which was also asked in [4].

Another question that remains is whether the 2-approximation for general graphs [18] can be improved. The approximability of MaxLeaf generalized to directed graphs has also been studied recently. This problem seems much harder; already much effort is needed to prove a $\sqrt{\mathrm{OPT}}$-approximation [6]. We expect however that there is still room for improvement for directed graphs. Is it even possible to prove a constant factor approximation ratio for directed graphs?

We used the novel approach of [4] of defining a (polynomial time computable) graph parameter $x(G)$, and proving upper and lower bounds in terms of $x(G)$. An algorithm is given that attains the lower bound, and combining the bounds gives the approximation ratio. This works well for MaxLeaf in cubic graphs, since known bounds without such a parameter already give a good approximation ratio. For which other problems does this approach work well?

References

1. Bonsma, P.: Spanning trees with many leaves in graphs with minimum degree three. SIAM J. Discrete Math. 22(3), 920–937 (2008)
2. Bonsma, P., Zickfeld, F.: Spanning trees with many leaves in graphs without diamonds and blossoms. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 531–543. Springer, Heidelberg (2008)
3. Cheng, X., Huang, X., Li, D., Wu, W., Du, D.: A polynomial-time approximation scheme for the minimum-connected dominating set in ad hoc wireless networks. Networks 42(4), 202–208 (2003)
4. Correa, J.R., Fernandes, C.G., Matamala, M., Wakabayashi, Y.: A 5/3-approximation for finding spanning trees with many leaves in cubic graphs. In: Kaklamanis, C., Skutella, M. (eds.) WAOA 2007. LNCS, vol. 4927, pp. 184–192. Springer, Heidelberg (2008)
5. Diestel, R.: Graph Theory. Springer, New York (1997)
6. Drescher, M., Vetta, A.: An approximation algorithm for the max leaf spanning arborescence problem. ACM transactions on algorithms (to appear)
7. Estivill-Castro, V., Fellows, M.R., Langston, M.A., Rosamond, F.A.: FPT is P-time extremal structure I. In: ACiD 2005. Texts in algorithmics, vol. 4, pp. 1–41. King's College Publications (2005)
8. Fomin, F.V., Grandoni, F., Kratsch, D.: Solving connected dominating set faster than 2^n. In: Arun-Kumar, S., Garg, N. (eds.) FSTTCS 2006. LNCS, vol. 4337, pp. 152–163. Springer, Heidelberg (2006)
9. Galbiati, G., Morzenti, A., Maffioli, F.: On the approximability of some maximum spanning tree problems. Theoret. Comput. Sci. 181(1), 107–118 (1997)

10. Griggs, J.R., Kleitman, D.J., Shastri, A.: Spanning trees with many leaves in cubic graphs. J. Graph Theory 13(6), 669–695 (1989)
11. Griggs, J.R., Wu, M.: Spanning trees in graphs of minimum degree 4 or 5. Discrete Math. 104(2), 167–183 (1992)
12. Guha, S., Khuller, S.: Approximation algorithms for connected dominating sets. Algorithmica 20(4), 374–387 (1998)
13. Kleitman, D.J., West, D.B.: Spanning trees with many leaves. SIAM J. Discrete Math. 4(1), 99–106 (1991)
14. Lemke, P.: The maximum-leaf spanning tree problem in cubic graphs is NP-complete. IMA publication no. 428, University of Minnesota, Mineapolis (1988)
15. Loryś, K., Zwoźniak, G.: Approximation algorithm for the maximum leaf spanning tree problem for cubic graphs. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 686–697. Springer, Heidelberg (2002)
16. Lu, H., Ravi, R.: Approximating maximum leaf spanning trees in almost linear time. Journal of Algorithms 29(1), 132–141 (1998)
17. Ruan, L., Du, H., Jia, X., Wu, W., Li, Y., Ko, K.: A greedy approximation for minimum connected dominating sets. Theoret. Comput. Sci. 329(1-3), 325–330 (2004)
18. Solis-Oba, R.: 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In: Bilardi, G., Pietracaprina, A., Italiano, G.F., Pucci, G. (eds.) ESA 1998. LNCS, vol. 1461, pp. 441–452. Springer, Heidelberg (1998)
19. Zickfeld, F.: Geometric and combinatorial structures on graphs. PhD thesis, Technische Universität Berlin, Berlin (2007)

On the Pseudo-achromatic Number Problem

Jianer Chen[1], Iyad A. Kanj[2], Jie Meng[1], Ge Xia[3], and Fenghui Zhang[1]

[1] Department of Computer Science, Texas A&M University, College Station, TX 77843, USA
{chen,jmeng,fhzhang}@cs.tamu.edu

[2] School of Computing, DePaul University, 243 S. Wabash Avenue, Chicago, IL 60604, USA. Supported by a DePaul University Competitive Research Grant
ikanj@cs.depaul.edu

[3] Department of Computer Science, Lafayette College, Easton, PA 18042, USA
Supported by a Lafayette College Research Grant
gexia@cs.lafayette.edu

Abstract. We study the parameterized complexity of the pseudo-achromatic number problem: Given an undirected graph and a parameter k, determine if the graph can be partitioned into k groups such that every two groups are connected by at least one edge. This problem has been extensively studied in graph theory and combinatorial optimization. We show that the problem has a kernel of at most $(k-2)(k+1)$ vertices that is constructable in time $O(m\sqrt{n})$, where n and m are the number of vertices and edges, respectively, in the graph, and k is the parameter. This directly implies that the problem is fixed-parameter tractable. We also study generalizations of the problem and show that they are parameterized intractable.

Keywords: pseudo-achromatic number, parameterized complexity, kernel, fixed-parameter tractability.

1 Introduction

The PSEUDO-ACHROMATIC NUMBER problem is to determine whether an undirected graph G can be partitioned into k groups/classes $(\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k)$ such that every two groups $\mathcal{G}_i$ and $\mathcal{G}_j$, $1 \le i < j \le k$, are connected by at least one edge. The problem is also referred to in the literature as the GRAPH COMPLETE PARTITION problem, and is formally defined as follows:

Definition 1. Let G be an undirected graph. The *pseudo-achromatic number* of G is the largest integer p such that there exists a surjective function $f : V(G) \to \{1, \ldots, p\}$ satisfying: for all i, j, where $1 \le i, j \le p$ and $i \ne j$, there exist $u \in f^{-1}(i)$, $v \in f^{-1}(j)$ such that $(u, v) \in E(G)$, where $f^{-1}(h)$ denotes the preimage set of h under f.

The PSEUDO-ACHROMATIC NUMBER problem is:

PSEUDO-ACHROMATIC NUMBER. Given a graph G and a positive integer k, determine if the pseudo-achromatic number of G is at least k.

We will be using the informal definition more frequently than the formal one.

It is easy to see that the PSEUDO-ACHROMATIC NUMBER problem is a variation of the graph coloring problem (or the achromatic number problem), the latter problem requiring the groups in the partition to be independent sets.

The PSEUDO-ACHROMATIC NUMBER problem was first introduced by Gupta in 1969 [11], and since then it has been studied extensively [1,2,3,4,9,12,13]. The problem is NP-complete even on restricted classes of graphs [3,9,12].

Kortsarz et al. [12] studied the approximability of the PSEUDO-ACHROMATIC NUMBER problem. It was proved in [12] that the problem has a randomized polynomial-time approximation algorithm of ratio $O(\sqrt{\lg n})$, which can be derandomized in polynomial time. This upper bound on the approximation ratio was shown to be asymptotically tight under the randomized model.

The PSEUDO-ACHROMATIC NUMBER problem was also considered from the extremal graph-theoretic point of view on special classes of graphs [2,4,13,14,15]. Balsubramanian et al. [1] gave a complete characterization of when the pseudo-achromatic number of the join of two graphs is the sum of the pseudo-achromatic numbers of the two graphs.

In the current paper we study the parameterized complexity of the PSEUDO-ACHROMATIC NUMBER problem. We show that the problem has a kernel of size at most $(k-2)(k+1)$ vertices that is computable in time $O(m\sqrt{n})$, where n and m are the number of vertices and edges, respectively, in the graph. This kernelization result directly gives an algorithm for the PSEUDO-ACHROMATIC NUMBER problem running in time $O(k^{k^2-k+2} + m\sqrt{n})$, thus showing that the problem is fixed-parameter tractable. The upper bound on the kernel size is obtained by developing elegant and highly non-trivial structural results, that are of independent interest.

We also study generalizations of the PSEUDO-ACHROMATIC NUMBER problem and prove that they are parameterized intractable. In particular, we consider the VERTEX GROUPING problem, in which an input instance has the form (G, H, k), where G and H are two graphs, and $k = |V(H)|$. The problem asks for the existence of a surjective function $f : V(G) \longrightarrow V(H)$ satisfying the property that $\forall u, v \in V(H)$, if $(u, v) \in E(H)$ then there exists $x \in f^{-1}(u), y \in f^{-1}(v)$ such that $(x, y) \in E(G)$. The PSEUDO-ACHROMATIC NUMBER problem is a special case of the VERTEX GROUPING problem in which the graph H is the complete graph on k vertices. The VERTEX GROUPING problem falls into the category of clustering problems, where a clustering of the graph G into $|V(H)|$ clusters is sought such that the inter-cluster properties are imposed by the graph H. We prove some (parameterized) intractability results for the VERTEX GROUPING problem. For example, we show that the problem is $W[1]$-hard, even when the graph H is the h-star graph (i.e., $K_{1,h-1}$).

2 Preliminaries

The reader is referred to Downey and Fellows' book [8] for more details about parameterized complexity theory.

A *parameterized problem* is a set of instances of the form (x, k), where $x \in \Sigma^*$ for a finite alphabet set Σ, and k is a non-negative integer called the *parameter*. A parameterized problem Q is *fixed parameter tractable*, or simply FPT, if there exists an algorithm A that on input (x, k) decides if (x, k) is a yes-instance of Q in time $f(k)n^{O(1)}$, where f is a recursive function independent of $n = |x|$. In analogy to the polynomial time hierarchy, a hierarchy for parameterized complexity, called the *W-hierarchy*, has been defined. At the 0th level of this hierarchy lies the class FPT of fixed-parameter tractable problems. The class of all problems at the i-th level of the W-hierarchy ($i > 0$) is denoted by $W[i]$. A parameterized-complexity preserving reduction (FPT-reduction) has been defined as follows. A parameterized problem Q is *FPT-reducible* to a parameterized problem Q' if there exists an algorithm of running time $f(k)|x|^c$ that on an instance (x, k) of Q produces an instance $(x', g(k))$ of Q' such that (x, k) is a yes-instance of Q if and only if $(x', g(k))$ is a yes-instance of Q', where the functions f and g depend only on k, and c is a constant. A parameterized problem Q is $W[i]$*-hard* if every problem in $W[i]$ is FPT-reducible to Q. Many well-known problems have been proved to be $W[1]$-hard including: CLIQUE, INDEPENDENT SET, SET PACKING, DOMINATING SET, HITTING SET and SET COVER. The *parameterized complexity hypothesis*, which is a working hypothesis for parameterized complexity theory, states that $W[i] \neq$ FPT for every $i > 0$.

The notion of the fixed-parameter tractability of a problem turns out to be closely related to the notion of the problem having a good data reduction (or preprocessing) algorithm. Formally speaking, a parameterized problem Q is *kernelizable* if there exists a polynomial-time reduction that maps an instance (x, k) of Q to another instance (x', k') of Q such that: (1) $|x'| \leq g(k)$ for some recursive function g, (2) $k' \leq k$, and (3) (x, k) is a yes-instance of Q if and only if (x', k') is a yes-instance of Q. The instance x' is called the *kernel* of x. It was shown that a parameterized problem is fixed-parameter tractable if and only if it is kernelizable [10].

For a graph G we denote by $V(G)$ and $E(G)$ the set of vertices and edges of G, respectively. A *matching* M in a graph G is a set of edges such that no two edges in M share an endpoint. A matching M of G is said to be *maximum* if the cardinality of M is maximum over all matchings in G. For a vertex v and a set of vertices Γ in G, we say that v is *connected to* Γ if v is adjacent to some vertex in Γ. Similarly, for two sets of vertices Γ and Γ' in G, we say that Γ is *connected to* Γ' if there exists a vertex in Γ that is connected to Γ'. For a vertex $v \in G$ we denote by $N(v)$ the set of neighbors of v in G. For a set of vertices Γ in G we denote by $N(\Gamma)$ the set of neighbors of all the vertices of Γ in G, i.e., $N(\Gamma) = \bigcup_{v \in \Gamma} N(v)$. We denote by S_h the $(h+1)$-star graph (i.e., $K_{1,h}$). The vertex of degree h in S_h is referred to as the *root* of the star, and the other h vertices are referred to as the *leaves* of the star. The *size* of the star S_h is the number of vertices in it, which is $h + 1$. We say that a graph G contains S_h if S_h is a subgraph (not necessarily induced) of G. For a background on network flows we refer the reader to [7], or to any standard book on combinatorial optimization.

3 The Kernel

In this section we show how to construct a kernel of size (number of vertices) at most $(k-2)(k+1)$ for the parameterized PSEUDO-ACHROMATIC NUMBER problem. We start by presenting some structural results that are essential for the kernelization algorithm, and that are of independent interest on their own.

3.1 Structural Results

The following lemma ascertains that graphs with large matchings have large pseudo-achromatic number.

Lemma 1. *If a graph G contains a matching of size at least $(k-1)k/2$, then the instance (G,k) is a yes-instance of the* PSEUDO-ACHROMATIC NUMBER *problem.*

Proof. Assuming that G contains a matching of at least $(k-1)k/2$ edges, we show how to group the vertices of G into k groups $(\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k)$ so that every pair of groups is connected. For every pair of groups $(\mathcal{G}_i, \mathcal{G}_j)$ where $1 \leq i < j \leq k$, we use a distinct edge (u,v) of the matching to connect the two groups by mapping the vertex u to $\mathcal{G}_i$ and v to $\mathcal{G}_i$. The remaining vertices of G are mapped arbitrarily to the groups. Since there are exactly $(k-1)k/2$ pairs of groups and at least $(k-1)k/2$ edges in the matching, every pair of groups is connected under this mapping. It follows that (G,k) is a yes-instance of the PSEUDO-ACHROMATIC NUMBER problem.

Lemma 2. *If a graph G contains a set of $k-1$ (mutually) vertex-disjoint stars of sizes $2, \ldots, k$, respectively, then the instance (G,k) is a yes-instance of the* PSEUDO-ACHROMATIC NUMBER *problem.*

Proof. Let $\mathcal{S} = \{s_1, \ldots, s_{k-1}\}$ be a set of vertex-disjoint stars in G, where s_i is the star graph S_i. We will map the vertices in $\mathcal{S}$ to k groups $(\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k)$ such that every pair of groups is connected.

For $i = 1, \ldots, k-1$, we map the root of s_i to group $\mathcal{G}_{i+1}$, and we map its leaves, in a one-to-one fashion, to groups $(\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_i)$. The remaining vertices in G are mapped arbitrarily to the groups. Since there is no overlap between the vertices of any two stars in S, this mapping is well defined. It is easy to verify now that every two distinct groups in $(\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k)$ are connected under the defined mapping. It follows that (G,k) is a yes-instance of the PSEUDO-ACHROMATIC NUMBER problem.

Lemma 3. *If a graph G contains a collection of (mutually) vertex-disjoint stars each of size at least 2 and at most $k+1$, and such that the total number of vertices in all the stars is more than $(k-2)(k+1)$, then the instance (G,k) is a yes-instance of the* PSEUDO-ACHROMATIC NUMBER *problem.*

Proof. Suppose that G contains a collection $\mathcal{P}$ of vertex-disjoint stars, each containing at least two vertices and at most $k+1$ vertices, and such that the total number of vertices of the stars in $\mathcal{P}$ is more than $(k-2)(k+1)$. Assume, to get a contradiction, that (G, k) is a no-instance of the PSEUDO-ACHROMATIC NUMBER problem.

Let s be the star graph S_h and s' be the star graph $S_{h'}$ such that s and s' are vertex-disjoint. By *merging* s and s' we mean creating the star graph $S_{h+h'}$ by identifying the roots of s and s'. Note that the size of the merged star is 1 less than the size of s plus the size of s'.

We construct from $\mathcal{P}$ a sequence of vertex-disjoint stars $\mathcal{S} = \langle s_{k-1}, \ldots, s_r \rangle$, for some integer $r \geq 1$, such that s_i has size at least $i+1$, for $r \leq i \leq k-1$. The procedure that constructs these stars is as follows.

For $i = k-1$ down to 1 do: if the largest star in $\mathcal{P}$ is an S_j, where $j \geq i$, assign it to s_i, and remove it from $\mathcal{P}$; otherwise, recursively merge the two stars of largest size in $\mathcal{P}$ and add the resulting star to $\mathcal{P}$ until either there is only one star left in $\mathcal{P}$, and in which case the procedure halts, or the largest star in $\mathcal{P}$ is an S_j, where $j \geq i$, and in which case we assign it to s_i, remove it from $\mathcal{P}$, and proceed to the next value of i in the for loop.

If a star s_i in $\mathcal{S}$ was created without merging stars in $\mathcal{P}$, we call s_i a *single star*, otherwise, we call s_i a *merged star*.

Note the following: if s_i is a merged star created from merging a collection of stars, and if s_i is used to produce a valid grouping of G, then clearly the stars that s_i was merged from can replace s_i to produce a valid grouping of G. Therefore, assuming that (G, k) is a no-instance of the PSEUDO-ACHROMATIC NUMBER problem, the last star s_r constructed by the above procedure before halting must satisfy $r \geq 2$. Otherwise, the sequence $\mathcal{S}$ would contain a set of $k-1$ vertex-disjoint stars of sizes $2, \ldots, k$, and by Lemma 2, the instance (G, k) would be a yes-instance of the problem, contradicting our assumption.

Now assume that the above procedure halts after constructing a sequence of vertex-disjoint stars $\mathcal{S} = \langle s_{k-1}, \ldots, s_r \rangle$, such that s_i has size at least $i+1$, for $2 \leq r \leq i \leq k-1$.

We define a *monotone subsequence* of $\mathcal{S}$ to be a consecutive subsequence $\langle s_i, s_{i-1} \ldots, s_j \rangle$ of $\mathcal{S}$ such that either $s_i, s_{i-1} \ldots, s_j$ are all single stars, or they are all merged stars. A monotone subsequence $\langle s_i, s_{i-1} \ldots, s_j \rangle$ of $\mathcal{S}$ is *maximal* if it is maximal under containment.

Let $\langle s_i, s_{i-1} \ldots, s_{i-\ell+1} \rangle$, $\ell \geq 1$, be a maximal monotone subsequence of $\mathcal{S}$, and note that $i-\ell+1 \geq 2$ (since $r \geq 2$). We will show that the total number of vertices in the stars of $\mathcal{P}$ that were used to form the subsequence $\langle s_i, s_{i-1} \ldots, s_{i-\ell+1} \rangle$ is at most $2(i + (i-1) + \ldots + (i-\ell+1))$. We distinguish two cases:

- **Case 1.** $\langle s_i, s_{i-1}, \ldots, s_{i-\ell+1} \rangle$ consists of single stars. We distinguish two subcases:
  - **Subcase 1.1.** $i = k-1$. Since every single star contains at most $k+1$ vertices by the statement of the lemma, the total number of vertices in the stars in the subsequence is bounded by $\ell(k+1) \leq 2(k-1+k-2+\ldots+k-\ell)$. The last inequality is true because $((k-1)-\ell+1) \geq 2$.
  - **Subcase 1.2.** $i < k-1$. By the maximality of the subsequence, s_{i+1} is a merged star. Since s_i is a single star, it is easy to verify that s_i has size

exactly $i+1$. The total number of vertices in the stars in the subsequence is bounded by $\ell(i+1) \leq 2(i+i-1+\ldots+i-\ell+1)$ because $i-\ell+1 \geq 2$.

– **Case 2.** $\langle s_i, s_{i-1}, \ldots, s_{i-\ell+1}\rangle$ consists of merged stars. Let s_j be any star in this subsequence, and suppose that s_j was constructed by merging stars $t_1, \ldots, t_q$ in $\mathcal{P}$. By the construction of s_j, the total number of leaves in the stars $t_1, \ldots, t_{q-1}$ is less than j (otherwise these stars would be sufficient to produce s_j), and the size of t_q is not larger than any of the sizes of $t_1, \ldots, t_{q-1}$. Therefore, we have:

$$|t_1| - 1 + |t_2| - 1 + \ldots + |t_{q-1}| - 1 \leq j - 1, \tag{1}$$

and

$$|t_q| \leq (|t_1| + |t_2| + \ldots + |t_{q-1}|)/(q-1). \tag{2}$$

Combining Inequality (1) with Inequality (2), and noting that $q \leq j$, we obtain:

$$|t_1| + |t_2| + \ldots + |t_q| \leq 2j. \tag{3}$$

Inequality (3) shows that the total number of vertices in the stars of $\mathcal{P}$ forming s_j is at most $2j$. By applying this inequality to each star s_j in the maximal monotone subsequence $\langle s_i, s_{i-1}, \ldots, s_{i-\ell+1}\rangle$ of merged stars, and by the linearity of addition, we obtain that the total number of vertices of $\mathcal{P}$ used to form the stars in $\langle s_i, s_{i-1}, \ldots, s_{i-\ell+1}\rangle$ is at most $2(i + (i-1) + \ldots + (i-\ell+1))$.

It follows from the above that, for any maximal monotone subsequence $\langle s_i, s_{i-1}, \ldots, s_{i-\ell+1}\rangle$ of $\mathcal{S}$, the total number of vertices of $\mathcal{P}$ used to form the stars in this subsequence is at most $2(i+(i-1)+\ldots+(i-\ell+1))$. Applying the above bound to every maximal monotone subsequence of $\mathcal{S}$, and by the linearity of addition, we conclude that the total number of vertices in $\mathcal{P}$ forming all the stars in $\mathcal{S}$ is at most $(k-r)(k+r-1)$.

Noting that the number of remaining non-empty stars in $\mathcal{P}$ cannot form an s_{r-1}, the total number of leaves in the remaining stars is at most $r-2$, and consequently, the total number of vertices in the remaining stars is at most $2(r-2)$. Therefore, the total number of vertices in $\mathcal{P}$ is at most $(k-r)(k+r-1) + 2(r-2) = k^2 - k - (r^2 - 3r + 4)$. Since $r \geq 2$, $\mathcal{P}$ has the maximum number of vertices when $r = 2$. It follows that the total number of vertices in $\mathcal{P}$ is at most $(k-2)(k+1)$, contradicting the hypothesis of the lemma.

This completes the proof.

3.2 The Auxiliary Flow Network and the Graph Pseudo-achromatic Number

Let G be a graph with pseudo-achromatic number at least k, and let $\mathcal{H}$ be a vertex grouping that partitions the vertices of G into k groups such that every pair of groups is connected.

We will show a nice relationship between the pseudo-achromatic number of a graph and graph matchings.

Let M be a maximum matching in G. Let $I = V(G) \setminus V(M)$, and note that I is an independent set. For a vertex $u \in V(M)$ we denote by $N_I(u)$ the set $N(u) \cap I$. Let M_2 be the set of edges in M whose both ends are connected to I.

Lemma 4. *Let (u, v) be an edge in M_2. Then $N_I(u) = N_I(v)$ and $|N_I(u)| = 1$.*

Proof. By definition, both $N_I(u)$ and $N_I(v)$ are nonempty. Therefore, either $N_I(u) \neq N_I(v)$ or $|N_I(u)| > 1$ would imply the existence of two different vertices $w_1 \in N_I(u)$ and $w_2 \in N_I(v)$. However, this would give an augmenting path (w_1, u, v, w_2) with respect to M, contradicting the maximality of the matching M.

Let $N_I(M_2)$ be the set $N(V(M_2)) \cap I$, and let $D = I \setminus N_I(M_2)$. We partition the edges of $M \setminus M_2$ into two sets M_1 and M_0, where M_1 consists of all the edges in $M \setminus M_2$ that have exactly one end connected to D, and $M_0 = M \setminus (M_2 \cup M_1)$. Note that the edges in $M_0 \cup M_2$ have no end connected to D (however, an edge in M_0 or in M_1 may have an end connected to $N_I(M_2)$).

The vertices in $V(M_1)$ are further partitioned into R and L, such that R is the set of vertices in $V(M_1)$ that are connected to D, and L is the set of remaining vertices in $V(M_1)$. By definition, each edge in M_1 has exactly one end in R and one end in L. Moreover, by the definition of the set M_0 and by Lemma 4, the vertices in the set D can only be connected to vertices in R (note that D is an independent set).

Let J be the subgraph of G with vertex set $R \cup D$ and edge set $\{(u, v) \mid u \in R \text{ and } v \in D\}$. We construct a flow network J_k from J as follows. Convert each undirected edge (u, v) in J, where $u \in R$ and $v \in D$, into a directed edge $\langle u, v\rangle$ of capacity 1. Add a source s and a sink t. For each vertex $u \in R$, add a directed edge $\langle s, u\rangle$ of capacity $k - 1$; and for each vertex $v \in D$, add a directed edge $\langle v, t\rangle$ of capacity 1.

Let f^* be an integer-valued maximum flow in J_k. In case of no confusion, we will identify the vertices and edges in $J_k - \{s, t\}$ with their counterparts in G.

For a vertex u, denote by f^*_u the flow through u, i.e., the total outgoing flow from u. Let $T_k = \{u \mid u \in D \text{ and } f^*_u = 0\}$. We have the following theorem whose proof is omitted due to the lack of space. The proof can be found in [6].

Theorem 1. *The instance (G, k) is a yes-instance of the* PSEUDO-ACHROMATIC NUMBER *problem if and only if $(G - T_k, k)$ is a yes-instance of the* PSEUDO-ACHROMATIC NUMBER *problem.*

The above theorem shows that the vertex set T_k can be safely removed from the graph G. Moreover, the graph $G - T_k$ has the following nice property.

Lemma 5. *The vertices in the graph $G' = G - T_k$ can be decomposed into a collection $\mathcal{P}$ of vertex-disjoint stars, each star of size at least 2 and at most $k + 1$.*

Proof. We will exhibit the collection of vertex-disjoint stars $\mathcal{P}$ in G'. We will denote by $V_{\mathcal{P}}$ the set of vertices of the stars in the collection $\mathcal{P}$, and by $E_{\mathcal{P}}$ the set of edges of the stars in $\mathcal{P}$.

The set of vertices of G' consists of the vertices in the matching M, the vertices in $N_I(M_2)$, and the vertices in D with a non-zero flow value. For a vertex u in R, let $S(u)$ be the star graph formed by the incident edge to u in M_1, together with the set of saturated edges in G' incident on u. Clearly, each such star $S(u)$ has size at least 2 and at most $k+1$ since the capacity of u in J_k is $k-1$. Moreover, for any two vertices u and v in R, the two star graphs $S(u)$ and $S(v)$ share no vertices; otherwise, there would be a shared vertex $w \in S(u) \cap S(v)$ of capacity 1 in J_k with two saturated edges incident on it, contradicting the flow properties. We add all such stars $S(u)$ to the collection $\mathcal{P}$.

We also include in $\mathcal{P}$ a maximal set of disjoint S_2 stars such that the root of each S_2 star is a vertex in $N_I(M_2)$ and its leaves are the end points of the same edge in M_2. Moreover, for every edge in M_2 whose endpoints are not yet in $V_{\mathcal{P}}$, we include it in $\mathcal{P}$ as an S_1 stars. Finally we include in $\mathcal{P}$ the matching edges in M_0 as S_1 stars.

It is clear that all the stars included in $\mathcal{P}$ are vertex-disjoint, and that each star has size at least 2 and at most $k+1$.

We claim that $V_{\mathcal{P}}$ contains all the vertices of G'. First observe that $V_{\mathcal{P}}$ contains the endpoints of all the edges in M. Second, since every vertex v in $D - T_k$ is incident on a saturated edge in G', v is included in $\mathcal{P}$. Moreover, since by definition every vertex $u \in N_I(M_2)$ forms an S_2 star with two vertices w and v, where (w, v) is an edge in M_2, and since by Lemma 4 no other vertex in $N_I(M_2)$ can form a star with the vertices w and v, it follows from the construction of $\mathcal{P}$ that $u \in V_{\mathcal{P}}$. Therefore, every vertex u in $N_I(M_2)$ is in $\mathcal{P}$, and V_P contains all the vertices of G' as desired.

3.3 Putting It All Together: The Kernelization Algorithm

Consider the decomposition of G defined in Subsection 3.2, and let M and T_k be as defined in Subsection 3.2. The kernelization algorithm is given in Figure 1.

```
Algorithm PseudoAchromaticNumberKernel
INPUT:   $(G, k)$
OUTPUT:  $(G', k')$
1. construct a maximum matching $M$ of $G$;
2. if $|M| \geq (k-1)k/2$ then return YES;
3. compute the set $T_k$ of vertices as described in Subsection 3.2; $G' = G - T_k$;
4. if $|V(G')| > (k-2)(k+1)$ then return YES;
5. return $(G', k' = k)$;
```

Fig. 1. The kernelization algorithm

Theorem 2. *Given an instance (G, k) of the* PSEUDO-ACHROMATIC NUMBER *problem, the algorithm* **PseudoAchromaticNumberKernel** *either decides the instance (G, k) correctly, or returns an instance (G', k') of the problem such that G' is a subgraph of G, $k' \leq k$, and (G, k) is a yes-instance if and only if (G', k') is. Moreover, the algorithm runs in time $O(m\sqrt{n})$, where n and m are the number of vertices and edges, respectively, in G.*

Proof. If the size of the maximum matching M in G is at least $(k-1)k/2$, then by Lemma 3.1, G is a yes-instance of the PSEUDO-ACHROMATIC NUMBER problem. Therefore, the algorithm **PseudoAchromaticNumberKernel** makes the right decision in step 2.

By Theorem 1, (G, k) is a yes-instance of the PSEUDO-ACHROMATIC NUMBER problem if and only if (G', k') is.

It suffices to argue that if $|V(G')| > (k-2)(k+1)$ (note that $k' = k$), then (G', k'), and hence (G, k), is a yes-instance of the PSEUDO-ACHROMATIC NUMBER problem, and the algorithm makes the right decision in step 4.

By Lemma 5, the set $V(G')$ can be decomposed into a collection of vertex-disjoint stars $\mathcal{P}$, each star of size at least 2 and at most $k+1$. Since $|V(G')| > (k-2)(k+1)$, it follows that the number of vertices in $\mathcal{P}$ is more than $(k-2)(k+1)$. Consequently, $\mathcal{P}$ satisfies the statement of Lemma 3, and (G', k') is a yes-instance of the PSEUDO-ACHROMATIC NUMBER problem.

Finally, to see that the algorithm **PseudoAchromaticNumberKernel** runs in time $O(m\sqrt{n})$, note first that the maximum matching M can be computed in $O(m\sqrt{n})$ time by a standard maximum matching algorithm [7]. Noting that the flow network J_k is a bipartite graph with at most $O(n)$ vertices and $O(m)$ edges, the maximum flow f^* in J_k can be computed in time $O(m\sqrt{n})$ [7]. All other steps can be performed in time $O(m)$, and the theorem follows.

Corollary 1. *The* PSEUDO-ACHROMATIC NUMBER *problem has a kernel of at most $(k-2)(k+1)$ vertices that is computable in time $O(m\sqrt{n})$, where n and m are the number of vertices and edges, respectively, in the graph, and k is the parameter.*

Using the $(k-2)(k+1)$ upper bound on the kernel size, we can solve the PSEUDO-ACHROMATIC NUMBER problem by enumerating all possible assignments of the vertices in the graph to the k groups, then checking whether any such assignment yields a valid grouping. We have the following corollary:

Corollary 2. *The* PSEUDO-ACHROMATIC NUMBER *problem can be solved in time $O(k^{k^2-k+2} + m\sqrt{n})$, and hence is fixed-parameter tractable, where n and m are the number of vertices and edges, respectively, in the graph.*

Proof. Given an instance (G, k) of the PSEUDO-ACHROMATIC NUMBER problem, where G has n vertices and m edges, we apply the algorithm **PseudoAchromat-icNumberKernel** to (G, k). The algorithm runs in $O(m\sqrt{n})$ time and either accepts the instance (G, k) correctly, or returns a kernel (G', k) where G' has at most $(k-2)(k+1)$ vertices. Now if G' can be partitioned into k groups that are mutually connected, then every vertex in G' must belong to one of the k groups. Therefore, there are at most $k^{(k-2)(k+1)}$ ways to partition G' into k groups. For each such partitioning, we can check whether the corresponding groups are mutually connected; this can be done in time $O(k^4)$. If we do not succeed in finding a valid partitioning then clearly the algorithm can reject the instance; otherwise, the algorithm returns a valid partitioning. The total running time of the algorithm is $O(k^4 \cdot k^{(k-2)(k+1)} + m\sqrt{n})$, which is $O(k^{k^2-k+2} + m\sqrt{n})$.

4 Hardness Results for the Vertex Grouping Problem

Recall from Section 1 that in the VERTEX GROUPING problem we are given an instance (G, H, k), where G and H are two graphs, and $k = |V(H)|$, and the problem asks for the existence of a surjective function $f : V(G) \longrightarrow V(H)$ satisfying the property that for all $u, v \in V(H)$, if $(u, v) \in E(H)$ then there exist $x \in f^{-1}(u)$ and $y \in f^{-1}(v)$ such that $(x, y) \in E(G)$. The VERTEX GROUPING problem can be defined more intuitively as follows.

Let G be an undirected graph. We define an operation on G, called *vertex grouping*, applied to a subset of vertices S as follows: remove all the vertices in S from G, add a new vertex w, and connect w to all the neighbors of S in $G - S$. The VERTEX GROUPING problem is:

> VERTEX GROUPING: Given two graphs G and H, where H is a graph of k vertices, and k is the parameter, decide if H can be obtained from G by a sequence of vertex grouping operations.

If H in the above definition is the complete graph on k vertices, then the VERTEX GROUPING problem becomes the PSEUDO-ACHROMATIC NUMBER problem, and hence is fixed parameter tractable. The following theorem shows that the VERTEX GROUPING problem is parameterized intractable in general.

Theorem 3. *The* VERTEX GROUPING *problem is* $W[1]$*-hard.*

Proof. We reduce the $W[1]$-hard problem INDEPENDENT SET to the VERTEX GROUPING problem.

Let (G, k) be an instance of the INDEPENDENT SET problem. Construct a graph G' by adding a new vertex w to G and connecting w to every vertex in G. Let H be a $(k+1)$-star with root r_H. Define the mapping π that, on an instance (G, k) of INDEPENDENT SET, produces the instance $(G', H, k+1)$ of VERTEX GROUPING. Clearly, the mapping π is computable in polynomial time, and hence π is an FPT-reduction. We show that (G, k) is a yes-instance of INDEPENDENT SET if and only if $(G', H, k+1)$ is a yes-instance of VERTEX GROUPING.

In effect, suppose that (G, k) is a yes-instance of INDEPENDENT SET, and let I be an independent set in G of size k. Consider the function $f : V(G') \longrightarrow V(H)$ that maps the k vertices of I in G' to the k leaves of the star H, in a one-to-one fashion, and maps all other vertices of G' to the root r_H of H. Then it is easy to verify that H is a vertex grouping of G' under the function f.

Conversely, suppose that H is a vertex grouping of G' under a function f. Consider any set of vertices I in G of cardinality k satisfying $f(I) = V(H) \setminus \{r_H\}$. Clearly, such a set I exists by the definition of the vertex grouping. Note that f is a bijection from I to $V(H) \setminus \{r_H\}$. Now for any two distinct vertices u and v of I, u and v are not adjacent in G, otherwise, by the definition of vertex grouping, $f(u)$ and $f(v)$ would be adjacent in H. It follows that I is an independent set of size k in G. This completes the proof.

The Exponential Time Hypothesis (ETH) states that many NP-hard problems including 3-SAT, INDEPENDENT SET, and VERTEX COVER, cannot be solved in time $2^{o(n)}$ (n is the number of variables for 3-SAT, and the number of vertices for INDEPENDENT SET and VERTEX COVER). ETH has become a working hypothesis for many researchers in the area of exact and parameterized algorithms. It was shown in [5] that, unless ETH fails, INDEPENDENT SET cannot be solved in time $n^{o(k)}$. It was also shown in [5] that if a parameterized problem Q is reducible to a parameterized problem Q' by an FPT reduction, called *linear fpt-reduction*, that preserves the order of the parameter and does not increase the size of the instance by more than a polynomial factor, and if Q cannot be solved in time $n^{o(k)}$ then it follows that Q' cannot be solved in time $n^{o(k)}$. Clearly, the reduction from INDEPENDENT SET to VERTEX GROUPING, given in the proof of Theorem 3, is a linear fpt-reduction. Therefore, we have the following theorem:

Theorem 4. *Unless ETH fails, the* VERTEX GROUPING *problem cannot be solved in time $n^{o(k)}$, where n and k are the number of vertices in G and H, respectively.*

We illustrate a relationship between the GRAPH ISOMORPHISM problem and the VERTEX GROUPING problem. Let G_1 and G_2 be two graphs on n vertices. We are interested in knowing how "similar" G_1 and G_2 are, under the notion of vertex grouping defined above. For this purpose, we introduce the following parameterized problem:

> GRAPH STRUCTURAL SIMILARITY: given two graphs G_1 and G_2 on n vertices, and a parameter k, decide if there exists a graph H of k vertices such that both (G_1, H, k) and (G_2, H, k) are yes-instances of the VERTEX GROUPING problem.

Intuitively, the graph structural similarity measures the degree of similarity (i.e., k) between two graphs under the notion of vertex grouping. In particular, if $k = n$, then the GRAPH STRUCTURAL SIMILARITY problem is equivalent to the GRAPH ISOMORPHISM problem.

Theorem 5. *The* GRAPH STRUCTURAL SIMILARITY *problem is $W[1]$-hard.*

Proof. As was shown in Theorem 3, the VERTEX GROUPING problem is $W[1]$-hard when the graph H is a star. An FPT-reduction can be constructed that takes an instance (G, H, k), where G has n vertices and H is a k-star, of the VERTEX GROUPING problem to an instance (G_1, G_2, k) of the GRAPH STRUCTURAL SIMILARITY problem, where $G_1 = G$ and G_2 is the n-star. Observing that any sequence of vertex grouping operations that are applied to G_2 can only result in a star graph, the $W[1]$-hardness of GRAPH STRUCTURAL SIMILARITY follows.

The reduction described in the proof of the above theorem is clearly a linear fpt-reduction. Therefore, it follows from Theorem 4 that:

Theorem 6. *Unless ETH fails, the* GRAPH STRUCTURAL SIMILARITY *problem cannot be solved in time $n^{o(k)}$, where n is the number of vertices in G_1 and G_2, and k is the parameter.*

References

1. Balasubramanian, R., Raman, V., Yegnanarayanan, V.: On the pseudoachromatic number of join of graphs. International Journal of Computer Mathematics 80(9), 1131–1137 (2003)
2. Bhave, V.: On the pseudoachromatic number of a graph. Fundamenta Mathematicae 102(3), 159–164 (1979)
3. Bodlaender, H.: Achromatic number is NP-complete for cographs and interval graphs. Information Processing Letters 32(3), 135–138 (1989)
4. Bollobás, B., Reed, B., Thomason, A.: An extremal function for the achromatic number. Graph Structure Theory, 161–166 (1991); pp. 18–37 (2005)
5. Chen, J., Huang, X., Kanj, I., Xia, G.: Strong computational lower bounds via parameterized complexity. Journal of Computer and System Sciences 72(8), 1346–1367 (2006)
6. Chen, J., Meng, J., Kanj, I., Xia, G., Zhang, F.: On the pseudo-achromatic number problem. Technical report 08-006 at:`http://www.cdm.depaul.edu/research/Pages/TechnicalReports.aspx`.
7. Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Introduction to Algorithms, 2nd edn. McGraw-Hill Book Company, Boston (2001)
8. Downey, R., Fellows, M.: Parameterized Complexity. Springer, Heidelberg (1999)
9. Edwards, K., McDiarmid, C.: The complexity of harmonious coloring for trees. Discrete Applied Mathematics 57, 133–144 (1995)
10. Downey, R., Fellows, M., Stege, U.: Parameterized complexity: a framework for systematically confronting computational intractability, in Contemporary Trends in Discrete Mathematics. In: Graham, R., Kratochvíl, J., Nešetřil, J., Roberts, F. (eds.) Proc. DIMACS-DIMATIA Workshop, Prague. AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 49, pp. 49–99 (1997)
11. Gupta, R.P.: Bounds on the chromatic and achromatic numbers of complementary graphs. In: Tutte, W.T. (ed.) Recnet Progress in Combinatorics, Proc. 3rd Waterloo Conference on Combinatorics, Waterloo, pp. 229–235. Academic Press, New York (1969)
12. Kortsarz, G., Radhakrishnan, J., Sivasubramanian, S.: Complete partitions of graphs. In: Proc. 16th Annual ACM-SIAM symposium on Discrete algorithms, pp. 860–869 (2005)
13. Sampathkumar, E., Bhave, V.: Partition graphs and coloring numbers of graphs. Discrete Mathematics 16, 57–60 (1976)
14. Yegnanarayanan, V.: On pseudocoloring of graphs. Utilitas Mathematica 62, 199–216 (2002)
15. Yegnanarayanan, V.: The pseudoachromatic number of a graph. Southern Aian Bulletin of Mathematics 24, 129–136 (2002)

Making Role Assignment Feasible: A Polynomial-Time Algorithm for Computing Ecological Colorings[⋆]

Pilu Crescenzi[1], Miriam Di Ianni[2], Federico Greco[3], Gianluca Rossi[2], and Paola Vocca[4]

[1] Dipartimento di Sistemi e Informatica, Università di Firenze, Firenze, Italy
[2] Dipartimento di Matematica, Università di Roma "Tor Vergata", Roma, Italy
{miriam.di.ianni,gianluca.rossi}@uniroma2.it
[3] Dipartimento di Matematica, Università di Perugia, Perugia, Italy
greco@dipmat.unipg.it
[4] Dipartimento di Matematica "Ennio De Giorgi", Università del Salento, Lecce, Italy
paola.vocca@unile.it

Abstract. In this paper, we study the problem of ecologically coloring a graph. Intuitively, an ecological coloring of a graph is a role assignment to the nodes of the graph, such that two nodes surrounded by the same set of roles must be assigned the same role (Borgatti and Everett, 1992). We prove that, for any simple undirected graph G with n_G distinct neighborhoods and for any integer k with $1 \leq k \leq n_G$, G admits an ecological coloring which uses exactly k roles, and that this coloring can be computed in polynomial time. Our result strongly contrasts with the NP-completeness result of the regular coloring problem, where it is required that two nodes with the same role must be surrounded by the same set of roles (Fiala and Paulusma, 2005). Hence, we conclude that not only the ecological coloring is easier to understand as a model of social relationships (Borgatti and Everett, 1994), but it is also feasible from a computational complexity point of view.

1 Introduction

One of the main goals of the analysis of a social network consists of determining patterns of relationships and interactions among social actors (such as persons and groups) in order to identify the social structure of the network [4,5]. To this aim, a social network is usually represented as a graph, whose nodes denote the network members and whose edges denote their relationships, which is analyzed from a structural point of view by means of methods that broadly fall into one of the following two categories: *relational analysis* methods that are often used in order to identify central members or to partition the graph into clusters, and *positional analysis* methods that examine the similarity between the connection of two network members with the other members. *Role assignment* is one of the main positional analysis methods, whose goal consists of classifying the members of a social network, so that members which are equally classified can be considered to behave in a similar way or to play a similar role [9]. If the

[⋆] This work was partially supported by the EU under the EU/IST Project 15964 *AEOLUS*.

number of roles is limited, this kind of classification can turn out to be extremely useful while trying to understand the overall structure of very complex social networks.

Different kinds of role assignment have been introduced in the literature. A *strong structural* role assignment, for example, imposes that if two actors play the same role, then they must have the same neighborhood [10], while a *regular* role assignment imposes that if two actors play the same role, then they must be surrounded by the same set of roles [1,12]. In this paper we are interested in another kind of role assignment, that is, *ecological* role assignments, according to which if two network members are surrounded by the same set of roles, then they must play the same role: in other words, the role played by a social actor is completely determined by the roles played by its neighbors [2]. The relationship imposed by an ecological role assignment is the opposite of the one imposed by a regular role assignment: a role assignment that satisfies both constraints is called *perfect* [3]. As stated by Borgatti and Everett, an ecological role assignment is *easier to understand* as a model of social relationships, in which a member's neighborhood tends to shape this member into that or this kind.

Computing a role assignment for a given network is equivalent to computing a coloring of the network nodes, such that the constraint imposed by the role assignment is satisfied by the colors assigned to the nodes. For instance, an *ecological coloring* of a graph is an assignment of colors to the nodes of the graph such that if two nodes "see" the same set of colors, then they are assigned the same color. Our main contribution is proving that, *for any simple undirected graph G with n_G distinct neighborhoods and for any integer k with $1 \leq k \leq n_G$, G admits an ecological coloring which uses exactly k colors*, and that *this coloring can be computed in polynomial time* by means of a bottom-up approach and by making use of some combinatorial properties of graphs whose nodes have all distinct neighborhoods.

Our results strongly contrast with the results obtained in [7,11] according to which deciding whether a graph can be regularly colored with k colors is NP-complete, for any $k \geq 2$. This somehow implies that an ecological role assignment not only is easier to understand but it is also more useful from a computational point of view than a regular role assignment, since it can be always efficiently computed. This allows the network analyzer to reduce the size of complex social networks to a desired size: this feature is extremely important in network analysis for which different aspects can be studied depending on network dimension (for example, the degree of relevance in information retrieval and the degree of relationship in e-communities discovery).

The paper is structured as follows. In the rest of this section, we give some preliminary definitions and results concerning the regular and the ecological coloring of a graph. In Section 2 we introduce the notion of twin-free graph [6,8], we prove how we can restrict our attention to this kind of graphs and we show some properties of these graphs that will be useful for designing the main algorithm. In Section 3 we prove our main result, that is, that any graph can be ecologically colored by using any reasonable number of colors. Finally, in Section 4 we conclude by stating our main open question.

1.1 Preliminaries

In the following, the term 'graph' will always denote a simple (that is, with no self-loops and multiple edges) undirected graph (unless otherwise specified). Given a graph

$G = (V, E)$, for any node $u \in V$, $N(u)$ denotes the neighborhood of u. A *coloring* of G which uses k colors is a surjective function $r : V \to [k]$.[1] Given a coloring r of G which uses k colors, the *colorhood* of a node $u \in V$ with respect to r is defined as the set

$$C_r(u) = \big\{i \in [k] : \exists v \in N(u)\ [r(v) = i]\big\}$$

A coloring r of a graph $G = (V, E)$ is *regular* [1] if, for any $u, v \in V$,

$$r(u) = r(v) \Rightarrow C_r(u) = C_r(v).$$

Observe that a graph can be regularly colored with one color if and only if either it contains no isolated nodes or its nodes are all isolated. Furthermore, any graph can be regularly colored with n colors, where n is the number of nodes of the graph. The k-REGULAR ROLE ASSIGNMENT (in short, k-RERA) decision problem consists in deciding whether a graph G admits a regular coloring which uses k colors: in [7,11] it is proved that k-RERA is NP-complete for any $k \geq 2$.

A coloring r of a graph $G = (V, E)$ is *ecological* [2] if, for any $u, v \in V$,

$$C_r(u) = C_r(v) \Rightarrow r(u) = r(v).$$

Observe that any graph can be ecologically colored with one color. However, it is not true that any graph can be ecologically colored with n colors, where n is the number of nodes of the graph (see the results of Section 2). In general, an ecological coloring is not necessarily regular and a regular coloring is not necessarily ecological. The k-ECOLOGICAL ROLE ASSIGNMENT (in short, k-ECRA) decision problem consists in deciding whether a graph G admits an ecological coloring which uses k colors.

2 Twin-Free Graphs

According to the definition of an ecological coloring, two nodes with the same neighborhood must be colored with the same color. The number of distinct neighborhoods contained in a graph is thus an upper bound on the number of colors that can be used by any ecological coloring. The following definition and results formalize this statement.

Definition 1 ([6,8]). *A graph $G = (V, E)$ is* twin-free *if, for any $u, v \in V$, $N(u) \neq N(v)$.*

Observe that any twin-free graph with n nodes can clearly be ecologically colored with n colors.

Given a graph $G = (V, E)$, we define an equivalence relation ρ_N on the vertices of G as follows: two vertices $u, v \in V$ are equivalent if and only if $N(u) = N(v)$. The *neighborhood graph* corresponding to G is the twin-free graph $G_N = (V_N, E_N)$ where V_N is the set of equivalence classes with respect to the relation ρ_N, and[2] $(x, y) \in E_N$ if all nodes in the equivalence class x are adjacent to all nodes in the equivalence class y. The *neighborhood degree* n_G of G is defined as the number of nodes in G_N. The following two results can be easily proved.

[1] In the following, for any positive integer n, $[n]$ will denote the set $\{1, 2, \ldots, n\}$.

[2] In the following, we will indicate an edge of a graph by using brackets instead of curly brackets, in order to avoid confusion with other two-elements sets.

Theorem 1. *A graph can be ecologically colored with k colors if and only if its neighborhood graph can be ecologically colored with k colors.*

Corollary 1. *Each graph G can be ecologically colored with n_G colors and it cannot be ecologically colored with $k > n_G$ colors.*

The following structural property of twin-free graphs has been independently proved in [6]: for the sake of completeness, we include our self-contained proof in the Appendix.

Theorem 2. *Let $G = (V, E)$ be a twin-free graph with n nodes. Then, there exists a node $u \in V$, such that the graph induced by $V - \{u\}$ is a twin-free graph with $n - 1$ nodes.*

The previous theorem allows us to prove the following result, which will be used as a subroutine within the second phase of the general ecological coloring algorithm (see line 9 of Figure 3).

Theorem 3. *Let $G = (V, E)$ be a twin-free graph with n nodes. Then, G can be ecologically colored with $n - 1$ colors.*

Proof. Let $G = (V, E)$ be a twin-free graph with n nodes. From Theorem 2 it follows that there exist two nodes u and v such that the graph $G^{u,v}$ induced by $V - \{u, v\}$ is a twin-free graph with $n-2$ nodes. Let r be any coloring of G that assigns $n-2$ different colors to the nodes of $G^{u,v}$ and that assigns the same new color to u and v. In order to prove that r is ecological, we proceed by contradiction and assume that there exist two nodes p and q such that $r(p) \neq r(q)$ and $C_r(p) = C_r(q)$. We then distinguish the following cases.

1. $\{p, q\} = \{u, v\}$. In this case, $r(p) = r(q)$, and, hence, we get a contradiction.
2. $\{p, q\} \subseteq V - \{u, v\}$. In this case, since $G^{u,v}$ is twin-free and since all its nodes are colored with different colors, $C_r(p)$ must be different from $C_r(q)$, and, hence, we get a contradiction.
3. $p \in \{u, v\} \wedge q \in V - \{u, v\}$. In this case, since all the nodes of $G^{u,v}$ are colored with different colors and since $C_r(p) = C_r(q)$, we have that $N(p) \cap (V - \{u, v\}) = N(q) \cap (V - \{u, v\})$. Moreover, p and q cannot be adjacent, since otherwise $r(q) \in C_r(p) - C_r(q)$, contradicting the fact that $C_r(p) = C_r(q)$. Since G is twin-free, exactly one node among p and q must be adjacent to the node in $\{u, v\} - \{p\}$: this implies that either $r(p) \in C_r(p) - C_r(q)$ or $r(p) \in C_r(q) - C_r(p)$, and, hence, we get a contradiction.
4. $p \in V - \{u, v\} \wedge q \in \{u, v\}$. This case is symmetric to the previous one. □

We conclude this section by stating two other interesting properties of twin-free graphs. The first one is formalized in the following result.

Lemma 1. *Let $G = (V, E)$ be a twin-free graph of n nodes and let $\mathcal{I} = \{I_1, \ldots, I_h\}$ be a partition of V into h non-empty independent sets with $2 \leq h < n$. Then, there exist $p, q \in [h]$ such that the pair of sets I_p and I_q induces a non-complete bipartite graph with at least one edge.*

Proof. The proof proceeds by contradiction. Assume that all pairs of independent sets in $\mathcal{I}$ induce either a complete bipartite graph or an independent set. This, in turn, implies that, since $h < n$ and, hence, at least one independent set is not a singleton, there must exist at least two nodes u and v in the same independent set which, for any $i \in [h]$, are adjacent either to all nodes or to no node in I_i. That is, u and v have the same neighborhood, thus contradicting the twin-free property of G. □

In order to state the last property of twin-free graphs that we will use in the next section, we need the following definition.

Definition 2. *Let $G = (V, E)$ be a graph and $\mathcal{I} = \{I_1, \ldots, I_h\}$ be a partition of V into $h \geq 1$ non-empty sets such that $I_1, \ldots, I_{h-1}$ are independent sets. We say that $\mathcal{I}$ is an* ecological family for G of size h *if either $h = 1$ or, for any $i, j \in [h]$ with $i \neq j$ and for any $u \in I_i$ and $v \in I_j$, there exists $t \in [h]$ such that u is adjacent to some node in I_t and v is not adjacent to any node in I_t.*

Lemma 2. *Let $G = (V, E)$ be a twin-free graph of n nodes and let $\mathcal{I} = \{I_1, \ldots, I_h\}$ be an ecological family for G of size $h < n$. Then, there exists an ecological family $\mathcal{I}'$ for G of size $h + 1$ or $h + 2$. Furthermore, $\mathcal{I}'$ can be computed in polynomial time.*

Proof. If I_h is not an independent set, then let I_{h1} be a maximal independent set for the subgraph of G induced by I_h, $I_{h2} = I_h - I_{h1}$ and $\mathcal{I}' = \mathcal{I} - \{I_h\} \cup \{I_{h1}, I_{h2}\}$. For any $i \in [h-1]$ and for any $j \in [h]$ such that $i \neq j$, let $u \in I_i \in \mathcal{I}$ and $v \in I_j \in \mathcal{I}$ and let $I_t \in \mathcal{I}$ be such that u is adjacent to some node in I_t and v is not adjacent to any node in I_t: clearly, this is still true in $\mathcal{I}'$ with I_t possibly replaced by I_{h1} or by I_{h2}. If, instead, $i = h_1$ and $j = h_2$, that is, $u \in I_{h1}$ and $v \in I_{h2}$, then, since I_{h1} is a maximal independent set for the subgraph of G induced by I_h, v has to be adjacent to some node in I_{h1} while u is not adjacent to any other node in I_{h1}.

Assume now that I_h is an independent set. In this case, from Lemma 1 it follows that there exist $p, q \in [h]$ such that the subgraph of G induced by $I_p \cup I_q$ is not an independent set and is not bipartite complete. We now find a partition of I_p into $I_{p1} \cup I_{p2}$ and a partition of I_q into $I_{q1} \cup I_{q2}$ such that $I_{p1} \neq \emptyset$, $I_{q1} \neq \emptyset$, $I_{p2} \cup I_{q2} \neq \emptyset$ and $I_{p1} \cup I_{q1}$ is an independent set. Since, the subgraph of G induced by $I_p \cup I_q$ is not an independent set and is not a bipartite complete graph, it has to contain two adjacent nodes, $u_1 \in I_p$ and $v_1 \in I_q$, and two non-adjacent nodes $u_2 \in I_p$ and $v_2 \in I_q$. Notice that they are not necessarily distinct, that is, it could happen that either $u_1 \equiv u_2$ or $v_1 \equiv v_2$ (but not both). Without loss of generality assume that $u_1 \not\equiv u_2$. Let $I_{q1} = \{x \in I_q : (u_2, x) \notin E\}$, $I_{p1} = \{x \in I_p : \forall y \in I_{q1}[(x, y) \notin E]\}$, $I_{p2} = I_p - I_{p1}$ and $I_{q2} = I_q - I_{q1}$: notice that $u_2 \in I_{p1}$ and $v_2 \in I_{q1}$. Furthermore, if $v_1 \equiv v_2$ then $u_1 \in I_{p2}$, otherwise $v_1 \in I_{q2}$. Finally, $I_{p1} \cup I_{q1}$ is an independent set by construction. Notice that I_{q2} could be empty: this happens if no node in I_q is adjacent to u_2, that is, only if u_2 is an isolated node in the subgraph induced by $I_p \cup I_q$.

We now define $\mathcal{I}'$ as follows.

1. $\mathcal{I}' = \mathcal{I} - \{I_p, I_q\} \cup \{I_{p1}, I_{p2}, I_{q1}, I_{q2}\}$ if $I_{q2} \neq \emptyset$ and $I_{p2} \neq \emptyset$.
2. $\mathcal{I}' = \mathcal{I} - \{I_p, I_q\} \cup \{I_{p1}, I_{p2}, I_{q1}\}$ if $I_{q2} = \emptyset$.
3. $\mathcal{I}' = \mathcal{I} - \{I_p, I_q\} \cup \{I_{p1}, I_{q1}, I_{q2}\}$ if $I_{p2} = \emptyset$.

It remains to show that $\mathcal{I}'$ is an ecological family for G. To this aim, consider a pair of nodes $u \in I_i \in \mathcal{I}$ and $v \in I_j \in \mathcal{I}$ with $i \neq j$ and let $I_t \in \mathcal{I}$ be such that u is adjacent to some node in I_t and v is not adjacent to any node in I_t. This is still true in $\mathcal{I}'$ with I_t eventually replaced by one set out of $I_{p1}, I_{p2}, I_{q1}, I_{q2}$. Hence, assume $i = j$. One of the following cases may occur.

- $u \in I_{p1}$ and $v \in I_{p2}$: in this case, by construction, u is not adjacent to any node in I_{q1} and v has to be adjacent to some node in I_{q1} (otherwise, v should be contained in I_{p1}).
- $u \in I_{q1}$ and $v \in I_{q2}$: this case is similar to the previous one.

Hence, $\mathcal{I}'$ is an ecological family for G and its size is either $h + 1$ or $h + 2$. □

3 The Ecological Coloring Algorithm

In this section we prove the existence of ecological colorings with any feasible number of colors. According to Theorem 1 we can state our main result in terms of twin-free graphs.

Let $G = (V, E)$ be a twin-free graph of n nodes and $k < n$ the number of colors we are interested in. Our algorithm works in two phases. During the first phase, an ecological family for G of size either $k - 1$ or k is computed by using Lemma 2. If the size of such a family is k an ecological coloring is directly derived from the family. Conversely, if the size of the ecological family is smaller than k then the second phase is started and the informations conveyed by the ecological family are used to compute the k-ecological coloring of G. The algorithm is described in Figure 1.

```
Input: A twin-free graph $G = (V, E)$ with $|V| = n$ and integer $k$ with $1 \leq k \leq n$.
Output: An ecological coloring $r$ for $G$ which uses $k$ colors.
1: Phase 1: apply Algorithm Partitioning with input $G$ and $k$ to compute an ecological
   family $\mathcal{I}$ for $G$ of size $h \leq k$;
2: if $h = k$ then
3:    for $(i \leftarrow 1; i \leq k; i \leftarrow i + 1)$ do
4:       $\forall u \in I_i$: $r(u) \leftarrow i$;
5: else
6:    Phase 2: apply Algorithm Refining with input $\mathcal{I}$ and $k$ to compute an ecological color-
      ing $r$ for $G$ which uses $k$ colors;
```

Fig. 1. The ecological coloring algorithm

In the next two subsections the two phases are detailed and the proof that they are in fact correct is drawn. This allows us to prove our main result, stated in the following theorem.

Theorem 4. *For any twin-free graph $G = (V, E)$ with n nodes and for any $k \in [n]$, G admits an ecological coloring which uses k colors. Such a coloring can be computed in polynomial time.*

Proof. Lemma 3 insures that Algorithm `Partitioning` computes indeed an ecological family $\mathcal{I}$ for G. If $|\mathcal{I}| = k$, lines 2–4 are executed. We now show that they compute an ecological coloring for G. Let u and v be two nodes such that $r(u) = p \neq r(v) = q$. Hence, $u \in I_p \in \mathcal{I}$ and $v \in I_q \in \mathcal{I}$. Since $\mathcal{I}$ is an ecological family for G, then there exists t such that u is adjacent to some node in I_t and v is not adjacent to any node in I_t. Since $r(x) = t$ if and only if $x \in I_t$, $C_r(u) \neq C_r(v)$. Hence, r is an ecological coloring of G. Conversely, if the **else** statement is executed then the computed coloring is ecological by Lemma 4. □

3.1 Phase 1: Computing an Ecological Family

This phase iteratively applies Lemma 2 in order to construct an ecological family for G. More formally, Algorithm `Partitioning` performing this task is described in Figure 2.

```
Input: A twin-free graph $G = (V, E)$ with $|V| = n$ and integer $k$ with $1 \leq k \leq n$.
Output: An ecological family $\mathcal{I}$ for $G$ of size at most $k$.
 1: $\mathcal{I} \leftarrow \{V\}$;
 2: while $|\mathcal{I}| < k - 1$ do
 3:     Let $\mathcal{I}'$ the ecological family for $G$ obtained by applying Lemma 2 to $\mathcal{I}$;
 4:     $\mathcal{I} \leftarrow \mathcal{I}'$;
```

Fig. 2. `Partitioning`: Phase 1 of the ecological coloring algorithm

Lemma 3. *Let G be a twin-free graph with n nodes and k a positive integer such that $k \leq n$. Algorithm* `Partitioning` *computes an ecological family for G of size either $k - 1$ or k.*

Proof. The condition of the **while** loop (line 2)requires $|\mathcal{I}| < k - 1$: since $k \leq n$, then Lemma 2 can be actually applied to set $\mathcal{I}$ at each iteration. The algorithm ends by computing an ecological family for G of size either $k - 1$ or k. □

3.2 Phase 2: The Last Refinement

In order to describe this phase, we need to introduce the notion of color graph. Given a coloring r of a graph $G = (V, E)$ which uses k colors, the *color graph* $C^{G,r} = ([k], E^{G,r})$ includes the edge (i, j) if and only there exist $u, v \in V$ such that $r(u) = i$, $r(v) = j$ and $(u, v) \in E$ (note that, in general, a color graph is not simple). If Phase 1 ends with an ecological family for G of size $k - 1$, the second phase is started and the ecological family for G of size $(k - 1)$ is used to compute the k-ecological coloring of G. Algorithm `Refining` shown in Figure 3 first tries to increase by 1 the size of the ecological family (according to Lemma 2) in order to color it in the same way as in lines 2–4 of the Algorithm in Figure 1. If this is not possible, then it both increases the size of $\mathcal{I}$ by 2 (always according to Lemma 2) and it computes a $(k - 2)$-coloring r_2 of $\mathcal{I}$. In order to perform the last step, it builds the color graph C^{G,r_1} of G with respect to the $(k - 1)$-coloring r_1 deriving from $\mathcal{I}$ and applies to it Theorem 2: observe that this

```
Input: An ecological family $\mathcal{I} = \{I_1, I_2, \ldots, I_{k-1}\}$ for a twin-free graph $G = (V, E)$ with
    $|V| = n$, and a positive integer $k \leq n$ .
Output: An ecological coloring $r$ of $G$ which uses $k$ colors.
 1: Let $r_1$ be the coloring according to which all nodes in $I_i$ are colored with color $i$, $i =$
    $1, \ldots, k-1$;
 2: if $\exists 1 \leq p \leq k-1$ and $u, v \in I_p$ such that $C_{r_1}(u) \neq C_{r_1}(v)$ then
 3:     $I_{p1} \leftarrow \{x \in I_p : C_{r_1}(x) = C_{r_1}(u)\}$;
 4:     $\forall u \notin I_{p1}, r(u) \leftarrow r_1(u); \forall u \in I_{p1}, r(u) \leftarrow k$;
 5: else
 6:     Let $I_p, I_q \in \mathcal{I}$ such that $I_p \cup I_q$ is not an independent set and induces a non-complete
        bipartite graph with at least two nodes in $I_p$ and at least two nodes in $I_q$;
 7:     Partition $I_p$ into $I_{p1}, I_{p2}$ and $I_q$ into $I_{q1}, I_{q2}$ such that all of them are not empty, $I_{p1} \cup I_{q1}$
        is an independent set, and both $I_{p1} \cup I_{q2}$ and $I_{p2} \cup I_{q1}$ are not independent sets (see proof
        of Lemma 2);
 8:     $C^{G,r_1} \leftarrow$ color graph of $G$ with respect to $r_1$;
 9:     Color $C^{G,r_1}$ with $k-2$ colors by applying Theorem 3: let $r_2$ be such a coloring;
10:     Transform $r_2$ into a coloring $r_3$ of $G$: $\forall I \in C^{G,r_1} \forall u \in I : r_3(u) \leftarrow r_2(I)$;
11:     $\forall u \notin I_{p1} \cup I_{q2} : r(u) \leftarrow r_3(u); \forall u \in I_{p1} : r(u) \leftarrow k-1; \forall u \in I_{q2} : r(u) \leftarrow k$;
```

Fig. 3. `Refining`: Phase 2 of the ecological coloring algorithm

is possible since C^{G,r_1} is simple because nodes with the same color belong to the same independent set. Finally, by exploiting both r_2 and the ecological family of size $k+1$, it computes the ecological coloring r for G that uses k colors.

Lemma 4. *Let G be a graph and $\mathcal{I}$ an ecological family for G of size $k-1$. Then, Algorithm* `Refining` *computes an ecological coloring of G with k colors.*

Proof. Notice that the coloring computed at line 1 is the same as that computed at line 3 of the algorithm in Figure 1 and, hence, it is ecological. When the **if** statement is executed (line 2), the partition $\mathcal{I} - \{I_p\} \cup \{I_{p1}, I_{p2}\}$ computed at line 3 is ecological by Lemma 2. Hence, the coloring r computed at line 4 can be easily proved to be ecological by the same arguments in the proof of Lemma 3.

Conversely, assume the **else** statement is executed (lines 5–11). In this case, from Lemma 1 it follows that there exists a pair of independent sets I_p and I_q that does not induce either an independent set or a complete bipartite graph. Moreover, since each pair of nodes in I_p (respectively, I_q) has the same colorhood, then I_p (respectively, I_q) contains at least two nodes. It hence follows that line 6 successfully terminates its execution. By the same arguments used in the proof of Lemma 2 it also follows that the partition described at line 7 can be computed and results into an ecological partition for G of size $k+1$. In turn, this corresponds to a $(k+1)$-coloring of G.

Hence, it is now needed to decrease the number of colors. To this aim, the color graph C^{G,r_1} is considered. Observe that, since r_1 is ecological, C^{G,r_1} is a twin-free graph with $k-1$ nodes. From Theorem 3 it follows that $C^{G,r}$ can be ecologically colored with $k-2$ colors: hence, coloring r_2 of line 9 can be computed. Line 10 then computes a new coloring r_3 of G which uses $k-2$ colors as follows: for any node x of G, $r_3(x) = r_2(r_1(x))$. It is easy to prove that r_3 is ecological. Indeed, since all nodes contained in the same independent set have the same colorhood, then for any node $u \in V$ it holds that $C_{r_3}(u) = C_{r_2}(r_1(u))$. As a consequence, since for any $u, v \in V$ such that $r_3(u) \neq r_3(v)$ it holds that $r_2(r_1(u)) \neq r_2(r_1(v))$ and r_2 is ecological, hence $C_{r_3}(u) = C_{r_2}(r_1(u)) \neq C_{r_2}(r_1(v)) = C_{r_3}(v)$. Finally, coloring r is computed by modifying r_3 at line 11. Nodes $u \in I_{p1}$ are assigned color $r(u) = k - 1$ and nodes $u \in I_{q1}$ are assigned color $r(u) = k$; colors of all the remaining nodes are left unchanged. This still results in an ecological coloring for G. In order to prove this last assertion, let us consider two nodes u and v such that $r(u) \neq r(v)$ and prove that $C_r(u) \neq C_r(v)$.

If $u, v \notin I_p \cup I_q$, then the assertion follows since $C_{r_3}(x) \subseteq C_r(x)$ for any $x \notin I_p \cup I_q$. The same reasoning applies if $u, v \in I_p \cup I_q$ and $r_3(u) \neq r_3(v)$. Finally, if $u, v \in I_p \cup I_q$ and $r_3(u) = r_3(v)$, then (without loss of generality) $u \in I_{p1}$ and $v \in I_{q2}$: in this case assume by contradiction that $C_r(u) = C_r(v)$. This might happen only if, in the graph induced by $I_p \cup I_q$, u is adjacent only to nodes in I_{q1} and v is adjacent only to nodes in I_{p2}: this is not possible, since $I_{p1} \cup I_{q1}$ is an independent set. □

4 Conclusions and Open Questions

In this paper we have proved that any graph can be ecologically colored in polynomial-time by making use of any reasonable number of colors. A rough analysis of the coloring algorithm yields a $O(k^3n^2)$ time complexity. Indeed, Algorithm `Partitioning` requires $O(k^3n^2)$ time: this is due to the fact that the most expensive step of this algorithm is searching for the two independent sets I_p and I_q defined in the proof of Lemma 2. The time complexity of Algorithm `Refining`, instead, is dominated either by the execution of line 6 of the algorithm, which requires $O(k^2n^2)$, or by the execution of the application of Theorem 3 to the color graph, which contains $k-1$ nodes: it is easy to verify that this latter step requires time $O(k^4 \log k)$. Hence, the overall time complexity of the coloring algorithm is $O(\max\{k^3n^2, k^2n^2, k^4 \log k\}) = O(k^3n^2)$. We think that, by using more sophisticated data structures, this analysis could be slightly improved.

It is easy to verify that different ecological colorings can be produced for a given graph and for a given number of colors. The ECRA(R) decision problem consists in deciding whether a graph G admits an ecological coloring whose color graph is R; it is easy to prove that ECRA(K_3) is NP-complete. It is then a very interesting open question *to look for a complete classification of the complexity of the* ECRA*(R) problem* similar to the one proposed in [7].

References

1. Borgatti, S.P., Everett, M.G.: The class of regular equivalences: algebraic structure and computation. Social Networks 11, 65–88 (1989)
2. Borgatti, S.P., Everett, M.G.: Graph colorings and power in experimental exchange networks. Social Networks 14, 287–308 (1992)
3. Borgatti, S.P., Everett, M.G.: Ecological and perfect colorings. Social Networks 16, 43–55 (1994)
4. Brandes, U., Erlebach, T. (eds.): Network Analysis: Methodological Foundations. In: Brandes, U., Erlebach, T. (eds.) Network Analysis. LNCS, vol. 3418. Springer, Heidelberg (2005)

5. Breiger, R.L.: The Analysis of Social Networks. In: Hardy, M.A., Bryman, A. (eds.) Handbook of Data Analysis, pp. 505–526. Sage Publications, London (2004)
6. Charon, I., Honkala, I., Hudry, O., Lobstein, A.: Structural Properties of Twin-Free Graphs. The Electronic Journal of Combinatorics 14 (2007)
7. Fiala, J., Paulusma, D.: A complete complexity classification of the role assignment problem. Theoretical Computer Science 349, 67–81 (2005)
8. Kotlov, A., Lovász, L.: The rank and size of graphs. Journal of Graph Theory 23, 185–189 (1996)
9. Lerner, J.: Role assignments. In: ch. 6 [4]
10. Lorrain, F., White, H.C.: Structural equivalence of individuals in social networks. Journal of Mathematical Sociology 1, 49–80 (1971)
11. Roberts, F.S., Sheng, L.: How hard is to determine if a graph has a 2-role assignment? Networks 37(2), 67–73 (2001)
12. White, D.R., Reitz, K.P.: Graph and semigroup homomorphisms on networks of relations. Social Networks 5, 193–234 (1983)

A Proof of Theorem 2

The proof of the theorem immediately follows from the following result.

Lemma 5. *Let $\mathcal{F} = \{N_1, \dots, N_n\}$ be a family of n distinct subsets of $[n]$. Then, there exists $i \in [n]$ such that, for any pair $N_j, N_k \in \mathcal{F}$, $N_j \neq N_k \cup \{i\}$.*

Proof. The proof is by contradiction. Assume that, for any $i \in [n]$, there exists a pair of two distinct sets $L_i, S_i \in \mathcal{F}$ such that $L_i = S_i \cup \{i\}$: if there exist more than one such pairs of sets, then we arbitrarily choose one of them as the only one associated with i. Observe that, for any distinct $i, j \in [n]$, $L_i \neq L_j$ or $S_i \neq S_j$ since otherwise the distinctness between L_i and S_i would imply that $i = j$.

Let us define a directed graph $G_{\mathcal{F}} = (\mathcal{F}, A_{\mathcal{F}})$ as follows. For any $h, k \in [n]$, $(N_h, N_k) \in A_{\mathcal{F}}$ if and only if there exists $i \in [n]$ such that $S_i = N_h$ and $L_i = N_k$. Clearly, $G_{\mathcal{F}}$ is acyclic, since otherwise there would exist a sequence $X_0, \dots, X_{h-1}$ of h distinct subsets of $[n]$ such that $X_i \subset X_{i+1}$ for $0 \leq i < h - 1$ and $X_{h-1} \subset X_0$: this would imply that $X_0 \subset X_0$.

We now prove that also the undirected graph corresponding to $G_{\mathcal{F}}$ does not contain any cycle. This implies that $G_{\mathcal{F}}$ contains at most $n - 1$ arcs: since, for any distinct $i, j \in [n]$, $L_i \neq L_j$ or $S_i \neq S_j$, this contradicts the fact that there must be exactly n arcs in $G_{\mathcal{F}}$, thus proving that there must exist $i \in [n]$ such that, for any pair $N_j, N_k \in \mathcal{F}$, $N_j \neq N_k \cup \{i\}$.

Assume, by contradiction, that the undirected graph corresponding to $G_{\mathcal{F}}$ contains a cycle $X_0, \dots, X_{h-1}$ with $h \geq 3$. Then, there must exist r with $0 \leq r \leq h - 1$ such that $(X_r, X_{r-1}) \in A_{\mathcal{F}} \wedge (X_r, X_{r+1}) \in A_{\mathcal{F}}$ (in the following, we assume that all operations are performed modulo h). Indeed, either $r = 0$ or there exists an incoming arc incident to X_0: in this latter case, we can follow the chain of incoming arcs starting from X_0 and, since G_F is acyclic, we certainly encounter a node X_r with no incoming arcs (see, for example, the cycle in Figure 4 for which $r = 4$).

Let $i \in [n]$ be the element such that $i \in X_{r-1} \wedge i \notin X_r$ and let us prove that $i \in X_s$, for any $s \neq r$ with $0 \leq s \leq h - 1$. This is due to the fact that if $i \in X_t$, for some t

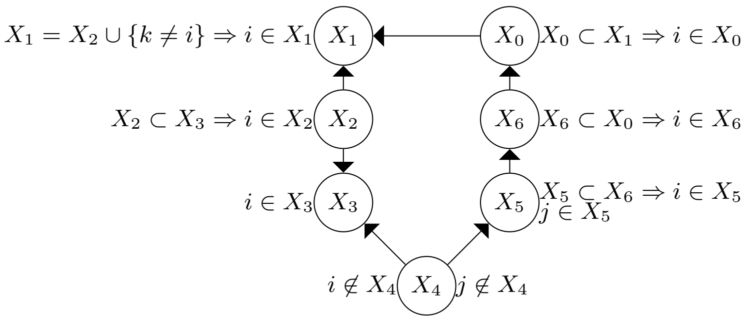

$X_1 = X_2 \cup \{k \neq i\} \Rightarrow i \in X_1$
X_1
X_0
$X_0 \subset X_1 \Rightarrow i \in X_0$
$X_2 \subset X_3 \Rightarrow i \in X_2$
X_2
X_6
$X_6 \subset X_0 \Rightarrow i \in X_6$
$i \in X_3$
X_3
X_5
$X_5 \subset X_6 \Rightarrow i \in X_5$
$j \in X_5$
$i \notin X_4$
X_4
$j \notin X_4$

Fig. 4. The proof of Lemma 5

with $0 \leq t \leq h-1$, and $(X_t, X_s) \in A_{\mathcal{F}}$ or $(X_s, X_t) \in A_{\mathcal{F}}$, then $i \in X_s$. Indeed, if $(X_t, X_s) \in A_{\mathcal{F}}$, then $X_t \subset X_s$; otherwise, if $(X_s, X_t) \in A_{\mathcal{F}}$, then $X_t - X_s \neq \{i\}$ since $s \neq r$ and, for any $i \in [n]$, there exist only two adjacent nodes of $G_{\mathcal{F}}$ whose difference is equal to $\{i\}$. Hence, in both cases we have that $i \in X_s$. We then have that $i \in X_{r+1}$ (see, for example, the cycle in Figure 4 where $i \notin X_4 \wedge i \in X_5$). On the other hand, there must exist $j \neq i$ such that $j \in X_{r+1} \wedge j \notin X_r$: hence, $X_{r+1} - X_r \supseteq \{i, j\}$ which contradicts the fact that $(X_r, X_{r+1}) \in A_{\mathcal{F}}$ (in the example, we have that $X_5 - X_4 \supseteq \{i, j\}$). This completes the proof of the fact that the undirected graph corresponding to $G_{\mathcal{F}}$ is acyclic and, hence, the proof of the lemma. $\square$

Faster Exact Bandwidth[⋆,⋆⋆]

Marek Cygan and Marcin Pilipczuk

Department of Mathematics, Computer Science and Mechanics,
University of Warsaw, Warsaw, Poland
{cygan,malcin}@mimuw.edu.pl

Abstract. We deal with exact algorithms for BANDWIDTH, a long studied NP-hard problem. For a long time nothing better than the trivial $O^*(n!)$ exhaustive search was known. In 2000, Feige an Kilian [4] came up with a $O^*(10^n)$-time algorithm. Since then there has been a growing interest in exponential time algorithms but this bound has not been improved.

In this paper we present a new and quite simple $O^*(5^n)$ algorithm. We also obtain even better bound in some special cases.

1 Introduction

In this paper we focus on exact exponential-time algorithms for the BANDWIDTH problem. Let $G = (V, E)$ be an undirected graph, where $n = |V|$ and $m = |E|$. For a given one-to-one function $\pi : V \to \{1, 2 \ldots, n\}$ (called *ordering*) its *bandwidth* is the maximum difference between positions of adjacent vertices, i.e. $\max_{uv \in E} |\pi(u) - \pi(v)|$. The *bandwidth* of the graph, denoted by $\mathrm{bw}(G)$, is the minimum bandwidth over all orderings. The BANDWIDTH problem asks to find an ordering with bandwidth $\mathrm{bw}(G)$.

BANDWIDTH problem seems to be hard from many perspectives. Although on special families of graphs $\mathrm{bw}(G)$ can be computed in polynomial time [1,6], in general BANDWIDTH is NP-hard even on some subfamilies of trees [5,7]. Moreover Unger [9] showed that BANDWIDTH problem does not belong to APX even in a very restricted case when G is a caterpillar, i.e. a very simple tree. It is also hard for any fixed level of the W hierarchy [2]. The best known polynomial-time approximation, due to Feige [3], has $O(\log^3 n\sqrt{\log n \log\log n})$ approximation guarantee.

From now on, we assume that the input for our problem contains additionally an integer b, $1 \leq b < n$. An ordering of V with bandwidth at most b will be called a *b-ordering*. We focus on checking if there exists a b-ordering and if that is the case, finding it. Note that once we can do it in some time bound T, using binary search we can also find an optimal ordering in $O(T \log \mathrm{bw}(G))$ time. This

[⋆] We would like to thank INFO.RO for sponsoring the WG 2008 Best Student Paper Award that has been awarded to this paper.

[⋆⋆] This research is partially supported by a grant from the Polish Ministry of Science and Higher Education, project N206 005 32/0807.

approach is also used by two exact algorithms for BANDWIDTH problem for general graphs. First of them, due to Saxe [8] is a nontrivial $O(n^{b+1})$-time and -space dynamic programming. It works well when b is small, in particular for $b \leq \frac{n}{\lg n}$ the time becomes $O^*(2^n)$. For arbitrary b, the best result we are aware of is a $O^*(10^n)$-time algorithm due to Feige and Kilian [4].

The main result of this paper is an algorithm for arbitrary b that works in $O^*(5^n)$ time and $O^*(2^n)$ space (see Section 4). Moreover, in Section 3 we present an approach that works more efficiently for large b, in particular we give:

- $O^*(2^{2n-b}) = O(2.83^n)$ time and $O(2^{n-b}) = O(1.42^n)$ space algorithm for $b \geq \frac{n}{2}$,
- $O^*(4^n)$ time and $O(2^b) = O(1.42^n)$ space algorithm for $\frac{n}{3} \leq b < \frac{n}{2}$.

Note that these algorithms approach the problem from the different side than the Saxe's one. Saxe's algorithms works efficiently for small b, whereas presented algorithms cope better with big b.

It is worth mentioning that exponential space in our algorithms is no problem for practical implementations, since in every case space bound is less than square root of the time bound, thus space will not be a bottleneck in a real life applications, at least considering today's proportions of computing speed and fast memory size.

2 Preliminaries

For $v \in V$ by $N(v)$ we denote a set of vertices adjacent to v, analogously for $S \subset V$ we define $N(S) = \bigcup_{v \in S} N(v)$.

We will often view an ordering π as a sequence of vertices $(\pi^{-1}(1), \ldots, \pi^{-1}(n))$. Also, for a given ordering π length of edge uv is $|\pi(u) - \pi(v)|$.

3 Algorithms for $b \geq \frac{n}{3}$

In this section we describe simple algorithms for cases where b is relatively big (comparing to n). Understanding these cases gave us more intuition about BANDWIDTH and enabled us to develop algorithm for arbitrary b.

3.1 Algorithm for $b \geq \frac{n}{2}$

In this section we assume that $b \geq \frac{n}{2}$ and $G = (V, E)$ is an arbitrary undirected graph. Within this limitation we provide $O^*(2^{2n-b})$ time and $O(2^{n-b})$ space algorithm.

The general idea is to consider all partitions of V into V_1 and V_2, $|V_2| = b$, and for each such partition verify whether there exists a b-ordering π with $\pi(v_1) < \pi(v_2)$ for any $v_1 \in V_1$ and $v_2 \in V_2$. Obviously every edge connecting vertices from the same group (V_1 or V_2) is not longer than b since $b \geq \frac{n}{2}$, thus we only need consider edges between V_1 and V_2.

Let us focus on the first group and consider some permutation of V_1 (w.l.o.g. $v_1, v_2, \dots, v_{n-b}$). We would like to have some criterion to check whether there exists some b-ordering with prefix $(v_1, v_2, \dots, v_{n-b})$. To achieve it we claim the following lemma:

Lemma 1. *Assume that $V = V_1 \cup V_2, |V_1| = s, |V_2| = n - s, s \le b, n - s \le b$, then a permutation $(v_1, \dots, v_s)$ of V_1 is a prefix of some b-ordering of G iff for every $1 \le k \le s$ we have $|\bigcup_{i=1}^{k} N(v_i) \setminus V_1| \le k + b - s$.*

Proof. It is easy to see that given condition is necessary, because if $(v_1, \dots, v_s)$ is a prefix of some b-ordering say $(v_1, \dots, v_n)$, then for every $k (1 \le k \le s)$ we have $(\bigcup_{i=1}^{k} N(v_i) \setminus V_1) \subset \{v_{s+1}, v_{s+2}, \dots, v_{k+b}\}$, thus $|\bigcup_{i=1}^{k} N(v_i) \setminus V_1| \le (k + b) - s$.

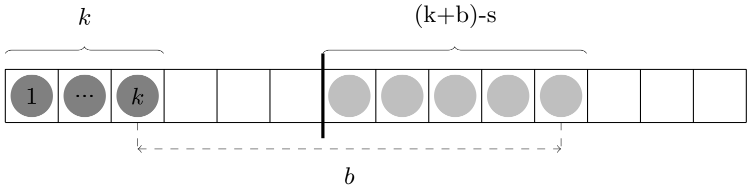


Fig. 1. First k vertices can have at most $(k + b) - s$ neighbors in V_2

To show that given condition is sufficient, assume that the condition holds for every k. Let us define function $\texttt{left} : V \setminus V_1 \to \mathbb{N} \cup \infty$, where for $j > s$ we put $\texttt{left}(v_j) = \min\{i : i \le s \wedge (v_i, v_j) \in E\}$. For v_j not adjacent to any vertex among V_1 we put $\texttt{left}(v_j) = \infty$. In other words, $\texttt{left}(v_j)$ is the index of the leftmost neighbor of v_j in the set $V_1 = \{v_1, v_2, \dots, v_s\}$. We can sort rest of the vertices $V_2 = V \setminus V_1$ according to the function $\texttt{left}$ (breaking ties arbitrarily) and get some ordering (w.l.o.g. $(v_1, \dots, v_n)$) of G. If this is a b-ordering we are done, otherwise let $j_{\min}$ be the minimum j, such that v_j is adjacent to some vertex v_x, where $x > j + b$. As $x \le n$ and $n - s \le b$, so $j \le s$. Since vertices V_2 are sorted according to $\texttt{left}$, thus for every y, $s + 1 \le y \le x$, vertex v_y is adjacent to some vertex among $\{v_1, \dots, v_{j_{\min}}\}$. Therefore $\{v_{s+1}, v_{s+2}, \dots, v_x\} \subset (\bigcup_{i=1}^{j_{\min}} N(v_i) \setminus V_1)$ and we have $|\bigcup_{i=1}^{j_{\min}} N(v_i) \setminus V_1| \ge x - s > j_{\min} + b - s$, contradiction.

As a consequence of Lemma 1 we have the following lemma:

Lemma 2. *Let $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, $|V_2| = b$. There is a $O^*(2^{n-b})$-time and -space algorithm which finds a b-ordering which assigns vertices of V_1 before the vertices of V_2, or states that no such ordering exists.*

Proof. We use dynamic programming over subsets of V_1 to check whether there exists a permutation of V_1, which realizes condition from Lemma 1. More precisely, for every subset $A \subset V_1$ we compute a boolean value $T(A)$ which is true iff vertices from A can be ordered as $(v_1, \dots, v_{|A|})$ in such a way that for all k, $1 \le k \le |A|$, we have $|\bigcup_{i=1}^{k} N(v_i) \setminus V_1| \le (k + b) - (n - b)$.

We note that equivalently, $T(A)$ is true iff $|N(A) \setminus V_1| \leq (|A|+b)-(n-b)$ and for some $v \in A$, $T(A \setminus \{v\})$ is true. This allows for using dynamic programming in $O^*(2^{n-b})$ time. Obviously, using standard techniques we can also *find* the relevant permutation of V_1 if $T(V_1)$ turns out to be true. Then the ordering of V_2 can be found as described in the proof of Lemma 1.

Clearly there are $\binom{n}{n-b} < 2^n$ possible partitions from Lemma 2, which can be found in $O^*(2^n)$ time and polynomial space. Hence we get following conclusion:

Theorem 3. *For $b \geq \frac{n}{2}$ we can find b-ordering in $O^*(2^{2n-b}) = O^*(2^{\frac{3n}{2}}) = O(2.83^n)$ time and $O(2^{n-b}) = O(2^{\frac{n}{2}}) = O(1.42^n)$ space, or state that no such ordering exists.*

3.2 Algorithm for $\frac{n}{3} \leq b < \frac{n}{2}$

In this section we assume that $\frac{n}{3} \leq b < \frac{n}{2}$ and $G = (V, E)$ is an arbitrary undirected graph. Within this limitation we provide $O^*(4^n)$ time and $O(2^b)$ space algorithm.

Here we simply apply Lemma 1 twice. Let $s = \lfloor \frac{n-b}{2} \rfloor$. Let $(v_1, v_2, \ldots, v_n)$ be an ordering of V and let $V_1 = \{v_1, \ldots, v_s\}$, $V_2 = \{v_{s+1}, \ldots, v_{s+b}\}$ and $V_3 = \{v_{s+b+1}, \ldots, v_n\}$. Then $|V_1| = s \leq \frac{n-b}{2} \leq \frac{1}{3}n \leq b$ and $|V_3| = n - b - s \leq \frac{n-b+1}{2} \leq b$, and obviously $|V_2| = b$. Note that the only edges that matter are those between V_2 and $V_1 \cup V_3$ and that if this is a b-ordering, there must be no edge between V_1 and V_3.

Our algorithm generates all such partitions and verifies whether there exists a b-ordering π, with $\pi(v_1) < \pi(v_2) < \pi(v_3)$ for any $v_1 \in V_1$, $v_2 \in V_2$ and $v_3 \in V_3$. From Lemma 1 applied to $V_2 \cup V_3$ and $V_1 \cup V_2$ (in the second case we apply Lemma for reversed ordering), we have the following Corollary:

Corollary 4. *An ordering $(v_{s+1}, v_{s+2}, \ldots, v_{s+b})$ of V_2 is a prefix of some b-ordering of $V_2 \cup V_3$ iff for every k, $1 \leq k \leq b$: $|\bigcup_{i=1}^{k} N(v_{s+i}) \cap V_3| \leq k$.*

The same ordering of V_2 is a suffix of some b-ordering of $V_1 \cup V_2$ iff for every k, $1 \leq k \leq b$: $|\bigcup_{i=0}^{k-1} N(v_{s+b-i}) \cap V_1| \leq k$.

Therefore the ordering of V_2 is a contiguous subsequence from positions $s+1$ to $s+b$ of some b-ordering of V iff both aforementioned conditions are satisfied.

Corollary 4 tells us how to find an ordering compatible with a given partition if one exists. Similarly as in Section 3.1 for every $A \subset V_2$ we compute boolean values $\texttt{pref}(A)$ and $\texttt{suf}(A)$, where $\texttt{pref}(A)$ is true iff $|N(A) \cap V_3| \leq |A|$ and $\texttt{suf}(A)$ is true iff $|N(V_2 \setminus A) \cap V_1| \leq b - |A|$. Then we find a sequence of subsets $\emptyset = A_0 \subset A_1 \subset A_2 \subset \cdots \subset A_b = V_2$ such that for each i, $0 \leq i < b$, we have $|A_{i+1}| - |A_i| = 1$ and for each i, $0 \leq i \leq b$ both $\texttt{pref}(A_i)$ and $\texttt{suf}(A_i)$ are true. This sequence implies the ordering of V_2, $(v_1, \ldots, v_b)$, such that $A_{i+1} \setminus A_i = \{v_i\}$. As before, the ordering of the remaining vertices, i.e. V_1 and V_3 is obtained by

Lemma 1. Clearly, the dynamic programming described above takes $O^*(2^b)$ time and $O(2^b)$ space.

There are at most $\binom{n}{b}\binom{n-b}{s}$ partitions of V, which can be generated in polynomial space; for each of them we use DP with $O(2^b) = O(2^{\frac{n}{2}})$ space and $O^*(2^b)$ time, but $\binom{n}{b}\binom{n-b}{s}2^b \leq 2^n 2^{n-b} 2^b = 4^n$, thus we can claim following theorem:

Theorem 5. *For $\frac{n}{3} \leq b < \frac{n}{2}$ we can find a b-ordering in $O^*(4^n)$ time and $O(2^b) = (1.42^n)$ space, or state that no such ordering exists.*

4 Algorithm $O^*(5^n)$

In this section, we assume that G is a connected undirected graph (if G is not connected we may find b-orderings of each connected component of G in an independent manner).

Imagine that we divide positions $\{1, \ldots, n\}$ into $\lceil \frac{n}{b+1} \rceil$ segments of length roughly $b+1$ elements (if $b+1 \nmid n$ then the last segment has $(n \mod (b+1))$ elements). The first segment contains positions $\{1, \ldots, b+1\}$, the second segment contains positions $\{b+2, \ldots, 2b+2\}$, and so on.

Proposition 6. *In every b-ordering adjacent vertices are either in the same segment or in neighboring segments.*

Our algorithm consists of two phases. During the first phase we generate several assignments of vertices into segments, in such a way that if there exists a b-ordering, its corresponding assignment will certainly be generated. Each of the generated assignments will be considered by the second phase independently.

The above general scheme of our algorithm follows the approach of Feige and Kilian [4]. However, our segments are of length $b+1$ instead of Feige and Kilian's $\frac{b}{2}$ and, more importantly, second phase in our algorithm is completely different from Feige and Kilian's second phase.

4.1 Partitioning V among Segments of Size $b+1$

Let D be any spanning tree of G. Let $(v_1, v_2, \ldots, v_n)$ be any root-to-leaf order of vertices in D, i.e. if v_j is a parent of v_i in D then $j < i$. Note that v_1 is the root of D. We can generate requested assignments in the following way:

1. Place root v_1 in one of the $\lceil \frac{n}{b+1} \rceil$ segments, in every possible way.
2. For every $i = 2, 3, \ldots n$, do:
   - Let v_j be the parent of v_i in D. Since $j < i$, v_j has already been assigned to some segment.
   - Assign v_i to a segment distant by at most one from the segment that v_j has been assigned to, in every possible way.

Proposition 7. *There are at most $3^{n-1}n$ generated assignments.*

4.2 Depth First Search over the Subsets of V

For each assignment generated by the previous phase we would like to check whether there exists a b-ordering compatible with the given assignment. First we check whether for each segment its length equals the number of assigned vertices. Second we check whether there are no edges between segments that are at distance greater than one. If both conditions are satisfied we may proceed further.

Obviously edges between vertices inside the same segment are not important, since each segment has at most $b+1$ elements, thus we may assume that edges in G connect vertices from neighboring segments only.

Now we may assign vertices to each position one by one, but the main idea of our algorithm is the order in which we fill in positions. For every position i, let $\texttt{segment}(i) = \lceil \frac{i}{b+1} \rceil$ be the segment number of this position, and let $\texttt{color}(i) = ((i-1) \mod (b+1))+1$ be the index of the position in its segment, which we will call the *color* of this position. Note that the color of position is the remainder of this number modulo $b+1$, but in the range $[1, b+1]$ instead of $[0, b]$.

Let us sort positions lexicographically according to pairs $(\texttt{color}(i), \texttt{segment}(i))$. To each of those positions we assign a vertex, in exactly this order. We will call this ordering *the color order* of positions.

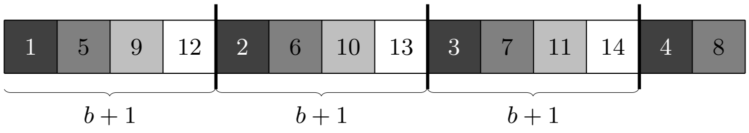


Fig. 2. Color order of positions for $n = 14$ and $b = 3$

The following lemma is the key observation in our algorithm.

Lemma 8. *Ordering π, compatible with the generated segment assignment, is a b-ordering iff for every edge uv if* $\texttt{segment}(\pi(u)) < \texttt{segment}(\pi(v))$ *then* $\texttt{color}(\pi(u)) > \texttt{color}(\pi(v))$.

Proof. Since π is compatible with the generated segment assignment, for every edge uv we have $|\texttt{segment}(\pi(u)) - \texttt{segment}(\pi(v))| \leq 1$. If $\texttt{segment}(\pi(u)) = \texttt{segment}(\pi(v))$ then uv is not longer than b. Otherwise, suppose w.l.o.g. that $\texttt{segment}(\pi(u)) + 1 = \texttt{segment}(\pi(v))$. Note that the distance between positions with the same color in neighboring segments is $b+1$, so uv is not longer than b iff u has greater color than v (see Figure 3).

Corollary 9. *Ordering π, compatible with generated segment assignment, is a b-ordering iff for every edge uv with* $\texttt{segment}(\pi(u)) + 1 = \texttt{segment}(\pi(v))$ *vertex u is assigned to greater position in the color order than vertex v.*

Now we can describe our algorithm.

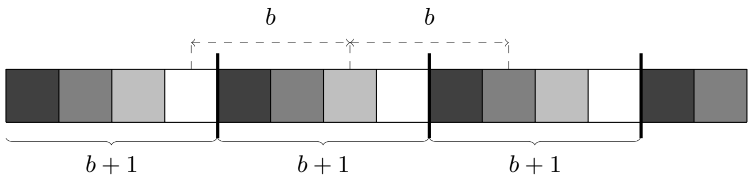


Fig. 3. Picturable proof of the Lemma 8

Definition 10. *By* state *in our algorithm we will denote a subset of vertices $A \subset V$ satisfying:*

- *Vertices of A can be assigned to the first $|A|$ positions in the color order, compatibly with the generated segment assignment.*
- *There is no edge uv with $u \in A$, $v \notin A$ and v assigned to segment with greater number than u.*

Note, that Corollary 9 implies the following lemma.

Lemma 11. *The following equivalence holds:*

1. *Let π be a b-ordering compatible with the generated segment assignment. For every $0 \leq k \leq n$ by A_k we will denote the set of vertices assigned in π to the first k positions in the color order. Then A_k is a* state.
2. *Let $\emptyset = A_0 \subset A_1 \subset \ldots \subset A_n = V$ be a sequence of* states *with $\{v_i\} = A_i \setminus A_{i-1}$. Then ordering π assigning vertex v_i to the position ordered i-th in the color order is a b-ordering.*

Proof. Point 1 is an obvious corollary from condition stated in Corollary 9. In Point 2 note that π is compatible with the generated segment assignment. Suppose that π is not a b-ordering. From Corollary 9 we know that there exists an edge uv with $\texttt{segment}(\pi(u)) + 1 = \texttt{segment}(\pi(v))$ and $\texttt{color}(\pi(u)) \leq \texttt{color}(\pi(v))$. Then u is before v in the color order, so there exists k such that $u \in A_k$ and $v \notin A_k$. However, this contradicts with the assumption that A_k is a *state.*

The algorithm is very simple now. By depth first search we seek for a path of states from the state $\emptyset$ to the state V. Being at state A we try to extend set A by one vertex v from appropriate segment in such a way that $A \cup \{v\}$ is still a state. Note that finding all possible extensions to state A can be done in polynomial time. Lemma 11 ensures that we find such a path iff there exists a b-ordering compatible with the generated segment assignment. Note, that if the algorithm finds the path of states, it can easily reproduce the corresponding b-ordering using the DFS stack.

Note that this algorithm needs $O^*(2^n)$ memory to keep track of visited states. It runs in $O^*(2^n)$ time for every generated assignment. There are at most $3^{n-1}n$ assignments, which leads to $O^*(6^n)$ time bound. In the next section we will prove $O^*(5^n)$ time bound.

4.3 $O^*(5^n)$ Time Bound

In this section we will prove the following theorem:

Theorem 12. *The algorithm described in the previous sections in all generated assignments visits at most* $2n5^{n-1}$ *states.*

Proof. Let us recall the arbitrary spanning tree D which we used to generate all possible segment assignments. Let $v_1, v_2, \ldots, v_n$ be an order in which we assigned vertices to segments, with v_1 being the root of D. Let us look at a state $A \subset V$ in some fixed generated segment assignment. The root v_1 can be assigned into any of $\lceil \frac{n}{b+1} \rceil \leq n$ segments. Every other vertex can be assigned to the same segment as its parent or to one of the neighboring segments. Let us call such vertex *same* if it was assigned to the same segment as its parent, *left* if it was assigned to the segment with smaller positions, and *right* if it was assigned to the segment with greater positions. Moreover, every vertex can be *black* (in state A) or *white* (not in state A).

Let v be a vertex with parent u. Note that if u is *white* ($u \notin A$) then, as A is a state, v cannot be both *black* and *left*. Similarly if v is *black* ($u \in A$) then v cannot be both *white* and *right*.

Therefore every non-root vertex has only 5 possibilities. Since the root of D can be assigned to any segment and be either *white* or *black*, we conclude that our algorithm will visit at most $2n5^{n-1} = O^*(5^n)$ states.

Note, that checking if a subset $A \subset V$ is a state and trying to extend one state in the DFS step can be done in polynomial time. Therefore we can claim the main result of this paper:

Theorem 13. *For arbitrary* b*,* $1 \leq b < n$ *we can find a* b*-ordering or state that it does not exist in* $O^*(5^n)$ *time and* $O^*(2^n)$ *space.*

Acknowledgments. We would like to thank Jakub Radoszewski and Filip Wolski for patience while listening to the very first version of our algorithm. Later on, Lukasz Kowalik and Pawel Gawrychowski gave us a lot of extremely valuable comments and remarks.

References

1. Assman, S., Peck, G., Syslo, M., Zak, J.: The bandwidth of caterpillars with hairs of length 1 and 2. SIAM J. Algebraic Discrete Methods 2, 387–393 (1981)
2. Bodlaender, H.L., Fellows, M.R., Hallett, M.T.: Beyond NP-completeness for problems of bounded width: Hardness for the w hierarchy (extended abstract). ACM Symposium on Theory of Computing, 449–458 (1994)
3. Feige, U.: Approximating the bandwidth via volume respecting embeddings. J. Comput. Syst. Sci. 60(3), 510–539 (2000)
4. Feige, U.: Coping with the NP-hardness of the graph bandwidth problem. In: Halldórsson, M.M. (ed.) SWAT 2000. LNCS, vol. 1851, pp. 10–19. Springer, Heidelberg (2000)

5. Garey, M., Graham, R., Johnson, D., Knuth, D.: Complexity results for bandwidth minimization. SIAM J. Appl. Math. 34, 477–495 (1978)
6. Kleitman, D., Vohra, R.: Computing the bandwidth of interval graphs. SIAM J. Discrete Math. 3, 373–375 (1990)
7. Monien, B.: The bandwidth-minimization problem for caterpillars with hairlength 3 is np-complete. SIAM J. Algebraic Discrete Methods 7, 505–512 (1986)
8. Saxe, J.: Dynamic programming algorithms for recognizing small-bandwidth graphs in polynomial time. SIAM Journal on Algebraic Methods 1, 363–369 (1980)
9. Unger, W.: The complexity of the approximation of the bandwidth problem. In: FOCS, pp. 82–91 (1998)

Additive Spanners for Circle Graphs and Polygonal Graphs*

Feodor F. Dragan[1], Derek G. Corneil[2], Ekkehard Köhler[3], and Yang Xiang[1]

[1] Algorithmic Research Laboratory, Department of Computer Science, Kent State University, Kent, OH 44242, USA
`dragan@cs.kent.edu, yxiang@cs.kent.edu`

[2] Department of Computer Science, University of Toronto, Toronto, Ontario, Canada
`dgc@cs.toronto.edu`

[3] Mathematisches Institut, Brandenburgische Technische Universität Cottbus, D-03013 Cottbus, Germany
`ekoehler@math.TU-Cottbus.DE`

Abstract. A graph $G = (V, E)$ is said to admit a system of μ collective additive tree r-spanners if there is a system $\mathcal{T}(G)$ of at most μ spanning trees of G such that for any two vertices u, v of G a spanning tree $T \in \mathcal{T}(G)$ exists such that the distance in T between u and v is at most r plus their distance in G. In this paper, we examine the problem of finding "small" systems of collective additive tree r-spanners for small values of r on circle graphs and on polygonal graphs. Among other results, we show that every n-vertex circle graph admits a system of at most $2\log_{\frac{3}{2}} n$ collective additive tree 2-spanners and every n-vertex k-polygonal graph admits a system of at most $2\log_{\frac{3}{2}} k+7$ collective additive tree 2-spanners. Moreover, we show that every n-vertex k-polygonal graph admits an additive $(k+6)$-spanner with at most $6n-6$ edges and every n-vertex 3-polygonal graph admits a system of at most 3 collective additive tree 2-spanners and an additive tree 6-spanner. All our collective tree spanners as well as all sparse spanners are constructible in polynomial time.

1 Introduction

A spanning subgraph H of G is called a *spanner* of G if H provides a "good" approximation of the distances in G. More formally, for $r \geq 0$, H is called an *additive r-spanner* of G if for any pair of vertices u and v their distance in H is at most r plus their distance in G [19]. If H is a tree then it is called an *additive tree r-spanner* of G [24]. (A similar definition can be given for multiplicative t-spanners [9, 22, 23] and for multiplicative tree t-spanners [6].) In this paper, we continue the approach taken in [10, 12, 13, 14, 18] of studying *collective tree spanners*. We say that a graph $G = (V, E)$ *admits a system of μ collective additive tree r-spanners* if there is a system $\mathcal{T}(G)$ of at most μ spanning trees of G such that for any two vertices u, v of G a spanning tree $T \in \mathcal{T}(G)$ exists

* This work was supported by the European Regional Development Fund (ERDF) and by NSERC.

such that the distance in T between u and v is at most r plus their distance in G (see [14]). We say that system $\mathcal{T}(G)$ collectively $+r$-spans the graph G and that G is (collectively) $+r$-spanned by $\mathcal{T}(G)$. Clearly, if G admits a system of μ collective additive tree r-spanners, then G admits an additive r-spanner with at most $\mu \times (n-1)$ edges (take the union of all those trees), and if $\mu = 1$ then G admits an additive tree r-spanner.

Collective tree spanners were investigated for a number of particular graph classes, including planar graphs, bounded chordality graphs, bounded genus graphs, bounded treewidth graphs, AT-free graphs and others (see [10, 12, 13, 14, 18]). Some families of graphs admit a constant number and some admit a logarithmic number of collective additive tree r-spanners, for small values of r.

One of the motivations to introduce this concept stems from the problems of designing compact and efficient distance and routing labeling schemes in graphs. A distance labeling scheme for trees is described in [21] that assigns each vertex of an n-vertex tree an $O(\log^2 n)$-bit label such that, given the labels of two vertices x and y, it is possible to compute in constant time, based solely on these two labels, the distance in the tree between x and y. A shortest path routing labeling scheme for trees is described in [27] that assigns each vertex of an n-vertex tree an $O(\log^2 n/\log\log n)$-bit label such that, given the label of a source vertex and the label of a destination, it is possible to compute in constant time, based solely on these two labels, the neighbor of the source that heads in the direction of the destination. Hence, if an n-vertex graph G admits a system of μ collective additive tree r-spanners, then G admits an additive r-approximate distance labeling scheme with the labels of size $O(\mu \log^2 n)$ bits per vertex and an $O(\mu)$ time distance decoder. Furthermore, G admits an additive r-approximate routing labeling scheme with the labels of size $O(\mu \log^2 n/\log\log n)$ bits per vertex. Once computed by the sender in $O(\mu)$ time (by choosing for a given destination an appropriate tree from the collection to perform routing), headers of messages never change, and the routing decision is made in constant time per vertex (see [13, 14]).

Other motivations stem from the generic problems of efficient representation of the distances in "complicated" graphs by the tree distances and of algorithmic use of these representations [1, 2, 5, 16]. Approximating a graph distance d_G by simpler distances (in particular, by tree–distances d_T) is useful in many areas such as communication networks, data analysis, motion planning, image processing, network design, and phylogenetic analysis (see [3,4,6,9,19,20,22,23,25,26]). An arbitrary metric space (in particular a finite metric defined by a graph) might not have enough structure to exploit algorithmically.

In this paper, we examine the problem of finding "small" systems of collective additive tree r-spanners for small values of r on *circle graphs* and on *polygonal graphs*. Circle graphs are known as the intersection graphs of chords in a circle [17]. For any fixed integer $k \geq 2$, the class of k-polygonal graphs can be defined as the intersection graphs of chords inside a convex k-polygon, where the endpoints of each chord lie on two different sides [15]. Note that permutation graphs are exactly 2-polygonal graphs and any n-vertex circle graph is a k-polygonal graph for some $k \leq n$. Our results are the following.

- For any constant c, there are circle graphs that cannot be collectively $+c$-spanned by any constant number of spanning trees.
- Every n-vertex circle graph G admits a system of at most $2\log_{\frac{3}{2}} n$ collective additive tree 2-spanners, constructible in polynomial time.
- There are circle graphs on n vertices for which any system of collective additive tree 1-spanners will require $\Omega(n)$ spanning trees.
- Every n-vertex circle graph admits an additive 2-spanner with at most $O(n \log n)$ edges.
- Every n-vertex k-polygonal graph admits a system of at most $2\log_{\frac{3}{2}} k + 7$ collective additive tree 2-spanners, constructible in polynomial time.
- Every n-vertex k-polygonal graph admits an additive $(k+6)$-spanner with at most $6n-6$ edges and an additive $(k/2+8)$-spanner with at most $10n-10$ edges, constructible in polynomial time.
- Every n-vertex 4-polygonal graph admits a system of at most 5 collective additive tree 2-spanners, constructible in linear time.
- Every n-vertex 3-polygonal graph admits a system of at most 3 collective additive tree 2-spanners and an additive tree 6-spanner, constructible in linear time.

2 Preliminaries

All graphs occurring in this paper are connected, finite, undirected, loopless and without multiple edges. In a graph $G = (V, E)$ the *length* of a path from a vertex v to a vertex u is the number of edges in the path. The *distance* $d_G(u, v)$ between vertices u and v is the length of a shortest path connecting u and v. For a vertex v of G, the sets $N_G(v)$ and $N_G[v] = N_G(v) \cup \{v\}$ are called the *open neighborhood* and the *closed neighborhood* of v, respectively. For a set $S \subseteq V$, by $N_G[S] = \bigcup_{v \in S} N_G[v]$ we denote the closed neighborhood of S and by $G(S)$ the subgraph of G induced by vertices of S. Let also $G \setminus S$ be the graph $G(V \setminus S)$ (which is not necessarily connected).

An graph G is called a *circle graph* if it is the intersection graph of a finite collection of chords of a circle [17] (see Fig. 1 for an illustration). Without loss of generality, we may assume that no two chords share a common endpoint. For any fixed integer $k \geq 3$, the class of *k-polygonal* (or *k-gon*) *graphs* is defined as the intersection graphs of chords inside a convex k-polygon, where the endpoints of each chord lie on two different sides [15] (see Fig. 2 for an illustration). *Permutation graphs* can be considered as 2-gon graphs as they are the intersection graphs of chords between two sides (or sides of a degenerate 2-polygon). Again, without loss of generality, we may assume that no two chords share a common endpoint. Clearly, if a graph G is a k-gon graph, it is also a k'-gon graph with $k' > k$, but the reverse is not necessarily true.

Let $G = (V, E)$ be a permutation graph with a given permutation model Π. Let L' and L'' be the two sides of Π. A vertex s of G is called *extreme* if at least one endpoint of the chord of Π, corresponding to s, is the leftmost or the rightmost endpoint either on L' or on L''. The following result was presented in [13]:

Lemma 1. **[13]** *Let G be a permutation graph and let s be an extreme vertex of G in some permutation model. Then, there exists a $BFS(s)$-tree of G, constructible in linear time, which is an additive tree 2-spanner of G.*

Since an induced cycle on 4 vertices is a permutation graph, permutation graphs cannot have any additive tree r-spanner for $r < 2$. Clearly, given any $BFS(s)$-tree T_s of G, $d_{T_s}(x, s) = d_G(x, s)$ holds for any $x \in V$.

3 Additive Spanners for Circle Graphs

In this section, we show that every n-vertex circle graph G admits a system of at most $2\log_{\frac{3}{2}} n$ collective additive tree 2-spanners. This upper bound result is complemented also with two lower bound results.

We start with the main lemma of this section which is also of independent interest.

Lemma 2. *Every n-vertex ($n \geq 2$) circle graph $G = (V, E)$ has two vertices a and b such that $S = N_G[a, b]$ is a balanced separator of G, i.e. each connected component of $G \setminus S$ has at most $\frac{2}{3}n$ vertices.*

Proof. Consider an intersection model $\phi(G)$ of G and let $\mathcal{C}$ be the circle in that model. Let also $\mathcal{P} := (p_1, p_2, \ldots, p_{2n})$ be the sequence in clockwise order of the $2n$ endpoints of the chords representing the vertices of G in $\phi(G)$. We divide the circle $\mathcal{C}$ into three circular arcs B (bottom), L (left) and R (right) each containing at most $\lceil\frac{2}{3}n\rceil$ consecutive endpoints (see Fig. 1 for an illustration). We say that a chord of $\phi(G)$ is an XY-chord if its endpoints lie on arcs X and Y ($X, Y \in \{B, L, R\}$) of $\mathcal{C}$. If v is an XY-chord then let v_X and v_Y be its endpoints on X and Y, respectively.

Let X be an arc from the set of arcs $\{B, L, R\}$. Since G is a connected graph, for any X, there must exist a chord in $\phi(G)$ with one endpoint in X and the other endpoint not in X. Moreover, since we have three arcs (B, L, R), there must exist an arc X in $\{B, L, R\}$ which has both types of chords: between X and $Y \in \{B, L, R\} \setminus \{X\}$ and between X and $Z \in \{B, L, R\} \setminus \{X, Y\}$. Assume, without loss of generality, that $X = B$. Let p be the point of $\mathcal{C}$ separating arcs L and R (see Fig. 1). Now choose a BL-chord a in $\phi(G)$ with endpoint a_L closest to p and choose a BR-chord b in $\phi(G)$ with endpoint b_R closest to p. By a, b we also denote the vertices of G which correspond to chords a and b.

Points a_B, a_L, b_R and b_B of $\mathcal{C}$ divide $\mathcal{C}$ into four arcs. We name these four arcs A_U, A_R, A_D and A_L. The arc $A_U := (a_L, b_R)$ is formed by all points of $\mathcal{C}$ from a_L to b_R in clockwise order. If chords a and b intersect, then we set $A_R := (b_R, a_B)$, $A_D := (a_B, b_B)$, and $A_L := (b_B, a_L)$ (all arcs begin at the left arc-endpoint and go clockwise to the right arc-endpoint). If chords a and b do not intersect, then

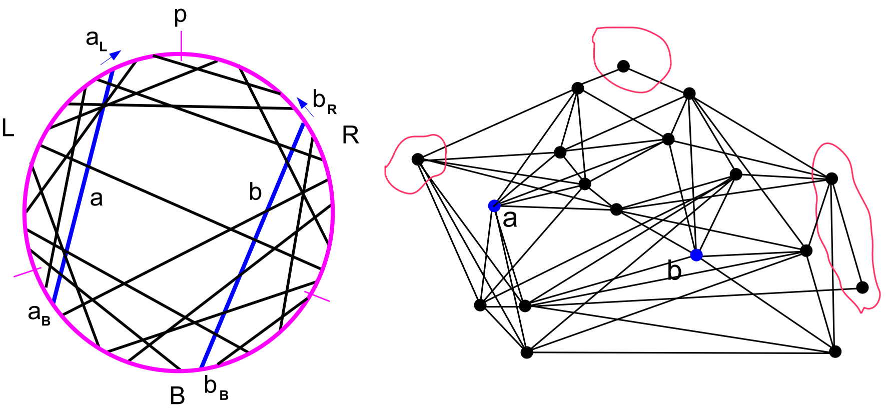


Fig. 1. A circle graph with an intersection model and two special chords a and b. A balanced separator $S = N_G[a, b]$ and the connected components of $G \backslash S$ are also shown.

set $A_R := (b_R, b_B)$, $A_D := (b_B, a_B)$, and $A_L := (a_B, a_L)$. We consider these arcs as open arcs, i.e., the points a_B, a_L, b_R and b_B do not belong to them.

By our choices of a and b, we guarantee that $\phi(G)$ has no chords with one endpoint in A_U and the other one in A_D (regardless of the adjacency of a and b). Denote by V_Y all chords from $\phi(G)$ (vertices of G) whose both endpoints are in A_Y, where Y is either U, or R, or D, or L. Then, it is easy to see that in G, the set $S := N_G[a, b]$ separates vertices of V_Y from vertices of $V_{Y'}$, where $Y, Y' \in \{U, R, D, L\}$, $Y \neq Y'$. Now, since A_L is a sub-arc of arc $B \cup L$, A_U is a sub-arc of arc $L \cup R$, A_R is a sub-arc of arc $R \cup B$, A_D is a sub-arc of arc B, and arcs A_U, A_R, A_D and A_L do not contain points a_B, a_L, b_R and b_B, we conclude that $|A_L \cap \mathcal{P}| \leq \frac{4}{3}n$, $|A_U \cap \mathcal{P}| \leq \frac{4}{3}n$, $|A_R \cap \mathcal{P}| \leq \frac{4}{3}n$ and $|A_D \cap \mathcal{P}| \leq \frac{2}{3}n$. Hence, the number of arcs in $\phi(G)$ whose both endpoints are in A_L (respectively, in A_U, A_R, A_D), and therefore the number of vertices in V_L ((respectively, in V_U, V_R, V_D), is at most $\frac{2}{3}n$. □

In [12], a large class of graphs, called *(α, γ, r)-decomposable graphs*, was defined, and it was proven that any (α, γ, r)-decomposable graph G with n vertices admits a system of at most $\gamma \log_{1/\alpha} n$ collective additive tree $2r$-spanners. Let α be a positive real number smaller than 1, γ be a positive integer and r be a non-negative integer. We say that an n-vertex graph G is (α, γ, r)*–decomposable* if $n \leq \gamma$ or there is a separator $S \subseteq V$ in G, such that the following three conditions hold:

- the removal of S from G leaves no connected component with more than αn vertices;
- there exists a subset $D \subseteq V$ such that $|D| \leq \gamma$ and for any vertex $u \in S$, $d_G(u, D) \leq r$;
- each connected component of $G \setminus S$ is an (α, γ, r)–decomposable graph, too.

Since, any subgraph of a circle graph is also a circle graph, and, by Lemma 2, each n-vertex circle graph $G = (V, E)$ admits a separator $S = N_G[D]$ (where

$D = \{a, b\}$, $a, b \in V$), such that no connected component of $G \setminus S$ has more than $\frac{2}{3}n$ vertices, we immediately conclude.

Corollary 1. *Every circle graph is* $(\frac{2}{3}, 2, 1)-$*decomposable.*

Theorem 1. *Every n-vertex circle graph G admits a system $\mathcal{T}(G)$ of at most* $2\log_{\frac{3}{2}} n$ *collective additive tree 2-spanners.*

Note that such a system of spanning trees $\mathcal{T}(G)$ for a n-vertex m-edge circle graph G, given together with an intersection model $\phi(G)$, can be constructed in $O(m \log n)$ time, since a balanced separator $S = N_G[a, b]$ of G can be found in linear $O(m)$ time (see [12] for details of the construction).

Taking the union of all these spanning trees in $\mathcal{T}(G)$, we also obtain a sparse additive 2-spanner for a circle graph G.

Corollary 2. *Every n-vertex circle graph G admits an additive 2-spanner with at most* $2(n-1)\log_{\frac{3}{2}} n$ *edges.*

We complement our upper bound result with the following lower bounds.

Proposition 1. *There are circle graphs on n vertices for which any system of collective additive tree 1-spanners will require $\Omega(n)$ spanning trees.*

Proof. Since complete bipartite graphs are circle graphs, we can use the lower bound shown in [14] for complete bipartite graphs. It says that any system of collective additive tree 1-spanners will need to have $\Omega(n)$ spanning trees for each complete bipartite graph on n vertices. □

Proposition 2. *For any constant c, there are circle graphs that cannot be collectively $+c$-spanned by any constant number of spanning trees.*

In [7] the authors show that a similar proposition holds for weakly chordal graphs. In fact, the same proof works for circle graphs.

4 Additive Spanners for *k*-Gon and for 3-Gon Graphs

4.1 Additive Spanners for *k*-Gon Graphs

In this section, among other results, we show that every n-vertex k-gon graph G admits a system of at most $2\log_{\frac{3}{2}} k + 7$ collective additive tree 2-spanners, an additive $(k+6)$-spanner with at most $6n-6$ edges, and an additive $(k/2+8)$-spanner with at most $10n-10$ edges. We will assume, in what follows, that our k-gon graph G is given together with its intersection model. Let $\mathcal{P}$ be the closed polygonal chain (the boundary) of the k-polygon in that model. The vertices of the k-polygon will be called the *corners*. The idea of the construction here is similar to the one used in Theorem 1. Yet, here we operate with the corners of the model rather than with the endpoints of the chords. More precisely, one can show that there are vertices a and b in the graph, such that $N_G[a, b]$ forms a

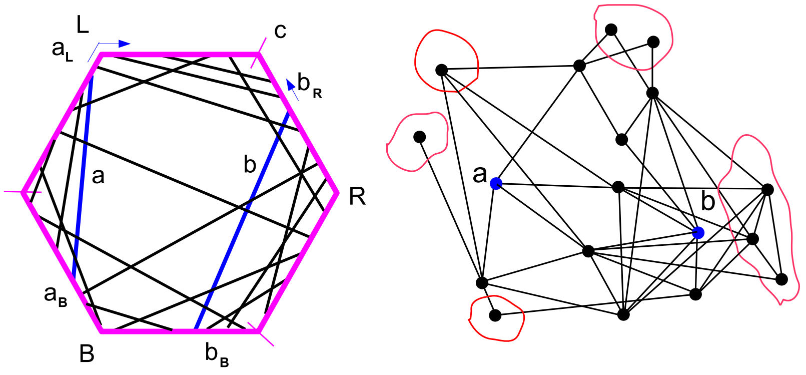


Fig. 2. A 6-gon graph with an intersection model and two special chords a and b. A balanced separator $S = N_G[a, b]$ and the connected components of $G \backslash S$ are also shown.

balanced (with respect to the number of corners) separator of G, similar to the one in Lemma 2 (see Fig. 2 for an illustration).

It is an easy observation that each of the so-created connected components is itself again a k'-gon graph for some $k' < k$. By applying this separator property recursively, we end up with a set of separators and some permutation graphs. By Lemma 1, permutation graphs have good additive tree spanners. We can put these trees together appropriately and can thereby show the following theorem (see [11] for details).

Theorem 2. *Every n-vertex m-edge k-gon graph G admits a system of at most $2\log_{\frac{3}{2}} k + 7$ collective additive tree 2-spanners, constructable in $O(m \log k)$ time. Moreover, every 3-gon graph admits a system of no more than 3 collective additive tree 2-spanners, and every 4-gon graph admits a system of no more than 5 collective additive tree 2-spanners.*

Similar to Corollary 2, we can merge the edges of all the spanning trees to create a single spanning graph.

Corollary 3. *Every n-vertex k-gon graph G admits an additive 2-spanner with at most $(2\log_{\frac{3}{2}} k + 7)(n - 1)$ edges. Moreover, every 3-gon graph admits an additive 2-spanner with at most $3(n - 1)$ edges, and every 4-gon graph admits an additive 2-spanner with at most $5(n - 1)$ edges.*

Instead of using the recursive separation algorithm all the way to permutation graphs, one can also stop at an earlier stage. By using properties of k-gon graphs (see [15]) and some general results on spanners of graphs with bounded length of largest induced cycle (see [8]) we can show the following theorem.

Theorem 3. *Every n-vertex m-edge k-gon graph G admits an additive $(2((\frac{2}{3})^\ell k + 4(1 - (\frac{2}{3})^\ell)) + 1)$-spanner with at most $2(\ell + 1)(n - 1)$ edges, for each $0 \leq \ell \leq \log_{3/2} k + 3$. Moreover, such a sparse spanner is constructable in $O(m \log k)$ time.*

Finally, when choosing ℓ equal to $0, 1, 2, 3$ or 4 in Theorem 3, we obtain:

Corollary 4. *Every n-vertex k-gon graph G admits an additive $(2k+1)$-spanner with at most $2n-2$ edges, an additive $(\frac{4}{3}k+4)$-spanner with at most $4n-4$ edges, an additive $(\frac{8}{9}k+6)$-spanner with at most $6n-6$ edges, an additive $(\frac{16}{27}k+7)$-spanner with at most $8n-8$ edges, and an additive $(\frac{32}{81}k+8)$-spanner with at most $10n-10$ edges.*

4.2 Additive Tree Spanners for 3-Gon Graphs

In this section, we show that any connected 3-gon graph G admits an additive tree 6-spanner constructible in linear time. Due to space restrictions we have to omit some parts of the proof here, and refer instead the reader to [11] for a complete proof of this result. Note that, since an induced cycle on 6 vertices is a 3-gon graph, 3-gon graphs cannot have any additive tree r-spanner for $r < 4$. The idea of the construction is as follows. The algorithm will identify permutation graphs in each of the 3 corners of the 3-gon and use the algorithm presented in Lemma 1 to construct effective tree spanners of each of these subgraphs. These 3 tree spanners are incorporated into a tree spanner for the entire graph by analyzing the structure in the "center" of the given 3-gon graph.

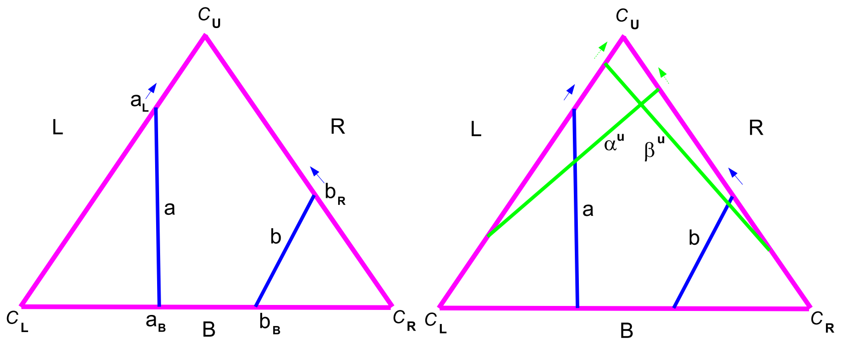


Fig. 3. A 3-gon intersection model Δ with special chords a, b, α^u and β^u

Let $G = (V, E)$ be a connected 3-gon graph. We may assume that G is not a permutation graph. Consider a 3-gon intersection model Δ of G and fix an orientation of Δ. Denote by L (left), R (right) and B (bottom) the corresponding sides of the 3-gon Δ, and by C_L, C_R and C_U the left, right and upper corners of Δ. We say that a chord of Δ is an XY-chord if its endpoints lie on sides X and Y of Δ. If v is an XY-chord then let v_X and v_Y be its endpoints on X and Y, respectively. Since G is not a permutation graph, we must have all three types of chords in Δ: LR-chords, LB-chords and RB-chords. Let a be the LB-chord of G whose endpoint on L is closest to the upper corner C_U of Δ. Let b be the RB-chord of G whose endpoint on R is closest to the upper corner of Δ (see the left 3-gon in Fig. 3 for an illustration). Note that a and b may or may not cross. By a, b we also denote the corresponding vertices of G.

Let V_U be the subset of LR-chords of Δ (of vertices of G) with endpoints in segments (a_L, C_U) and (b_R, C_U). We will add at most two more LR-chords to V_U to form a permutation graph named G_U. Choose (if it exists) an LR-chord α^u in Δ such that α^u_L belongs to segment (C_L, a_L) of L, α^u_R belongs to segment (C_U, b_R) of R and α^u_R is closest to the corner C_U. Clearly, if α^u exists then it must intersect a (but not b). Analogously, choose (if it exists) an LR-chord β^u in Δ such that β^u_R belongs to segment (C_R, b_R) of R, β^u_L belongs to segment (C_U, a_L) of L and β^u_L is closest to the corner C_U. Again, if β^u exists then it must intersect b (but not a). Note that, if $V_U \neq \emptyset$, then at least one of $\{\alpha^u, \beta^u\}$ must exist (since otherwise, G is not connected), and if both chords exist then they must intersect each other. See the right picture in Fig. 3. Now, we define our permutation graph G_U to be the subgraph of G induced by vertices $V_U \cup \{\alpha^u, \beta^u\}$, assuming that $V_U \neq \emptyset$ (see Fig. 4 for an illustration). If $V_U = \emptyset$, then we set G_U to be an empty graph.

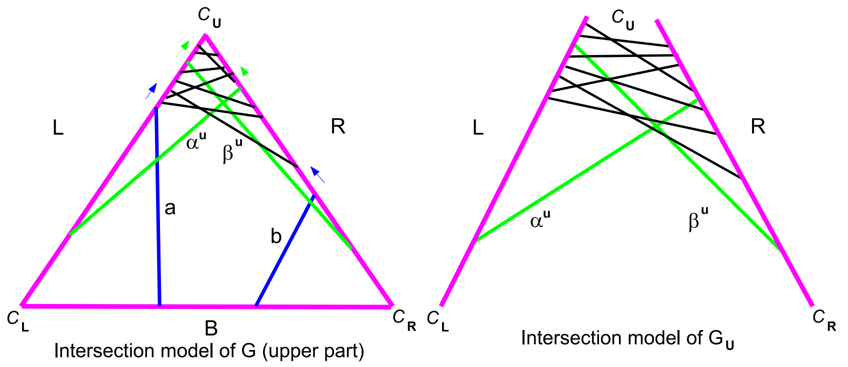

C_U
L
R
α^u
β^u
a
b
C_L
B
C_R
Intersection model of G (upper part)
C_U
L
R
α^u
β^u
C_L
C_R
Intersection model of G_U

Fig. 4. Permutation graph G_U extracted from G

Define s^u to be a vertex from $\{\alpha^u, \beta^u\}$ as follows: if both α^u and β^u exist, then if α^u has a neighbor in V_U which is not a neighbor of β^u, set $s^u := \alpha^u$; otherwise, set $s^u := \beta^u$. Since G_U is a permutation graph and s^u is extreme, by Lemma 1, G_U has a linear time constructable $BFS(s^u)$-tree T_U such that $d_{T_U}(x, y) \leq d_{G_U}(x, y) + 2$ and $d_{T_U}(x, s^u) = d_{G_U}(x, s^u)$ for any x, y in $V_U \cup \{\alpha^u, \beta^u\}$.

Let V_L be the subset of all chords of Δ (of vertices of G) with endpoints in segments (C_L, a_L) and $(C_L, a_B) \cap (C_L, b_B)$. We will add at most two more chords to V_L to form a permutation graph named G_L. Choose (if it exists) a chord α^ℓ in Δ such that one endpoint of α^ℓ belongs to segment (C_L, a_L) of L, the other endpoint belongs to $R \cup (a_B, C_R) \cup (b_B, C_R)$ and α^ℓ_L is closest to the corner C_L. Equivalently, among all chords of Δ intersecting a or b, α^ℓ is chosen to be the chord with an endpoint α^ℓ_L in (C_L, a_L) closest to C_L. Note that α^ℓ may or may not cross b. Also, choose (if it exists) an RB-chord β^ℓ in Δ such that β^ℓ_R belongs to segment (C_R, b_R) of R, β^ℓ_B belongs to segment $(C_L, a_B) \cap (C_L, b_B)$ of B and

β^ℓ_B is closest to the corner C_L. Notice, if β^ℓ exists then it must intersect both a and b. Furthermore, if $V_L \neq \emptyset$, then at least one chord from $\{\alpha^\ell, \beta^\ell\}$ must exist (since, otherwise, G is not connected). Now, we define our permutation graph G_L. If $V_L = \emptyset$, then set G_L to be an empty graph. Otherwise, G_L is set to be the subgraph of G induced by vertices $V_L \cup \{\alpha^\ell, \beta^\ell\}$ with one extra edge $(\alpha^\ell, \beta^\ell)$ added if it was not already an edge of G (see Fig. 5 for an illustration).

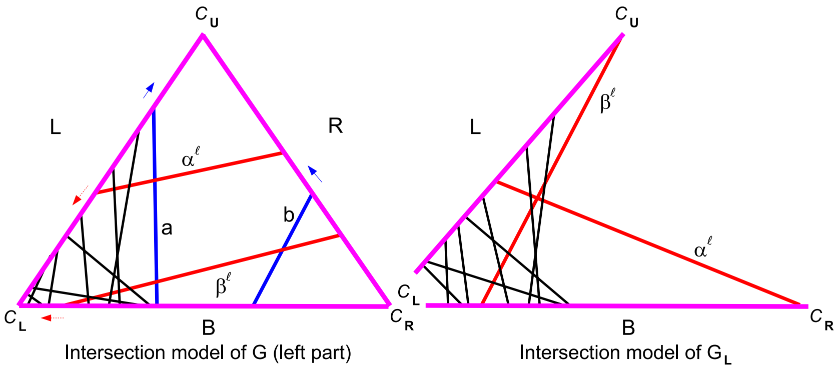


Fig. 5. Permutation graph G_L obtained from G

Define s^ℓ to be a vertex from $\{\alpha^\ell, \beta^\ell\}$ as follows: if both α^ℓ and β^ℓ exist, then if α^ℓ has a neighbor in V_L which is not a neighbor of β^ℓ, then set $s^\ell := \alpha^\ell$; otherwise, set $s^\ell := \beta^\ell$. Since G_L is a permutation graph, by Lemma 1, G_L has a linear time constructable $BFS(s^\ell)$-tree T_L such that $d_{T_L}(x, y) \leq d_{G_L}(x, y) + 2$ and $d_{T_L}(x, s^\ell) = d_{G_L}(x, s^\ell)$ for any x, y in $V_L \cup \{\alpha^\ell, \beta^\ell\}$.

Taking symmetry into account, similar to α^ℓ, β^ℓ and G_L, we can define for the corner C_R of Δ two specific chords α^r, β^r and a permutation graph G_R. We will have β^r adjacent to both a and b, and α^r adjacent to a or b. Define s^r to be a vertex from $\{\alpha^r, \beta^r\}$, and if both α^r and β^r exist, then if α^r has a neighbor in V_R which is not a neighbor of β^r, then set $s^r := \alpha^r$; otherwise, set $s^r := \beta^r$. Again, by Lemma 1, there is a linear time constructable $BFS(s^r)$-tree T_R of G_R such that $d_{T_R}(x, y) \leq d_G(x, y) + 2$ and $d_{T_R}(x, s^r) = d_G(x, s^r)$ for any x, y in $V_R \cup \{\alpha^r, \beta^r\}$.

Now we create a spanning tree T of G from the trees T_U, T_L and T_R as follows. Initially, T is just the union of T_U, T_L and T_R. We know that $\{\beta^\ell, \beta^r, \alpha^u\} \subseteq N_G(a)$ and $\{\beta^\ell, \beta^r, \beta^u\} \subseteq N_G(b)$. Make vertex a adjacent to α^u and vertex b adjacent to β^u in T, and denote $M := \{\alpha^\ell, \beta^\ell, \alpha^r, \beta^r\}$. If $M \subseteq N_G(a)$, then make vertex a adjacent in T to each vertex in M. If $M \setminus N_G(a) \neq \emptyset$ but $M \subseteq N_G(b)$, then make vertex b adjacent in T to each vertex in M. If neither $M \subseteq N_G(a)$ nor $M \subseteq N_G(b)$, then make vertices α^ℓ, β^ℓ adjacent in T to a common neighbor in $\{a, b\}$ and vertices α^r, β^r adjacent in T to a common neighbor in $\{a, b\}$. Remove from T the edge $(\alpha^\ell, \beta^\ell)$ (it was a part of tree T_L if both α^ℓ and β^ℓ existed) and the edge (α^r, β^r) (it was a part of tree T_R if both α^r and β^r existed).

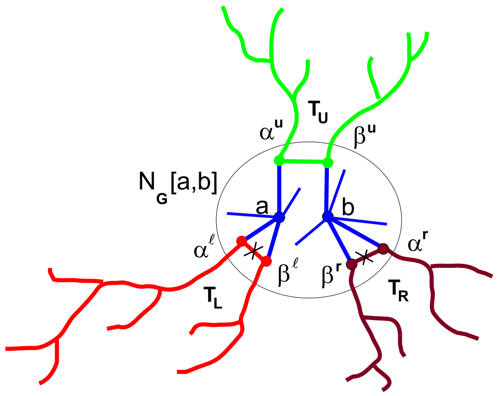


Fig. 6. Trees T_L, T_R and T_U connected via $N_G[a,b]$ to form a tree spanner of G

If a and b are adjacent in G, then add edge (a,b) to T. If a is not adjacent to b in G but $d_G(a,b)=2$, then choose a common neighbor z of a and b in $N_G[a,b]$ and add edges (a,z) and (b,z) to T. In these cases, i.e., when $d_G(a,b) \leq 2$, remove the possible edge (α^u, β^u) from T (it was a part of the tree T_U if both α^u and β^u existed). If $d_G(a,b) > 2$ then $d_G(a,b)=3$, chords β^ℓ, β^r do not exist and the edge (α^u, β^u) from T_U goes to T if both chords α^u and β^u exist. If one of these chords does not exist, then there must be two vertices x, y that are adjacent in G, with $x \in N_G(b)$ and $y \in N_G(a)$ (see [11] for details) and we put the edge (x,y) into T. Finally, make all vertices from $N_G(a) \setminus \{\alpha^\ell, \beta^\ell, \alpha^u, \beta^u, \alpha^r, \beta^r, b, z\}$ adjacent to a in T and all remaining vertices from $N_G(b)$ (i.e., those that are not adjacent to a in T) adjacent to b; see Fig. 6 for an illustration. Clearly, T constructed this way is a spanning tree of G. Using a careful analysis, we can show that T is an additive tree 6-spanner of G (see [11] for the complete proof).

Theorem 4. *Any connected 3-gon graph G admits an additive tree 6-spanner constructible in linear time.*

References

1. Bartal, Y.: Probabilistic approximations of metric spaces and its algorithmic applications. In: FOCS 1996, pp. 184–193 (1996)
2. Bartal, Y.: On approximating arbitrary metrices by tree metrics. In: STOC 1998, pp. 161–168 (1998)
3. Barthélemy, J.-P., Guénoche, A.: Trees and Proximity Representations. Wiley, New York (1991)
4. Bhatt, S., Chung, F., Leighton, F., Rosenberg, A.: Optimal simulations of tree machines. In: FOCS 1986, pp. 274–282 (1986)
5. Charikar, M., Chekuri, C., Goel, A., Guha, S., Plotkin, S.: Approximating a Finite Metric by a Small Number of Tree Metrics. In: FOCS 1998, pp. 379–388 (1998)
6. Cai, L., Corneil, D.G.: Tree spanners. SIAM J. Discrete Math. 8, 359–387 (1995)

7. Corneil, D.G., Dragan, F.F., Köhler, E., Xiang, Y.: Lower Bounds for Collective Additive Tree Spanners (in preparation)
8. Chepoi, V.D., Dragan, F.F., Yan, C.: Additive Sparse Spanners for Graphs with Bounded Length of Largest Induced Cycle. Theoretical Computer Science 347, 54–75 (2005)
9. Chew, L.P.: There are planar graphs almost as good as the complete graph. J. of Computer and System Sciences 39, 205–219 (1989)
10. Corneil, D.G., Dragan, F.F., Köhler, E., Yan, C.: Collective tree 1-spanners for interval graphs. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 151–162. Springer, Heidelberg (2005)
11. Dragan, F.F., Corneil, D.G., Köhler, E., Xiang, Y.: Additive Spanners for Circle Graphs and Polygonal Graphs (manuscript, 2008), `http://www.cs.kent.edu/~dragan/Coll-Spanners-Circle.pdf`
12. Dragan, F.F., Yan, C.: Collective Tree Spanners in Graphs with Bounded Genus, Chordality, Tree-width, or Clique-width. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 583–592. Springer, Heidelberg (2005)
13. Dragan, F.F., Yan, C., Corneil, D.G.: Collective Tree Spanners and Routing in AT-free Related Graphs. Journal of Graph Algorithms and Applications 10, 97–122 (2006)
14. Dragan, F.F., Yan, C., Lomonosov, I.: Collective tree spanners of graphs. SIAM J. Discrete Math. 20, 241–260 (2006)
15. Elmallah, E.S., Stewart, L.: Polygon Graph Recognition. Journal of Algorithms 26, 101–140 (1998)
16. Fakcharoenphol, J., Rao, S., Talwar, K.: A tight bound on approximating arbitrary metrics by tree metrics. In: STOC 2003, pp. 448–455 (2003)
17. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs, 2nd edn. Elsevier, Amsterdam (2004)
18. Gupta, A., Kumar, A., Rastogi, R.: Traveling with a Pez Dispenser (or, Routing Issues in MPLS). SIAM J. Comput. 34, 453–474 (2005)
19. Liestman, A.L., Shermer, T.: Additive graph spanners. Networks 23, 343–364 (1993)
20. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. SIAM Monographs on Discrete Math. Appl. SIAM, Philadelphia (2000)
21. Peleg, D.: Proximity-Preserving Labeling Schemes and Their Applications. J. of Graph Theory 33, 167–176 (2000)
22. Peleg, D., Schäffer, A.A.: Graph Spanners. J. Graph Theory 13, 99–116 (1989)
23. Peleg, D., Ullman, J.D.: An optimal synchronizer for the hypercube. In: PODC 1987, pp. 77–85 (1987)
24. Prisner, E.: Distance approximating spanning trees. In: Reischuk, R., Morvan, M. (eds.) STACS 1997. LNCS, vol. 1200, pp. 499–510. Springer, Heidelberg (1997)
25. Sneath, P.H.A., Sokal, R.R.: Numerical Taxonomy. W.H. Freeman, San Francisco (1973)
26. Swofford, D.L., Olsen, G.J.: Phylogeny reconstruction. In: Hillis, D.M., Moritz, C. (eds.) Molecular Systematics, pp. 411–501. Sinauer Associates Inc., Sunderland (1990)
27. Thorup, M., Zwick, U.: Compact routing schemes. In: SPAA 2001, pp. 1–10 (2001)

Upward Straight-Line Embeddings of Directed Graphs into Point Sets

Alejandro Estrella-Balderrama[1], Fabrizio Frati[2], and Stephen G. Kobourov[1]

[1] Department of Computer Science – University of Arizona, U.S.A.
{aestrell,kobourov}@cs.arizona.edu
[2] Dipartimento di Informatica e Automazione – Università di Roma Tre
frati@dia.uniroma3.it

Abstract. In this paper we consider the problem of characterizing the directed graphs that admit an upward straight-line embedding into every point set in convex or in general position. In particular, we show that no biconnected directed graph admits an upward straight-line embedding into every point set in convex position, and we provide a characterization of the Hamiltonian directed graphs that admit upward straight-line embeddings into every point set in general or in convex position. We also describe how to construct upward straight-line embeddings of directed paths into convex point sets and we prove that for directed trees such embeddings do not always exist. Further, we investigate the related problem of upward simultaneous embedding without mapping, proving that deciding whether two directed graphs admit an upward simultaneous embedding without mapping is $\mathcal{NP}$-hard.

1 Introduction

A *straight-line embedding* of a graph into a point set P is a mapping of each vertex to a point of P and of each edge to a straight-line segment between its end-points such that no two edges intersect. The problem of constructing straight-line embeddings of graphs into planar point sets is well-studied from both a combinatorial and an algorithmic point of view and comes in several different flavours within the Graph Drawing literature.

Gritzmann *et al.* [12] proved that a graph admits a straight-line embedding into *every* point set in general position if and only if it is an *outerplanar graph*. From an algorithmic point of view, an $O(n \log^3 n)$-time algorithm [1] and a $\Theta(n \log n)$-time algorithm [2] are known for constructing straight-line embeddings of outerplanar graphs and trees into given point sets in general position, respectively. Cabello [4] proved that the problem of deciding whether a planar graph admits a straight-line embedding into a given point set is $\mathcal{NP}$-hard. If edges are not required to be straight, then by the results of Kaufmann and Wiese [13], every planar graph admits a planar drawing with at most two bends per edge into every point set and such a bound can not be improved.

Several problems ask for embedding more graphs on the same point set. Determining the minimum cardinality $f(n)$ of a set of points P such that every n-vertex planar graph admits a straight-line embedding into an n-point subset of P is a well-know and widely-open problem [5,6,14,15]. Recently, the problem of constructing simultaneous embeddings without mapping, i.e., straight-line planar drawings of n-vertex graphs on the same set of n points, has been considered. Brass *et al.* [3] proved that a planar graph and any number of outerplanar graphs admit a simultaneous embedding without mapping. Whether every two planar graphs admit a simultaneous embedding without mapping is still unknown.

Surprisingly, less attention has been devoted to the directed versions of such problems. When visualizing a directed graph, one usually requires an *upward* drawing, i.e., a drawing such that each edge monotonically increases in the y-direction. Giordano *et al.* [11] show a directed counterpart of the results in [13], namely that every upward planar digraph has an upward planar embedding with at most two bends per edge into every point set. As a directed counterpart of the results in [3], the same authors show that any number of trees admit an upward simultaneous embedding without mapping.

In this paper we study additional directed versions of some of the problems cited above. In Section 3, we consider the problem of determining which directed graphs admit a planar straight-line upward embedding into every point set in general or in convex position. We show that no biconnected directed graph with more than three vertices has a straight-line upward embedding into every point set in convex position. We also characterize the Hamiltonian directed graphs that admit a straight-line upward embedding into every point set in convex or in general position. We prove that every directed path admits a straight-line upward embedding into every point set in convex position and that every directed tree of diameter at most four admits a straight-line upward embedding into every point set in general position. Finally, we prove that not all directed trees admit a straight-line upward embedding into every point set in convex position.

In Section 4, answering a question of Giordano *et al.* [11], we show two upward planar directed graphs that do not admit any upward simultaneous embedding without mapping. Further, we prove that deciding if two upward planar digraphs admit an upward simultaneous embedding without mapping is $\mathcal{NP}$-hard.

Several proofs are omitted or sketched, due to space limitations. Full proofs of each statement can be found in the extended version of the paper [9].

2 Preliminaries

An *upward planar directed graph* is a directed graph that admits a planar drawing such that each edge is represented by a curve monotonically increasing in the y-direction. Every upward planar digraph admits a *straight-line* upward planar drawing [7], i.e., an upward planar drawing in which every edge is represented by a segment. The *underlying graph* of a directed graph G is the undirected graph obtained by removing the directions on the edges of G. In the following we refer to directed paths, directed cycles, directed trees, directed outerplanar graphs, meaning upward planar directed graphs whose underlying graphs are paths, cycles, trees, and outerplanar graphs, respectively. A *Hamiltonian* directed graph G is a directed graph containing a path $(v_1, v_2, \cdots, v_n)$ passing through all vertices of G such that edge (v_i, v_{i+1}) is directed from v_i to v_{i+1}, for each $1 \leq i < n$. A *source* (resp. *sink*) in a directed graph G is a vertex having only outgoing edges (resp. having only incoming edges).

A set of points in the plane is in *general position* if no three points lie on the same line. The *convex hull* of a set of points P is the set of points that can be obtained as a convex combination of the points of P. A set of points is in *convex position* if no point is in the convex hull of the others.

An *upward straight-line embedding* of an n-vertex directed graph G into a set P of n points is a mapping of each vertex of G to a point of P providing a straight-line upward planar drawing. An *upward simultaneous embedding without mapping* of two n-vertex upward planar directed graphs G_1 and G_2 is a pair of upward planar straight-line drawings in which the vertices of G_1 and G_2 are placed on the same set of n points.

In order to deal with upward embeddings of directed graphs into point sets, we assume that no two points of any point set have the same y-coordinate[1]. Then, the points of any point set can be totally ordered by increasing y-coordinate. Hence, we refer to the i-th point as to the point such that exactly $i-1$ points have smaller y-coordinate. The first and the last point of a point set P are denoted by $p_m(P)$ and $p_M(P)$, respectively. In a convex point set P, two points are *adjacent* if the segment between them is on the border of the convex hull of P. Points $\{v_1, v_2, \cdots, v_k\}$ in a convex point set P are *consecutive* if v_i and v_{i+1} are adjacent, for each $1 \leq i < k$. We call a *one-side* convex point set any convex point set P in which $p_M(P)$ and $p_m(P)$ are adjacent.

3 Graphs with Upward Straight-Line Embeddings into Every Point Set

In this section we consider the problem of determining which directed graphs admit an upward straight-line embedding into every point set in general or in convex position.

3.1 Biconnected Directed Graphs and Hamiltonian Directed Graphs

First, we show that no biconnected directed graph with more than three vertices has an upward straight-line embedding into every point set in convex position, and hence into every point set in general position. The following two lemmata are well-known:

Lemma 1. *Let C be any directed cycle. The number of sources in C is equal to the number of sinks.*

Lemma 2. *Let O be a straight-line embedding of a directed graph into a point set in convex position. Then O is an outerplanar embedding.*

We now observe the following lemmata.

Lemma 3. *Let G be a directed graph containing a cycle C. Suppose that C has at least two vertices u and v that are sources in C. Then there exists a convex point set P such that G has no upward straight-line embedding into P.*

[1] Such an assumption is not a great loss of generality as every point set can be turned into one without two points having the same y-coordinate, by rotating the Cartesian axes by an arbitrarily-small angle. Further, assuming that no two points have the same y-coordinate avoids trivial counter-examples and the *a priori* impossibility of drawing an edge between two specified points of the point set.

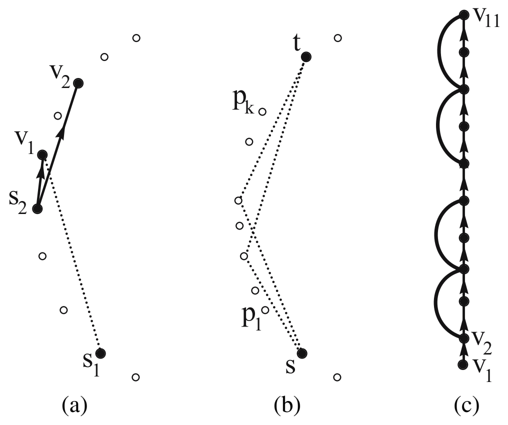


Fig. 1. (a)–(b) Illustrations for the proofs of Lemmata 3 and 4, respectively. Dotted segments represent paths connecting two vertices. (c) A graph belonging to the family $\mathcal{G}_{11}$ of Hamiltonian directed graphs with 11 vertices that admit upward straight-line embeddings into every point set.

Proof. Consider any one-side convex point set P. Let s_1 and s_2 be two sources in C. Suppose, w.l.o.g., that the point on which s_1 is drawn has y-coordinate smaller than the one on which s_2 is drawn. Since s_2 is a source in C, there exist edges (s_2, v_1) and (s_2, v_2) going out of s_2 and belonging to C. Suppose, w.l.o.g., that the point on which v_1 is drawn has y-coordinate smaller than the one on which v_2 is drawn. Since C is a cycle, there exist two disjoint paths connecting v_1 and s_1. One of these paths does not contain s_2 and v_2 and hence it crosses edge (s_2, v_2); see Fig. 1.a. □

Lemma 4. *Let G be a directed graph containing a cycle C. Suppose that C has exactly one source s and one sink t. Suppose also that each of the two directed paths $\mathcal{P}_1$ and $\mathcal{P}_2$ of C connecting s and t has at least one node different from s and t. Then there exists a convex point set P such that G has no upward straight-line embedding into P.*

Proof. Consider any one-side convex point set P. Consider any drawing Γ of G into P and let $p(s)$ and $p(t)$ be the points of P where s and t are drawn in Γ, respectively. Consider the subset P_k of P whose points have y-coordinates greater or equal than the one of $p(s)$ and less or equal than the one of $p(t)$. Since Γ is straight-line and upward, both $\mathcal{P}_1$ and $\mathcal{P}_2$ lie inside the convex hull H_k of P_k. As $\mathcal{P}_1$ and $\mathcal{P}_2$ touch the border of H_k in at least one point different from $p(s)$ and $p(t)$, $\mathcal{P}_1$ and $\mathcal{P}_2$ cross; see Fig. 1.b. □

Theorem 1. *There exists no biconnected directed graph with more than three vertices that admits an upward straight-line embedding into every point set in convex position.*

Proof. Consider any biconnected directed graph G with more than three vertices. Consider any one-side convex point set P_1. By Lemma 2, any embedding Γ of G into P_1 is outerplanar, hence all vertices of G are incident to the outer face of G. Since G is biconnected, the outer face of G is a simple cycle C. Hence, C is a cycle passing through all vertices of G. By Lemma 3, C contains exactly one vertex s that is source in C and hence, by Lemma 1, exactly one vertex t that is sink in C. Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be the directed paths of C connecting s and t. By Lemma 4, one out of $\mathcal{P}_1$ and $\mathcal{P}_2$, say $\mathcal{P}_1$, is a Hamiltonian directed path and the other one, say $\mathcal{P}_2$, is edge (s,t). Now consider any convex point set P_2 in which the line l through $p_m(P_2)$ and $p_M(P_2)$ determines two half-planes both containing points of P_2. Such a point set exists if $n \geq 4$. Since $\mathcal{P}_1$ is a Hamiltonian directed path between s and t, there is a vertex of $\mathcal{P}_1$ in each point of P_2. Then, there is at least one edge of $\mathcal{P}_1$ crossing l and hence crossing (s,t). □

Next, we characterize those Hamiltonian directed graphs that admit an upward straight-line embedding into every point set in general position and into every point set in convex position. Let $\mathcal{P}_n = (v_1, v_2, \cdots, v_n)$ be an n-vertex directed path, where edge (v_i, v_{i+1}) is directed from v_i to v_{i+1}, for $1 \leq i \leq n-1$. Let $\mathcal{G}_n$ be the family of n-vertex Hamiltonian directed graphs defined as follows: Each graph $G \in \mathcal{G}_n$ can be obtained by adding to $\mathcal{P}_n$ a set of edges E, where each edge of E is directed from a vertex v_i to a vertex v_{i+2}, for some $1 \leq i \leq n-2$, and no two edges (v_i, v_{i+2}) and (v_{i+1}, v_{i+3}) belong to E, for any $1 \leq i \leq n-3$; see Fig. 1.c.

Theorem 2. *An n-vertex Hamiltonian directed graph admits an upward straight-line embedding into every point set in general position if and only if it belongs to $\mathcal{G}_n$.*

Proof. First, we prove the necessity. Suppose that there exists an Hamiltonian directed graph G that admits an upward straight-line embedding into every point set and that does not belong to $\mathcal{G}_n$. If $\mathcal{P}_n = (v_1, v_2, \cdots, v_n)$ is the Hamiltonian directed path of G, then either G contains an edge (v_i, v_j), with $i+2 < j \leq n$, for some $1 \leq i \leq n-3$, or it contains two edges (v_i, v_{i+2}) and (v_{i+1}, v_{i+3}), for some $1 \leq i \leq n-3$.

Suppose G contains edge (v_i, v_j), with $i+2 < j \leq n$. Consider any convex point set P with the following property. Let l be the line through the i-th and the j-th point of P. Then the $(i+1)$-th point and the $(i+2)$-th point of P are on different sides of l. For the upward constraint, the k-th vertex of $\mathcal{P}_n$ is drawn on the k-th point of P, for $k = 1, \cdots, n$. Hence, edge (v_{i+1}, v_{i+2}) crosses edge (v_i, v_j). Suppose G contains two edges (v_i, v_{i+2}) and (v_{i+1}, v_{i+3}), for some $1 \leq i \leq n-3$. Consider any one-side convex point set P. For the upward constraint, the k-th vertex of $\mathcal{P}_n$ is drawn on the k-th point of P, for $k = 1, \cdots, n$. Then, edge (v_i, v_{i+2}) crosses edge (v_{i+1}, v_{i+3}).

We prove the sufficiency. Consider any n-vertex Hamiltonian directed graph G in $\mathcal{G}_n$ and any point set P in general position. Draw $\mathcal{P}_n = (v_1, v_2, \cdots, v_n)$ into P as a y-monotone path. Draw edges (v_i, v_{i+2}) belonging to G. Since the drawing is straight-line, edge (v_i, v_{i+2}) intersects or overlaps only those edges intersecting the open horizontal strip S delimited by the horizontal lines through v_i and v_{i+2}. By the definition of $\mathcal{G}_n$, the only edges that have intersections with S are (v_i, v_{i+1}), (v_i, v_{i+2}) and (v_{i+1}, v_{i+2}). Since P is in general position, then v_i, v_{i+1}, and v_{i+2} are not on the same line and edges (v_i, v_{i+1}), (v_i, v_{i+2}) and (v_{i+1}, v_{i+2}) do not intersect or overlap. □

3.2 Directed Paths and Directed Trees

We show how to construct upward straight-line embeddings of directed paths into point sets in convex position. We observe the following:

Lemma 5. *Let P be any one-side convex point set with n points and let $\mathcal{P} = (v_1, v_2, \cdots, v_n)$ be any n-vertex directed path. If edge (v_1, v_2) is directed from v_1 to v_2 (resp.*

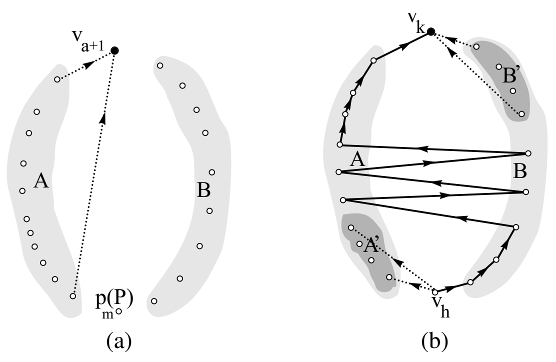


Fig. 2. (a) Illustration for the proof of Theorem 3, when (v_a, v_{a+1}) is directed from v_a to v_{a+1}, and (v_{a+1}, v_{a+2}) is directed from v_{a+2} to v_{a+1}. (b) Illustration for the proof of Theorem 3, when (v_a, v_{a+1}) is directed from v_a to v_{a+1}, and (v_{a+1}, v_{a+2}) is directed from v_{a+1} to v_{a+2}.

from v_2 to v_1), then there exists an upward straight-line embedding of $\mathcal{P}$ into P in which v_1 is on $p_m(P)$ (resp. v_1 is on $p_M(P)$).

Proof sketch. The argument uses induction on the number of points of P (and of vertices of $\mathcal{P}$). If $n = 1$ the statement trivially follows. Consider any one-side convex point set P with n points, and any n-vertex directed path $\mathcal{P} = (v_1, v_2, \cdots, v_n)$. Suppose that (v_1, v_2) is directed from v_1 to v_2, the other case being analogous. Consider the point set $P' = P \setminus \{p_m(P)\}$ which is a one-side convex point set. By induction $\mathcal{P}' = (v_2, \cdots, v_n)$ admits an upward straight-line embedding into P' in which v_2 is either on $p_M(P')$ or on $p_m(P')$ (depending on the direction of edge (v_2, v_3)). In both cases v_1 can be mapped to $p_m(P)$ and edge (v_1, v_2) can be drawn as a segment. The resulting drawing Γ is easily shown to be straight-line, upward, and planar. □

Theorem 3. *Every n-vertex directed path admits an upward straight-line embedding into every convex point set with n points.*

Proof sketch. Let $\mathcal{P} = (v_1, v_2, \cdots, v_n)$ be any directed path and let P be any convex point set with n points. Let A and B be the subsets of P to the left and to the right, respectively, of the line through $p_M(P)$ and $p_m(P)$. Let $|A| = a$ and $|B| = b$. Consider edges (v_a, v_{a+1}) and (v_{a+1}, v_{a+2}).

If edge (v_a, v_{a+1}) is directed from v_a to v_{a+1} and (v_{a+1}, v_{a+2}) is directed from v_{a+2} to v_{a+1} (see Fig. 2.a), apply Lemma 5 to construct an upward straight-line embedding of path $\mathcal{P}_1 = (v_a, v_{a-1}, \cdots, v_1)$ into A in which v_a is placed either on $p_M(A)$ or on $p_m(A)$, and apply Lemma 5 to construct an upward straight-line embedding of path $\mathcal{P}_2 = (v_{a+1}, v_{a+2}, \cdots, v_n)$ into $B \cup \{p_M(P), p_m(P)\}$ in which v_{a+1} is placed on $p_M(P)$. The resulting drawing Γ is easily shown to be straight-line, upward, and planar. An upward straight-line embedding of $\mathcal{P}$ into P can be constructed analogously if (v_a, v_{a+1}) is directed from v_{a+1} to v_a and (v_{a+1}, v_{a+2}) is directed from v_{a+1} to v_{a+2}.

If (v_a, v_{a+1}) is directed from v_a to v_{a+1} and (v_{a+1}, v_{a+2}) is directed from v_{a+1} to v_{a+2} (see Fig. 2.b), let h be the smallest index such that edge (v_i, v_{i+1}) is directed

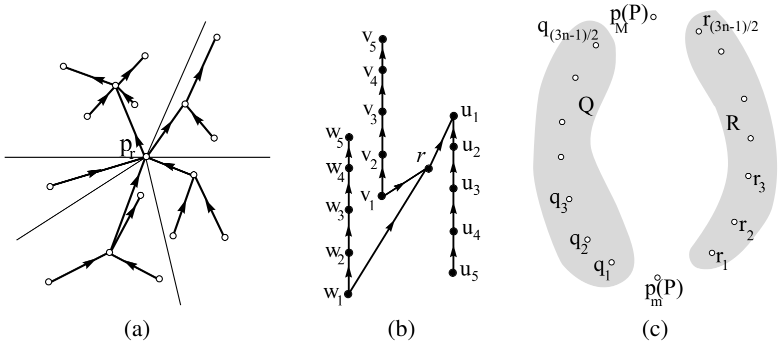


Fig. 3. (a) Illustration for Theorem 4. (b)–(c) A tree T and a point set P such that T does not admit any upward straight-line embedding into P.

from v_i to v_{i+1}, for $i = h, h+1, \ldots, a$, and let k be the greatest index such that edge (v_i, v_{i+1}) is directed from v_i to v_{i+1}, for $i = a, a+1, \ldots, k-1$. Consider path $\mathcal{P}_1 = (v_{h-1}, v_{h-2}, \cdots, v_1)$ and consider the point set $A' \subseteq A$ composed of the first $h-1$ points of A. Apply Lemma 5 to construct an upward straight-line embedding of $\mathcal{P}_1$ into A' such that v_{h-1} is placed either on $p_M(A')$, or on $p_m(A')$. Consider path $\mathcal{P}_2 = (v_{k+1}, v_{k+2}, \cdots, v_n)$ and consider the point set $B' \subseteq B$ composed of the last $n-k$ points of B. Apply Lemma 5 to construct an upward straight-line embedding of $\mathcal{P}_2$ into B' such that v_{k+1} is placed either on $p_M(B')$, or on $p_m(B')$. Consider path $\mathcal{P}_3 = (v_h, v_{h+1}, \cdots, v_k)$ and consider the point set $C' \equiv P \setminus \{A' \cup B'\}$. Construct an upward straight-line embedding of $\mathcal{P}_3$ into C' such that the i-th vertex of $\mathcal{P}_3$ is placed on the i-th point of C', for $i = h, \cdots, k$. The resulting drawing Γ is easily shown to be straight-line, upward, and planar. An upward straight-line embedding of $\mathcal{P}$ into P can be constructed analogously if (v_a, v_{a+1}) is directed from v_{a+1} to v_a, and (v_{a+1}, v_{a+2}) is directed from v_{a+2} to v_{a+1}. $\square$

Directed trees of *diameter* at most four, i.e., in which the maximum number of edges in any path is at most four, admit upward straight-line embeddings into every point set in convex or in general position, as in the following theorem, whose constructive proof is shown in Fig. 3.a and detailed in the full version of the paper [9].

Theorem 4. *Every directed tree with diameter at most four admits an upward straight-line embedding into every point set in general position.*

Next, we show that not all directed trees admit a straight-line upward embedding into every point set in convex position (and hence into every point set in general position). The proof uses as main tool the following lemma. Consider any tree T and any convex point set P. Let u be any node of T and let $T_1, T_2, \cdots, T_k$ be the subtrees of T obtained by removing u and its incident edges from T.

Lemma 6. *In any upward straight-line embedding of T into P, the vertices of T_i are mapped into a set of consecutive points of P, for each $i = 1, 2, \cdots, k$.*

Assume n is odd and greater or equal than 5. Consider a directed tree T composed of: (i) one vertex r of degree three, (ii) three paths of n vertices $\mathcal{P}_1 = (u_1, u_2, \cdots, u_n)$, where (u_i, u_{i+1}) is directed from u_{i+1} to u_i, $\mathcal{P}_2 = (v_1, v_2, \cdots, v_n)$, where (v_i, v_{i+1}) is directed from v_i to v_{i+1}, and $\mathcal{P}_3 = (w_1, w_2, \cdots, w_n)$, where (w_i, w_{i+1}) is directed from w_i to w_{i+1}, and (iii) edge (r, u_1) directed from r to u_1, edge (r, v_1) directed from v_1 to r, and edge (r, w_1) directed from w_1 to r; see Fig. 3.b. Let P be a convex point set with $3n + 1$ points such that a set Q of $(3n - 1)/2$ points $q_1, q_2, \cdots, q_{(3n-1)/2}$ are to the left and a set R of $(3n - 1)/2$ points $r_1, r_2, \cdots, r_{(3n-1)/2}$ are to the right of the line connecting $p_M(P)$ and $p_m(P)$. Assume that $y(p_m(P)) < y(q_1) < y(r_1) < y(q_2) < y(r_2) < \cdots < y(q_{(3n-1)/2}) < y(r_{(3n-1)/2}) < y(p_M(P))$; see Fig. 3.c. It is not difficult to show that T does not admit any upward straight-line embedding into P.

Theorem 5. *For every n odd greater or equal than 5, there exists a $(3n + 1)$-vertex directed tree T and a $(3n + 1)$-point convex point set P such that T does not admit a straight-line upward embedding into P.*

4 Upward Simultaneous Embeddings of Directed Graphs

First, we show two n-vertex upward planar directed graphs that do not admit any upward simultaneous embedding without mapping. Let G^1_n be the graph with $n \geq 5$ vertices $u^1_1, u^1_2, \cdots, u^1_n$, with a Hamiltonian directed path $\mathcal{P}_1 = (u^1_1, u^1_2, \cdots, u^1_n)$, with edges (u^1_1, u^1_3), (u^1_1, u^1_4), (u^1_1, u^1_5), (u^1_2, u^1_4), and (u^1_2, u^1_5), and with any other set of edges such that G^1_n is still upward planar. Let G^2_n be the graph with n vertices $u^2_1, u^2_2, \cdots, u^2_n$, with a Hamiltonian directed path $\mathcal{P}_2 = (u^2_1, u^2_2, \cdots, u^2_n)$, with edges (u^2_1, u^2_3), (u^2_1, u^2_4), (u^2_1, u^2_5), (u^2_2, u^2_4), and (u^2_3, u^2_5), and with any other set of edges such that G^2_n is still upward planar. It is not difficult to show that G^1_n and G^2_n do not admit any upward simultaneous embedding without mapping.

Theorem 6. *For every $n \geq 5$, there exist two n-vertex upward planar directed graphs that do not admit a simultaneous embedding without mapping.*

Next, we show that deciding whether two upward planar directed graphs G_1 and G_2 admit a simultaneous embedding without mapping is an $\mathcal{NP}$-hard problem (in the following called UPWARD SIMULTANEOUS EMBEDDING WITHOUT MAPPING). In order to prove the $\mathcal{NP}$-hardness of such a problem, we perform a reduction from 3-PARTITION [10]. An instance of 3-PARTITION consists of a set A of $3m$ elements, a bound $B \in Z^+$, and a size $s(a) \in Z^+$ for each element $a \in A$, such that $B/4 < s(a) < B/2$ and such that $\sum_{a \in A} s(a) = mB$. The 3-PARTITION problem is to decide whether A can be partitioned into m disjoint sets $A_1, A_2, \ldots, A_m$ such that, for $1 \leq i \leq 3$ and $1 \leq j \leq m$, $\sum_{a_i \in A_j} s(a_i) = B$.

We describe how to construct an instance of UPWARD SIMULTANEOUS EMBEDDING WITHOUT MAPPING from an instance of 3-PARTITION.

Graph G_1 (see Fig. 4.a) contains m directed paths $\mathcal{P}_i = (u^{i,1}_1, u^{i,2}_1, \ldots, u^{i,2B}_1)$ of $2B$ vertices, with $1 \leq i \leq m$. Edge $(u^{i,j}_1, u^{i,j+1}_1)$ is directed from $u^{i,j}_1$ to $u^{i,j+1}_1$, for $1 \leq j \leq 2B - 1$ and $1 \leq i \leq m$. Further, G_1 has an edge directed from vertex $u^{i,2B}_1$ to vertex $u^{i+1,2B}_1$, for each i odd such that $1 \leq i \leq m - 1$, and an edge directed from vertex $u^{i+1,1}_1$ to vertex $u^{i,1}_1$, for each i even such that $2 \leq i \leq m-1$. Finally, G_1 has two vertices w_1 and z_1 such that, for every vertex $u^{i,j}_1$, with $1 \leq i \leq m$ and $1 \leq j \leq 2B$,

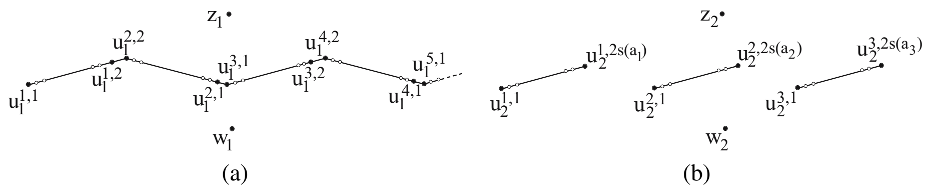


Fig. 4. (a)-(b) Graphs G_1 and G_2. In order to improve readability, edges incident to vertices w_1, z_1, w_2, and z_2 are not shown. Labels are shown only for the first and the last vertex of paths $\mathcal{P}_i$ of G_1 and for the first and the last vertex of the paths of each component G_2^i. An edge (a, b) is oriented from a to b if, in the figure, the y-coordinate of b is greater than the one of a.

there exists an edge from $u_1^{i,j}$ to z_1 and an edge from w_1 to $u_1^{i,j}$. It is easy to see that G_1 has only one upward planar embedding, up to a flip of the whole graph. Further, the subgraph $\mathcal{P}$ of G_1 induced by the all the vertices of G_1 except for w_1 and z_1 is a directed path. We say that two vertices $u_1^{i_1,j_1}$ and $u_1^{i_2,j_2}$ of G_1 are *consecutive* in G_1 if they are adjacent in $\mathcal{P}$.

Graph G_2 (see Fig. 4.b) has a triconnected component G_2^i for each element $a_i \in A$ (except for the elements a_i such that $s(a_i) = 1$ to which biconnected components G_2^i correspond). All the G_2^i's share two vertices w_2 and z_2. Graph G_2^i has $2 \cdot s(a_i) + 2$ vertices, where $2 \cdot s(a_i)$ vertices form a directed path $(u_2^{i,1}, u_2^{i,2}, \cdots, u_2^{i,2\cdot s(a_i)})$ such that edge $(u_2^{i,j}, u_2^{i,j+1})$ is directed from $u_2^{i,j}$ to $u_2^{i,j+1}$, for $1 \leq j \leq 2 \cdot s(a_i) - 1$. For every $1 \leq i \leq 3m$ and $1 \leq j \leq 2 \cdot s(a_i)$, there is an edge directed from $u_2^{i,j}$ to z_2 and an edge directed from w_2 to $u_2^{i,j}$. Notice that, an embedding of G_2 is completely specified by a left-to-right order of the G_2^i's, up to flip of some G_2^i's. We say that two vertices $u_2^{i_1,j_1}$ and $u_2^{i_2,j_2}$ of G_2 are *consecutive* in G_2 if $i_1 = i_2$ and $j_2 = j_1 \pm 1$. Notice that, in any upward simultaneous embedding of G_1 and G_2, vertex w_1 of G_1 must be mapped to vertex w_2 of G_2 and vertex z_1 of G_1 must be mapped to vertex z_2 of G_2. We observe the following:

Lemma 7. *Let $u_2^{i,j}$ and $u_2^{i,j+1}$ be two consecutive vertices of G_2, for some $1 \leq i \leq 3m$ and $1 \leq j \leq 2 \cdot s(a_i) - 1$. In any upward simultaneous embedding without mapping of G_1 and G_2, vertices $u_2^{i,j}$ and $u_2^{i,j+1}$ are mapped to consecutive vertices of G_1.*

Corollary 1. *Consider any upward simultaneous embedding without mapping of G_1 and G_2 in which two vertices $u_1^{i_1,j_1}$ and $u_1^{i_2,j_2}$ of G_1 have been mapped to two vertices of the same component G_2^k of G_2. All vertices between $u_1^{i_1,j_1}$ and $u_1^{i_2,j_2}$ in $\mathcal{P}$ have been mapped to vertices of G_2^k.*

Lemma 8. *In any upward simultaneous embedding without mapping of G_1 and G_2 there exists no component G_2^i of G_2 which has two vertices mapped to vertices $u_1^{j,2B}$ and $u_1^{j+1,2B-1}$, for every j odd, and there exists no component G_2^i of G_2 which has two vertices mapped to vertices $u_1^{j,1}$ and $u_1^{j+1,2}$, for every j even.*

We obtain the following:

Theorem 7. *UPWARD SIMULTANEOUS EMBEDDING WITHOUT MAPPING is $\mathcal{NP}$-hard.*

Proof. The reduction described at the beginning of the section can clearly be performed in polynomial time. In fact, 3-PARTITION is $\mathcal{NP}$-hard in the strong sense, hence it is $\mathcal{NP}$-hard even if $2mB$, which is the size of the constructed instance of UPWARD SIMULTANEOUS EMBEDDING WITHOUT MAPPING, is bounded by a polynomial in m. We show that an instance of 3-PARTITION admits a solution if and only if the corresponding instance of UPWARD SIMULTANEOUS EMBEDDING WITHOUT MAPPING admits a solution.

Consider any instance A of 3-PARTITION admitting a solution $A_1, A_2, \dots, A_m$ such that, for $1 \leq i \leq 3$ and $1 \leq j \leq m$, $\sum_{a_i \in A_j} s(a_i) = B$. We show how to map the vertices of G_2 to the vertices of G_1. For each $i = 1, \cdots, m$, consider components G_2^x, G_2^y, and G_2^z of G_2 corresponding to elements a_x, a_y, and a_z of A_i, respectively. Since G_2^k has $2 \cdot s(a_k)$ vertices different from w_2 and z_2, then G_2^x, G_2^y, and G_2^z have exactly $2B$ vertices different from w_2 and z_2. Map vertex $u_2^{x,j}$ of G_2^x to vertex $u_1^{i,j}$ of G_1, for each $1 \leq j \leq 2 \cdot s(a_x)$; map vertex $u_2^{y,j}$ of G_2^y to vertex $u_1^{i,2\cdot s(a_x)+j}$ of G_1, for each $1 \leq j \leq 2 \cdot s(a_y)$; map vertex $u_2^{z,j}$ of G_2^z to vertex $u_1^{i,2\cdot s(a_x)+2\cdot s(a_y)+j}$ of G_1, for each $1 \leq j \leq 2 \cdot s(a_z)$. It is easy to see that G_2, when its vertices are mapped to the vertices of G_1 as described above, is a subgraph of G_1. Hence, an upward simultaneous embedding of G_1 and G_2 is obtained by any upward straight-line drawing of G_1.

Now consider any upward simultaneous embedding (Γ_1, Γ_2) of G_1 and G_2. We show how to construct a solution $A_1, A_2, \dots, A_m$ for the corresponding instance A of 3-PARTITION. We claim that the vertices of three components $G_2^{x,i}$, $G_2^{y,i}$, and $G_2^{z,i}$ of G_2, except for w_2 and z_2, have been mapped to all and only the vertices of path $\mathcal{P}_i$ of G_1, for each $i = 1, 2, \cdots, m$. The claim directly implies the existence of a solution to the instance of 3-PARTITION, since the claim implies that $|G_2^{x,i}| + |G_2^{y,i}| + |G_2^{z,i}| = 2B$ and hence, by construction, the three elements of A corresponding to $G_2^{x,i}$, $G_2^{y,i}$, and $G_2^{z,i}$ sum up to B. Consider the component $G_2^{x,1}$ of G_2 which has a vertex mapped to $u_1^{1,1}$. By Corollary 1, the vertices of $G_2^{x,1}$ are mapped to consecutive vertices of G_1, hence they are mapped to the vertices of $\mathcal{P}$ from $u_1^{1,1}$ to $u_1^{1,|G_2^{x,1}|}$. Analogously, the component $G_2^{y,1}$ of G_2 which has a vertex mapped to $u_1^{1,|G_2^{x,1}|+1}$ has vertices mapped to vertices of $\mathcal{P}$ from $u_1^{1,|G_2^{x,1}|+1}$ to $u_1^{1,|G_2^{x,1}|+|G_2^{y,1}|}$. Observe that $|G_2^{x,1}| + |G_2^{y,1}| < 2B$, since each component of G_2 has less than B vertices (by the assumption that $s(a_i) < B/2$). Hence, there exists a component $G_2^{z,1}$ of G_2 which has a vertex mapped to $u_1^{1,|G_2^{x,1}|+|G_2^{y,1}|+1}$. By Corollary 1, the vertices of $G_2^{z,1}$ are mapped to consecutive vertices of G_1. Suppose that $|G_2^{x,1}| + |G_2^{y,1}| + |G_2^{z,1}| > 2B$. Since each of $|G_2^{x,1}|$, $|G_2^{y,1}|$, and $|G_2^{z,1}|$ is even, $|G_2^{x,1}| + |G_2^{y,1}| + |G_2^{z,1}|$ is even, as well, hence $G_2^{z,1}$ has a vertex mapped to $u_1^{2,2B-1}$. By Lemma 8, (Γ_1, Γ_2) is not an upward simultaneous embedding without mapping of G_1 and G_2. Now suppose that $|G_2^{x,1}| + |G_2^{y,1}| + |G_2^{z,1}| < 2B$. Then, there exists a component $G_2^{w,1}$ that is mapped to $u_1^{1,|G_2^{x,1}|+|G_2^{y,1}|+|G_2^{z,1}|+1}$. By Corollary 1, the vertices of $G_2^{w,1}$ are mapped to consecutive vertices of G_1. Further, by construction and by the assumption that $s(a_i) > B/4$, $|G_2^{x,1}| + |G_2^{y,1}| + |G_2^{z,1}| + |G_2^{w,1}| > 2B$. Hence, since $|G_2^{x,1}| + |G_2^{y,1}| + |G_2^{z,1}| + |G_2^{w,1}|$ is even, $G_2^{w,1}$ has vertices mapped both to $u_1^{1,2B}$ and to $u_1^{2,2B-1}$. By Lemma 8, (Γ_1, Γ_2) is not an upward simultaneous embedding without mapping of G_1 and G_2. It follows that $|G_2^{x,1}| + |G_2^{y,1}| + |G_2^{z,1}| = 2B$. The previous argument can be iterated to show that the vertices of three components $G_2^{x,i}$, $G_2^{y,i}$, and $G_2^{z,i}$ of G_2, except for w_2 and z_2, have been mapped to all and only the vertices of path $\mathcal{P}_i$ of G_1, for each $i = 1, 2, \cdots, m$, proving the claim and hence the theorem. □

5 Conclusions and Open Problems

In this paper we have shown combinatorial and complexity results regarding the problem of constructing upward straight-line embeddings of directed graphs into point sets.

We have shown families of directed graphs that admit a straight-line upward embedding into every point set in convex or in general position, and families that do not. However, the problem of characterizing those graphs admitting a straight-line upward embedding into every point set in general or in convex position is still open. With this in mind we note that if an upward planar directed graph admits an upward straight-line embedding into every point set in general or in convex position, not all its subgraphs, in general, admit an upward straight-line embedding into every point set in general or in convex position; see Fig. 5. However, the necessary conditions of Lemmata 2, 3, and 4 strongly restrict the class of upward planar directed graphs to investigate. The following two problems naturally arise from the results of this paper: (1) Does every directed path admit a straight-line upward embedding into every point set in general position? (2) Does every directed caterpillar admit a straight-line upward embedding into every point set in convex/general position?

Deciding whether a directed graph admits an upward straight-line embedding into a given point set in general position is likely to be $\mathcal{NP}$-hard, since the same problem is $\mathcal{NP}$-hard in its undirected version [4]. However, we do not know the time complexity of testing whether a directed graph admits an upward straight-line embedding into a given point set in convex position. The same problem can be solved in linear time for undirected graphs as a graph admits a straight-line embedding into a given point set in convex position if and only if it is outerplanar, which can be tested in linear time [16].

We proved that deciding whether two upward planar directed graphs admit an upward simultaneous embedding without mapping is $\mathcal{NP}$-hard. We observe that the same problem is polynomial-time solvable for trees [11]. Hence, it would be interesting to solve the problem for subclasses of planar digraphs richer than directed trees, e.g., *outerplanar digraphs* and *series-parallel digraphs*.

A final open problem is to find the minimum cardinality $f(n)$ of a set of points P in the plane such that every n-vertex planar digraph admits an upward straight-line

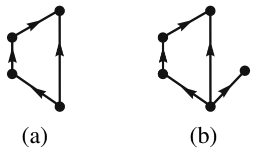


Fig. 5. (a) An upward planar directed graph G_1 that, by Theorem 1, does not admit an upward straight-line embedding into every 4-point point set in convex position. (b) An upward planar directed graph G_2 that admits an upward straight-line embedding into every 5-point point set in convex position and that contains G_1 as a subgraph.

embedding in which the vertices are drawn at points of P. While this is the directed version of one of the most studied Graph Drawing problems [5,6,14,15], the only known result is that the minimum size of any grid into which every planar digraph can be drawn is exponential [8], hence a polynomial upper bound for $f(n)$ would be interesting.

References

1. Bose, P.: On embedding an outer-planar graph in a point set. Computat. Geom. Th. Appl. 23(3), 303–312 (2002)
2. Bose, P., McAllister, M., Snoeyink, J.: Optimal algorithms to embed trees in a point set. J. Graph Algorithms Appl. 1(2), 1–15 (1997)
3. Braß, P., Cenek, E., Duncan, C.A., Efrat, A., Erten, C., Ismailescu, D., Kobourov, S.G., Lubiw, A., Mitchell, J.S.B.: On simultaneous planar graph embeddings. Computat. Geom. Th. Appl. 36(2), 117–130 (2007)
4. Cabello, S.: Planar embeddability of the vertices of a graph using a fixed point set is NP-hard. J. Graph Algorithms Appl. 10(2), 353–366 (2006)
5. Chrobak, M., Karloff, H.: A lower bound on the size of universal sets for planar graphs. Sigact News 20(4), 83–86 (1989)
6. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10(1), 41–51 (1990)
7. Di Battista, G., Tamassia, R.: Algorithms for plane representations of acyclic digraphs. Theor. Comput. Sci. 61, 175–198 (1988)
8. Di Battista, G., Tamassia, R., Tollis, I.G.: Area requirement and symmetry display of planar upward drawings. Disc. & Computat. Geometry 7, 381–401 (1992)
9. Estrella-Balderrama, A., Frati, F., Kobourov, S.: Upward straight-line embeddings of directed graphs into point sets. Tech. Report RT-DIA-133-2007, University of Roma Tre (2008), `http://dipartimento.dia.uniroma3.it/ricerca/rapporti/rt/2008-133.pdf`
10. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York (1979)
11. Giordano, F., Liotta, G., Mchedlidze, T., Symvonis, A.: Computing upward topological book embeddings of upward planar digraphs. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 172–183. Springer, Heidelberg (2007)
12. Gritzmann, P., Mohar, B., Pach, J., Pollack, R.: Embedding a planar triangulation with vertices at specified positions. The American Mathematical Monthly 98, 165–166 (1991)
13. Kaufmann, M., Wiese, R.: Embedding vertices at points: Few bends suffice for planar graphs. J. Graph Algorithms Appl. 6(1), 115–129 (2002)
14. Kurowski, M.: A 1.235 lower bound on the number of points needed to draw all n-vertex planar graphs. Inf. Process. Lett. 92(2), 95–98 (2004)
15. Schnyder, W.: Embedding planar graphs on the grid. In: SODA 1990, pp. 138–148 (1990)
16. Wiegers, M.: Recognizing outerplanar graphs in linear time. In: Tinhofer, G., Schmidt, G. (eds.) WG 1986. LNCS, vol. 246, pp. 165–176. Springer, Heidelberg (1987)

Complexity of the Packing Coloring Problem for Trees

Jiří Fiala[1,⋆] and Petr A. Golovach[2,⋆⋆]

[1] Department of Applied Mathematics and Institute for Theoretical Computer Science (ITI), Charles University, Prague, Czech Republic
`fiala@kam.mff.cuni.cz`

[2] Department of Informatics, University of Bergen, PB 7803, 5020 Bergen, Norway
`petr.golovach@ii.uib.no`

Abstract. Packing coloring is a partitioning of the vertex set of a graph with the property that vertices in the i-th class have pairwise distance greater than i. We solve an open problem of Goddard et al. and show that the decision whether a tree allows a packing coloring with at most k classes is NP-complete.

We accompany this NP-hardness result by a polynomial time algorithm for trees for closely related variant of the packing coloring problem where the lower bounds on the distances between vertices inside color classes are determined by an infinite nondecreasing sequence of bounded integers.

Keywords: Packing coloring, computational complexity, graph algorithm, chordal graph.

1 Introduction

The concept of packing coloring comes from the area of frequency planning in wireless networks. This model emphasizes the fact that some frequencies might be used more sparely than the others.

In graph terms, we ask for a partitioning of the vertex set of a graph G into disjoint classes $X_1, \dots, X_k$ (representing frequency usage) according to the following constraints. Each color class X_i should be an *i-packing* i.e. a set of vertices with the property that any distinct pair $u, v \in X_i$ satisfies $\text{dist}(u, v) > i$. Here $\text{dist}(u, v)$ is the *distance* between u and v, i.e. the length of a shortest path from u to v and it is declared to be infinite when u and v belong to distinct components of connectivity.

Such partitioning into k classes is called a *packing k-coloring*, even though it is allowed that some sets X_i can be empty. The smallest integer k for which exists a packing k-coloring of G is called the *packing chromatic number* of G, and it is denoted by $\chi_p(G)$. The notion of the packing chromatic number was

⋆ ITI is supported by the Ministry of Education of the Czech Republic as project 1M0021620808.

⋆⋆ Supported by Norwegian Research Council.

established by Goddard et al. [7] under the name *broadcast chromatic number*. The term packing chromatic number was introduced by Brešar et al. [3].

Determining the packing chromatic number is difficult, even for special graph classes. For example, Sloper [11] showed that for trees of maximum degree three the the upper bound is seven, while χ_p is unbounded already on trees of maximum degree four. Goddard et al. [7] provided polynomial time algorithms for cographs and split graphs.

The packing chromatic number of the hexagonal grid is also seven as was shown by Brešar et al. [3] (the lower bound) and by Fiala and Lidický [personal communication] (the upper bound). Goddard et al. [7] also showed that the χ_p of the infinite two-dimensional square grid lies between 9 and 22. On the other hand, Finbow and Rall [10] proved that the packing chromatic numbers of the triangular infinite lattice as well as of the infinite three-dimensional square grid are unbounded.

The following decision problem arises naturally:

PACKING COLORING
Instance: A graph G and a positive integer k.
Question: Does G allow a packing k-coloring?

Goddard et al. [7] showed that the PACKING COLORING problem is NP-complete for general graphs and $k = 4$. They also asked about the computational complexity of this problem for trees. It was suggested by Brešar et al. [3] that the problem for trees can be difficult. Our main result is an affirmative proof of this conjecture:

Theorem 1. *The* PACKING COLORING *problem is* NP*-complete for trees.*

In contrary, the existence of a packing k-coloring can be expressed by a formula in Monadic Second Order Logic (MSOL), when k becomes fixed. It follows from work of Courcelle [4] that the PACKING COLORING problem is solvable in polynomial time for bounded treewidth graphs when k is fixed. In addition, we get the following corollaries for closely related graph classes:

Corollary 1. *The* PACKING COLORING *problem is fixed parameter tractable for chordal graphs with respect to the parameter k.*

Proof. If the given chordal graph G has a clique of size greater than k, then no packing k-coloring exists, since vertices of the clique have to be colored by distinct colors.

Otherwise, G has bounded clique size. Consequently it has also bounded treewidth and the result follows. □

Consequently, even in the case when k is not fixed, the PACKING COLORING problem becomes easy for special tree-like graphs:

Corollary 2. *The* PACKING COLORING *problem is solvable in polynomial time for graphs of bounded treewidth and of bounded diameter.*

Proof. Let us consider the following maximization problem: for a given graph G and a positive integer d we ask for some induced subgraph G' of G of maximum size that allows a packing d-coloring. By the results of Arnborg et al. [1] this problem can be solved by a linear algorithm on graphs of restricted treewidth for any fixed d, if the tree decomposition is given. (Also follows from a result of Courcelle et al. [5] on an adaptation of MSOL for optimization problems.)

Suppose that d is the upper bound on diameters of the considered graph class. If $k \leq d$, then the Packing Coloring problem can be solved in polynomial time. Otherwise any color $c > d$ can be used on at most one vertex of G. Therefore, $\chi_p(G) = d + |V_G \setminus V_{G'}|$, where G' is an optimal solution of the auxiliary maximization problem. □

Finally, we focus our attention to the more general concept of *S-packing coloring* introduced by Goddard et al. [7] as a generalization of packing coloring which provides more realistic model for the frequency assignment. Let $S = (s_1, s_2, \dots)$ be an infinite nondecreasing sequence of positive integers. In this new setting, vertices in the i-th class X_i are required to have distance greater than s_i. For example, the concept of the ordinary packing coloring is the S-packing coloring for $S = (1, 2, \dots)$. We address the following decision problem:

S-Packing Coloring
Parameter: A nondecreasing sequence S.
Instance: A graph G and a positive integer k.
Question: Does G allow an S-packing coloring with at most k color classes?

We show that the minimum number of color classes can be determined in polynomial time when the sequence S is bounded from above. To our knowledge the machinery of MSOL developed by Courcelle et al. [5] cannot be used directly for problems of such kind. On the other hand, this problem belongs to the class of *regular combination problems* considered by Borie [2], who proved that these problems can also be solved in polynomial time for graphs of bounded treewidth. Hence, we have the following corollary.

Corollary 3. *If S is a nondecreasing sequence with values bounded by a constant, then the S-*Packing Coloring *problem can be solved in polynomial time on graphs of bounded treewidth.*

For completeness, we provide a short description of an explicit algorithm for the S-Packing Coloring problem to prove Theorem 2.

Theorem 2. *For nondecreasing sequences S with values bounded by a constant t the S-*Packing Coloring *problem can be solved for trees on n vertices by an algorithm with running time $O(n^{2t+3})$.*

Our algorithm involves dynamic programming to evaluate all partial colorings for the initial classes with $s_i < t$ while minimizing the maximal number of uncolored vertices that are pairwise at distance at most t (i.e. the number of remaining color classes).

2 Proof of Theorem 1

For integers $a \le b$ we define discrete intervals as $[a,b] := \{a, a+1, \dots, b\}$.

2.1 Auxiliary constructions

We first construct a gadget where some vertices are forced predetermined colors in an arbitrary packing k-coloring.

Construction 1. *Let $t \le k$ be a positive integer. Construct a tree S_t with three levels as follows: The only vertex v_0 of the first level, called the* central *vertex, is of degree $t-1$, and all its neighbors $v_1, v_2, \dots, v_{t-1}$ are of degree k. The vertices $v_0, v_1, \dots, v_{t-1}$ are called the* inner *vertices of S_t.*

Lemma 1. *For every packing k-coloring of S_t the inner vertices are colored by distinct colors. Also for every subset I of $[1,k]$ of size at least t, a packing k-coloring of S_t exists such that the inner vertices are colored by distinct colors from I.*

Proof. If a packing k-coloring of S_t exists, then none of vertices v_i, $i \in [1, t-1]$ is colored by color 1, since it would be impossible to find k distinct colors in $[2,k]$ to color the neighbors of v_i. Hence, the colors of all inner non central vertices are greater or equal to 2, and each may present at most once as the maximal distance on the inner vertices is two. The central vertex (which can be colored by 1) is adjacent to other inner vertices and therefore must be colored by a different color.

For the second claim we construct the packing k-coloring from I as follows: Use elements of I bijectively on the inner vertices with the rule that the central vertex is colored by 1, if it is present in I. All leaves in the third level are colored by the color 1. □

Given some tree S_t, choose one of its leaves arbitrarily and call it the *root of S_t*. To simplify some expressions we involve an auxiliary parameter $d := 28$.

Construction 2. *For an odd $k > d$ and any $i \in [d+1, k]$ we construct the tree T_i as follows:*

1. *Take a copy of the tree S_i with the root u_3.*
2. *If $i < k$, then add a copy of S_k and join u_3 with the root of S_k by a path $u_3, u_4, \dots, u_{k-3}$ of length $k-6$.*
3. *If $i < k-2$, then for each odd j such that $i < j < k$ we add two copies of the tree S_j. The root of one of the two copies of S_j, called the top copy, is joined by a path of length $\lceil \frac{j}{2} \rceil - 3$ to the vertex $u_{\lceil j/2 \rceil}$. The root of the other one, called the bottom copy, is joined to the same $u_{\lceil j/2 \rceil}$ by a path of length $\lceil \frac{j}{2} \rceil - 4$.*
4. *Finally, if $i < k-1$ and i is odd, we add an extra copy of S_{i+1} and join its root to $u_{\lceil \frac{i}{2} \rceil}$ by a path of length $\lceil \frac{i}{2} \rceil - 3$.*

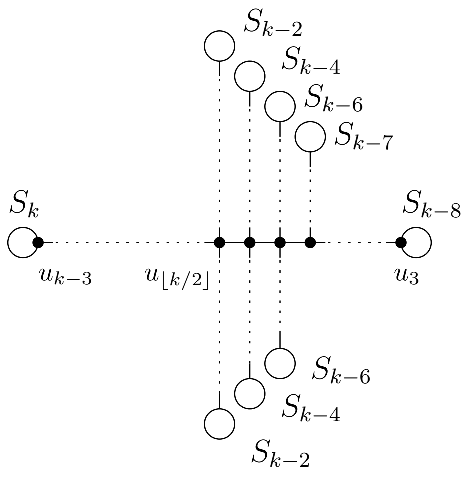


Fig. 1. The tree T_{k-8}

Denote by U the set of inner vertices of the copy of S_i in T_i. We choose the root of T_i *as some leaf in the copy of S_i that is at distance four from u_3.*

The construction of the tree T_{k-8} is depicted in Figure 1.

Lemma 2. *If i and k satisfy the assumptions of Construction 2, then*

1. *the vertices of U are colored by different colors from the set $[1, i]$ in any packing k-coloring of T_i;*
2. *the tree T_i admits a packing k-coloring such that the vertices colored by a color $c \in [i+1, k]$ are at distance more than c from the root of T_i. Moreover, the root of T_i is colored by the color 1, and vertices colored by the colors $i, i-1, i-2$ are at distance at least three from the root.*

Proof. Assume first that a packing k-coloring of T_i is given. By Lemma 1 the inner vertices of S_k are colored only by colors $[1, k]$, which proves the lemma in the case $i = k$.

We now determine the maximal distances between the inner vertices of used copies of S_j. They are summarized in Table 1.

Any color $j' > j$ cannot be used on the bottom copy of S_j, since they are used either on the top copies of $S_{j'-1}$ — for even j' — or on the bottom copies of $S_{j'}$ — for odd j'. The two copies of S_{k-2} have to be considered separately, but in that case we consider the copy of S_k at distance $k-1$. Hence, by Lemma 1 the interval $[1, j]$ is used on any bottom copy of S_j.

For the top copies of S_j an analogous argument can be used, considering the close distance to copies of $S_{j'}$ with $j' > j$. In addition, the color j is forbidden there, since it is used on the bottom copy of S_j and the distance is at most j. Again, by Lemma 1 the set $[1, j+1] \setminus \{j\}$ is used on the inner vertices of S_j.

The cases of S_{i+1} and S_i are treated in the same way.

Table 1. Maximal distances between the inner vertices of used copies of S_j

From	**To**	**Max. distance**
top S_j	S_k	$3 + \lceil \frac{j}{2} \rceil - 3 + (k - 3 - \lceil \frac{j}{2} \rceil) + 3 = k$
	top $S'_j, j' > j$	$\lceil \frac{j}{2} \rceil + \frac{j'-j}{2} + \lceil \frac{j'}{2} \rceil = j' + 1$
	bottom $S'_j, j' > j$	$\lceil \frac{j}{2} \rceil + \frac{j'-j}{2} + \lceil \frac{j'}{2} \rceil - 1 = j'$
bottom S_j	S_k	$\lceil \frac{j}{2} \rceil - 1 + (k - \lceil \frac{j}{2} \rceil) = k - 1$
	top $S'_j, j' > j$	$\lceil \frac{j}{2} \rceil + \frac{j'-j}{2} + \lceil \frac{j'}{2} \rceil = j' + 1$
	bottom $S'_j, j' > j$	$\lceil \frac{j}{2} \rceil + \frac{j'-j}{2} + \lceil \frac{j'}{2} \rceil - 1 = j'$
	top S_j	$\lceil \frac{j}{2} \rceil + \lceil \frac{j}{2} \rceil - 1 = j$
S_{i+1}	S_k	$\lceil \frac{i}{2} \rceil + (k - \lceil \frac{i}{2} \rceil) = k$
	top S_j	$\lceil \frac{i}{2} \rceil + \frac{j-i}{2} + \lceil \frac{j}{2} \rceil = j + 1$
	bottom S_j	$\lceil \frac{i}{2} \rceil + \frac{j-i}{2} + \lceil \frac{j}{2} \rceil - 1 = j$
S_i	S_k	$3 + k - 6 + 3 = k$
	top S_j	$\lceil \frac{j}{2} \rceil + \lceil \frac{j}{2} \rceil = j + 1$
	bottom S_j	$\lceil \frac{j}{2} \rceil + \lceil \frac{j}{2} \rceil - 1 = j$
	S_{i+1}	$\lceil \frac{i}{2} \rceil + \lceil \frac{i}{2} \rceil = i + 1$

Now we describe a packing k-coloring of T_i which satisfies the second claim. On each copy of S_j we use the coloring with colors from $[1, j]$, such that the center and all leaves are colored by the color 1, and the neighbor of the root of S_i is colored by 2. In addition, the neighbor of the root of T_i is colored by 3. We continue with the part of the tree around vertices $u_{\lfloor k/2 \rfloor}, \ldots, u_{\lceil i/2 \rceil}$. The periodic coloring pattern is depicted in Fig. 2 and uses only colors from the interval $[1, 9]$. What remains yet uncolored are paths, each of length at least eight. Along these paths we use pattern $1, 2, 1, 3, 1, 2, \ldots$ with possible appearance of the color 4 so the path coloring fits well with the coloring determined so far. By careful observation of distances between inner sets of trees S_j one can verify that we get a valid packing k-coloring. Moreover, vertices colored by the color $j > i$ are at distance more than j from the root of T_i as it was required. □

Construction 3. *Given $L \subset [d+1, k]$ of at most three elements we construct a tree T_L as follows. Take a copy of the tree S_{d+1}, and choose an inner non central vertex u arbitrarily. For every $j \in [d+1, k] \setminus L$ take an extra copy of the tree T_j and connect its root with a unique leaf neighbor of u by a path of length $j - 6$ (i.e., use different neighbors of u for different trees T_j).*

The root of T_L *is any leaf of S_{d+1} that is at distance three from u.*

Lemma 3. *If L and k satisfy assumptions of Construction 3, then*

1. *for every packing k-coloring of T_L the inner vertices of S_{d+1} are colored by different colors from the set $[1, d] \cup L$;*
2. *for every packing k-coloring of T_L at least one inner vertex of S_{d+1} is colored by the color from the set L;*
3. *for every set $I \subset [1, d] \cup L$, $|I| = d+1$, the tree T_L admits a packing k-coloring such that*
   - *the inner vertices of S_{d+1} are colored by the colors from I,*

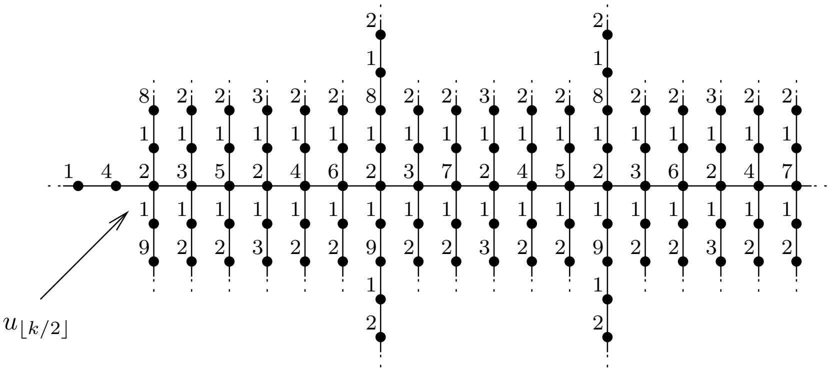


Fig. 2. The periodic coloring pattern used for the central part of T_i

- *vertices colored by the color j for $j \in [d+1,k] \setminus I$ are at distance more than j from the root of T_L, and*
- *vertices colored by the colors from L are 3-distant from the root.*

Proof. The first two claims follow immediately from the Lemmas 2 and 1.

For the proof of the third claim we construct the required packing k-coloring of T_L as follows: All vertices adjacent to the inner vertices of S_{d+1} are colored by the color 1. If $1 \in I$, then the central vertex of S_{d+1} is also colored by 1 as well. Consequently, $t = |L|$ inner vertices of S_{d+1} which are different from the central vertex and from u, and that are not adjacent to the root, are chosen and colored by the colors from L. The remaining inner vertices of S_{d+1} are colored by the remaining colors from I. The vertices of trees T_j are colored according to the second claim of Lemma 2. Finally, every path between the root of some T_j and u is colored by colors 1, 2, 3, with possible one appearance of the color 4. □

2.2 Polynomial Reduction

We proceed with reduction of the well known NP-complete 3-Satisfiability problem [6, problem L02, page 259] to our Packing Coloring problem for trees.

Let Φ be a boolean formula in conjunctive normal form with 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 $k := 4n+2d-1$ and $r := 2(d+n-1)$. For every variable x_i we define the set $X_i := \{2i+r, 2i+r+1\}$.

For every clause c_j a three element set $C_j \subset [1,k]$ is constructed as follows: If the clause c_j contains the literal x_i, then the integer $2i+r$ is included to the set C_j. On the other hand, if $\overline{x}_i \in c_j$, then $2i+r+1 \in C_j$.

Construction 4. *We construct the final tree T_Φ from the disjoint union of trees T_{X_i} over all variables x_i together with trees T_{C_j} over all clauses c_j. In addition we insert an extra new vertex u and join it to the roots of trees $T_{X_1}, T_{X_2}, \ldots, T_{X_n}$ by paths of length $d-3$. We also join u with the roots of $T_{C_1}, T_{C_2}, \ldots, T_{C_m}$ by paths of length $\lceil \frac{k}{2} \rceil - 3$.*

Lemma 4. *The tree T_Φ has a packing k-coloring if and only if the formula Φ can be satisfied.*

Proof. Suppose that a packing k-coloring of T_Φ exists. According to the second claim of Lemma 3 at least one element of the set X_i is used for among colors of the inner vertices of S_{d+1} in any T_{X_i} (in the sequel we denote this set of inner vertices by U_i). If the color $2i+t$ is used, then we set $x_i := \mathsf{false}$, and $x_i := \mathsf{true}$ otherwise.

For every $j \in [1,m]$ at least one color $c \in C_j$ is used on an inner vertex of S_{d+1} in T_{C_i} (this set we denote by W_j). Suppose that $c = 2i+t$ for some $i \in [1,n]$. In such a case the clause C_j contains the literal x_i. Since vertices of W_j and U_i are at distance at most $d + \lceil \frac{k}{2} \rceil = 2n + 2d \leq 2i + 2(d+n-1) = 2i + r$, the color $2i+r$ is not on the set U_i, and the variable x_i has to be assigned true.

Analogously, if $c = 2i+r+1$ for some $i \in [1,n]$, then the clause c_j contains literal $\overline{x}_i$. By the same arguments as before, the color $2i+r+1$ is not used on U_i, and $x_i = \mathsf{false}$.

Assume that a satisfying assignment of variables $x_1, x_2, \ldots, x_n$ for the formula Φ exists. For every $i \in [1,n]$ on any tree T_{X_i} we use the coloring described in the third statement of Lemma 3 arranged such that the vertices of U_i are colored by the set $[1,d] \cup \{2i+r+1\}$ if $x_i = \mathsf{true}$, and by the set $[1,d] \cup \{2i+r\}$ in the case when $x_i = \mathsf{false}$.

Note that the distance between different sets U_i is least $2d-4$. Also, if some color $c \in [d+1,k]$ is used among the sets U_i, then it is used only for a single vertex in a single set. Suppose that given clause C_j is satisfied by positively evaluated literal $x_i = \mathsf{true}$. Now the vertices of W_j are colored by the colors of the set $[1,d] \cup \{2i+r\}$ as described in Lemma 3. If C_j is satisfied by a literal $\overline{x}_i = true$, then vertices of W_j are colored by $[1,d] \cup \{2i+r+1\}$.

The distance between different sets W_i is least $2\lceil \frac{k}{2} \rceil - 4$, and by Lemma 3 vertices of different sets W_j which are colored by the colors from $[d+1,k]$ are at distance $2\lceil \frac{k}{2} \rceil > k = 4n + 2d - 1 = 2n + 1 + r \geq 2i + 1 + r$ for any $i \in [1,n]$. Also if a color $c \in [d+1,,k]$ is used for coloring of vertices of W_j, then it can not be used on any set U_i.

Finally, we complete the packing k-coloring of T_Φ on the vertex u and the vertices from the paths between u and trees T_{X_i} and T_{C_j}. We proceed similarly as in the previous constructions — color u by 4, and use pattern $1,2,1,3,1,2,\ldots$ on the paths, with possible one more appearance of the color 4, if necessary. □

Since trees S_i have $O(k^2)$ vertices, and trees T_i have $O(k^3)$ vertices, the final tree T_Φ has $O(n^4(n+m))$ vertices. Hence our reduction is polynomial and the proof of Theorem 1 is finished.

3 Proof of Theorem 2

Without loss of generality assume that s_r is the last element of S smaller than t. For every $k \leq r$ the S-PACKING COLORING problem can be solved polynomially for trees (and for graphs of restricted treewidth), e.g., by the machinery of MSOL.

We construct a dynamic programming algorithm under assumptions that $k > r$ and $s_2 > 1$. (If $s_2 = 1$, then two color classes always suffices for any tree; one class can be used only if the tree has only one vertex.)

Assume that T is a rooted tree on n vertices. If W is a subset of children of some node v, then we denote by $T_{v,W}$ subtree of T rooted in v and containing all vertices from W together with all their descendants.

For a tree $T_{v,W}$ we explore all its partial $(s_1, \dots, s_r)$-packing colorings with respect to the following parameters:

– the distances d_i between the root v and the closest vertex from the i-th class for every $i \in [1, r]$; it's only essential to know the distance only when $d_i \leq s_i$,
– the numbers p_j of uncolored vertices that are at distance at most $j \leq t$ from v for every $j \in [0, t]$.

We encode these two sets of parameters by sequences $D = (d_1, d_2, \dots, d_{r-1})$ such that $d_i \in [0, s_i] \cup \{\infty\}$, and $P = (p_0, p_1, \dots, p_t)$ such that $0 \leq p_1 \leq p_2 \leq \dots \leq n$.

Among those partial colorings that provide the same parameters we identify the maximal number of uncolored vertices that are pairwise at distance smaller than t and choose the coloring that minimizes this value. In particular, our algorithm computes for each triple $T_{v,W}, D, P$ the minimal integer $c(T_{v,W}, D, P)$ such that there is a partition of $V(T_{v,W})$ into sets $X_1, \dots, X_r, Y$ for which the following conditions are fulfilled:

– for every $i \in [1, r]$ the set X_i is an S_i packing in $T_{v,W}$,
– for every $i \in [1, r] : d_i = \min\{\text{dist}(v, z) \colon z \in X_i, \text{dist}(v, z) \leq s_i\}$;
it is assumed that $d_i = \infty$ if no such z exists,
– for every $j \in [0, t] : p_j = |\{z \in Y \colon \text{dist}(v, z) \leq j\}|$,
– for every $Z \subset Y$, satisfying $u, v \in Z : \text{dist}(u, v) \leq t$, holds that $|Z| \leq c(T_{v,W}, D, P)$.

If no such partition exists, then we define $c(T_{v,W}, D, P) = \infty$.

The sequences D are used to properly extend partial packing colorings, while sequences P allow us to determine the maximum size of the set Z. In other words Z induces a clique in the t-th power of T. (In the t-th power vertices are adjacent if and only if they are at distance at most t in the original graph). Here we strongly rely on the well known fact that powers of trees are chordal [8,9], and their chromatic numbers are equal to the size of their maximum clique.

The algorithm consists from three subroutines. The first subroutine `Leaf` is called if T_v has only one vertex v (i.e. v is a leaf of T).

The subroutine `NewRoot` is called for a vertex v with a child w, and it computes $c(T_{v,\{w\}}, D, P)$ from the values of $c(T_{w,N(w)}, D', P')$. Here $N(w)$ stands for the set of children of w.

The last subroutine `Join` is called for vertices of T which are not leaves. It computes from the tables of values $c(T_{v,W_i}, D_i, P_i)$ for two subtrees T_{v,W_1} and T_{v,W_2} with a unique common vertex v, which is the root of the trees, the value of $c(T_{v,W}, D, P)$ for the union of these trees $T_{v,W}$, where $W = W_1 \cup W_2$.

```
Subroutine Leaf($T_v, D, P$);
if $d_1 = d_2 = \cdots = d_r = \infty$ and $p_0 = p_1 = \cdots = p_t = 1$ then
    $c(T_{v,\emptyset}, D, P) := 1$;
else
    if $\exists i \in [1, r]$ such that $d_i = 0$ and $d_j = \infty$ for all $j \in [1, r] \setminus \{i\}$,
        and $p_0 = p_1 = \cdots = p_t = 0$ then
        $c(T_{v,\emptyset}, D, P) := 0$;
    else
        $c(T_{v,\emptyset}, D, P) := \infty$;
Return $c(T_{v,\emptyset}, D, P)$
```

```
Subroutine NewRoot($T_{v,\{w\}}, D, P$);
$c(T_{v,\{w\}}, D, P) := \infty$;
if ( $\exists i \in [1, r]$ such that $d_i = 0$ and $d_j > 0$ for all $j \in [1, r] \setminus \{i\}$, and $p_0 = 0$ )
    or ( $d_j > 0$ for all $j \in [1, r]$ and $p_0 = 1$ ) then
    for $j := 1$ to $t$ do
        $p'_{j-1} := p_j$;
    $J := \{j \in [1, r] \colon d_j = 0$ or $d_j = \infty\}$;
    forall $j \in [1, r] \setminus J$ do
        $d'_j := d_j - 1$;
    for $p'_t := p'_{t-1}$ to $n$ do
        $P' := (p'_0, p'_1, \ldots, p'_t)$;
        for every choice $d'_j \in \{s_j, \infty\}$ for all $j \in J$ do
            $D' := (d'_1, d'_2, \ldots, d'_r)$;
            if $c(T_{v,\{w\}}, D, P) > c(T_{w,N(w)}, D', P')$ then
                $c(T_{v,\{w\}}, D, P) := c(T_{w,N(w)}, D', P')$;
Return $c(T_{v,\{w\}}, D, P)$
```

Our algorithm starts from leaves of the tree T and constructs for them tables of values $c(T_{v,\emptyset}, D, P)$ by the subroutine `Leaf`. If v is not a leaf, then we use the subroutine `NewRoot` if it has only one child. If v has more children $w_1, w_2, \ldots, w_l$, then the subroutine `NewRoot` is used for the construction of auxiliary tables for values $c(T_{v,\{w_i\}}, D, P)$ for all $i \in [1, l]$. Consequently, we use the subroutine `Join` and construct consecutively tables for trees $T_{v,\{w_1,w_2,\ldots,w_i\}}$ for $i = 2, 3, \ldots, l$. Finally, table is constructed for the root u. If there are D and P for which $c(T_{u,N(u)}, D, P) + r > k$, then the tree T allows an S-packing k-coloring. Otherwise no such coloring exists.

Now we estimate the time complexity. Since the sequence S is fixed, there is a constant number of sequences D. There are $O(n^{t+1})$ sequences P and all such sequences can be listed in time $O(n^{t+2})$. Note that we have to list these sequences only once. The construction of the tables with all values $c(T_{v,\emptyset}, D, P)$ for leaves of T by the subroutine `Leaf` demands $O(n^{t+2})$ operations, since T has no more than n leaves. Each call of the subroutine `NewRoot` takes $O(n)$ operations. Since we use this subroutine for every edge of T, the total number of operations is

```
Subroutine Join($T_{v,W_1}, T_{v,W_2}, D, P$);
$c(T_{v,W}, D, P) := \infty$;
if $\exists i \in [1,r]$ such that $d_i = 0$ and $d_j > 0$ for all $j \in [1,r] \setminus \{i\}$, and $p_0 = 0$ then
    forall $P_1 := (p_0^{(1)}, p_1^{(1)}, \dots, p_t^{(1)})$ and $P_2 := (p_0^{(2)}, p_1^{(2)}, \dots, p_t^{(2)})$
        such that $p_j^{(1)} + p_j^{(2)} = p_j$ for all $j \in [0,t]$ do
        forall $D_1 := (d_1^{(1)}, d_2^{(1)}, \dots, d_r^{(1)})$ and $D_2 := (d_1^{(2)}, d_2^{(2)}, \dots, d_r^{(2)})$
            such that $d_i = \min\{d_j^{(1)}, d_j^{(2)}\}$ for all $j \in [1,r]$,
            and $d_j^{(1)} + d_j^{(2)} > s_j$ for all $j \in [1,r] \setminus \{i\}$ do
            $m := \max\{c(T_{v,W_1}, D_1, P_1), c(T_{v,W_2}, D_2, P_2)\}$;
            for $j := 0$ to $t$ do
                if $m < p_j^{(1)} + p_{t-j}^{(2)}$ then $m := p_j^{(1)} + p_{t-j}^{(2)}$;
            if $c(T_{v,W}, D, P) > m$ then $c(T_{v,W}, D, P) := m$;
if $d_i > 0$ for all $i \in [1,r]$ and $p_0 = 1$ then
    forall $P_1 := (p_0^{(1)}, p_1^{(1)}, \dots, p_t^{(1)})$ and $P_2 := (p_0^{(2)}, p_1^{(2)}, \dots, p_t^{(2)})$
        such that $p_j^{(1)} + p_j^{(2)} = p_j$ for all $j \in [1,t]$ and $p_0^{(1)} = p_0^{(2)} = 1$ do
        forall $D_1 := (d_1^{(1)}, d_2^{(1)}, \dots, d_r^{(1)})$ and $D_2 := (d_1^{(2)}, d_2^{(2)}, \dots, d_r^{(2)})$
            such that $d_i = \min\{d_j^{(1)}, d_j^{(2)}\}$ and $d_j^{(1)} + d_j^{(2)} > s_j$ for all $j \in [1,r]$ do
            $m := \max\{c(T_{v,W_1}, D_1, P_1), c(T_{v,W_2}, D_2, P_2)\}$;
            for $j := 0$ to $t$ do
                if $m < p_j^{(1)} + p_{t-j}^{(2)} - 1$ then $m := p_j^{(1)} + p_{t-j}^{(2)} - 1$;
            if $c(T_{v,W}, D, P) > m$ then $c(T_{v,W}, D, P) := m$
Return $c(T_{v,W}, D, P)$
```

$O(n^{t+2})$. At every call of the subroutine Join all possible sequences P_1, P_2, D_1 and D_2 are considered. For any sequence P there are $O(n^{t+1})$ such sequences, and all such sequences for all P can be listed in time n^{2t+3}. We can use this table of sequences for the all calls of the subroutine. Every call of the subroutine demands $O(n^{t+1})$ operations, and the table of all values of $c(T_{v,W}, D, P)$ can be constructed in time $O(n^{2t+2})$. The total number of such tables is no more than the number of edges of T. Correspondingly, the total number of operations is $O(n^{2t+3})$.

4 Conclusion and Open Problems

We have shown that for bounded sequences the S-Packing Coloring problem is solvable in polynomial time for trees. On the other hand, Theorem 1 shows that it is NP-complete for the sequence $(1, 2, 3, \dots)$. It would be interesting to classify computational complexity of the S-Packing Coloring problem for different sequences. It can be easily seen that the proof of Theorem 1 can be extended for sequences $S = (s_1, s_2, s_3, \dots)$ of different positive integers such that $s_i = \Theta(i^c)$ for some constant c.

As a consequence of a result of Sloper [11] and Corollary 1 the packing chromatic number can be computed polynomially for trees of maximum degree three. This raises the question whether χ_p can be determined efficiently for bounded degree trees.

Acknowledgments. The research was initiated at WFAP'07 — Second Workshop on Frequency Assignment Problems in Wireless Networks organized in September 23–27, 2007 at Sádek u Třebíče by DIMATIA and ITI, whose generous support of both authors is gratefully acknowledged.

References

1. Arnborg, S., Lagergren, J., Seese, D.: Easy problems for tree-decomposable graphs. Journal of Algorithms 12, 308–340 (1991)
2. Borie, R.B.: Generation of polynomial-time algorithms for some optimization problems on tree-decomposable graphs. Algorithmica 14, 123–137 (1995)
3. Brešar, B., Klavžar, S., Rall, D.F.: On the packing chromatic number of cartesian products, hexagonal lattice, and trees. Discrete Applied Mathematics 155, 2303–2311 (2007)
4. Courcelle, B.: The monadic second-order logic of graphs iii: tree-decompositions, minor and complexity issues. RAIRO Informatique Théorique et Applications 26, 257–286 (1992)
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. Garey, M.R., Johnson, D.S.: Computers and Intractability.A Guide to the Theory of NP-Completeness. W.H. Freeman and Company, San Francisco (1979)
7. Goddard, W., Hedetniemi, S.M., Hedetniemi, S.T., Harris, J.M., Rall, D.F.: Broadcast chromatic numbers of graphs. Ars Combinatoria 86, 33–49 (2008)
8. Kearney, P.E., Corneil, D.G.: Tree powers. Journal of Algorithms 29, 111–131 (1998)
9. Lin, Y.L., Skiena, S.S.: Algorithms for square roots of graphs. SIAM Journal on Discrete Mathematics 8, 99–118 (1995)
10. Rall, D.F., Finbow, A.: On the packing chromatic number of some infinite graphs. Discrete Applied Mathematics (to appear, 2007)
11. Sloper, C.: An eccentric coloring of trees. The Australasian Journal of Combinatorics 29, 309–321 (2004)

Characterizations of Restricted Pairs of Planar Graphs Allowing Simultaneous Embedding with Fixed Edges

J. Joseph Fowler[1,⋆], Michael Jünger[2,⋆⋆], Stephen Kobourov[1,⋆], and Michael Schulz[2,⋆⋆]

[1] University of Arizona, USA
{jfowler,kobourov}@cs.arizona.edu
[2] University of Cologne, Germany
{mjuenger,schulz}@informatik.uni-koeln.de

Abstract. A set of planar graphs share a *simultaneous embedding* if they can be drawn on the same vertex set V in the Euclidean plane without crossings between edges of the same graph. *Fixed edges* are common edges between graphs that share the same simple curve in the simultaneous drawing. Determining in polynomial time which pairs of graphs share a simultaneous embedding with fixed edges (SEFE) has been open.

We give a necessary and sufficient condition for whether a SEFE exists for pairs of graphs whose union is homeomorphic to K_5 or $K_{3,3}$. This allows us to characterize the class of planar graphs that always have a SEFE with any other planar graph. We also characterize the class of biconnected outerplanar graphs that always have a SEFE with any other outerplanar graph. In both cases, we provide efficient algorithms to compute a SEFE. Finally, we provide a linear-time decision algorithm for deciding whether a pair of biconnected outerplanar graphs has a SEFE.

1 Introduction

In many practical applications including the visualization of large graphs and very-large-scale integration (VLSI) of circuits on the same chip, edge crossings are undesirable. A single vertex set can be used with multiple edge sets that each correspond to different edge colors or circuit layers. While the pairwise union of all edge sets may be non-planar, a planar drawing of each layer may be possible, as crossings between edges of distinct edge sets are permitted. Finding such drawings is the basic problem of *simultaneous embedding* (SE) and this can be viewed as a generalization of the notion of planarity to multiple graphs.

Without restrictions on the types of edges, any number of planar graphs can be drawn on the same fixed set of vertex locations [13]. However, difficulties arise once straight-line edges are required. This is the problem of *simultaneous geometric embedding* (SGE). If edge bends are allowed, then having common

⋆ This work is supported in part by NSF grants CCF-0545743 and ACR-0222920.
⋆⋆ This work is supported in part by the German Science Foundation (JU204/11-1) and by the German Academic Exchange Service.

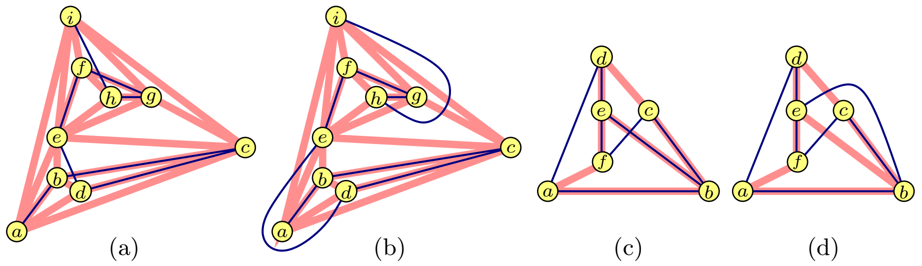


Fig. 1. The path and planar graph in (a) do not have a SGE with straight-line edges [2], but have a SEFE in (b). The two outerplanar graphs in (c) do not have a SEFE, but have a SE in (d) if the edge (b, e) is not fixed.

edges drawn in the same way using the same simple curve preserves the "mental map". Such edges are called fixed edges leading to the problem of *simultaneous embedding with fixed edges* (SEFE). Since straight-line edges between a pair of vertices are also fixed edges, any graph that has a SGE also has a SEFE, but the converse is not true; see Fig. 1 that shows SGE $\subset$ SEFE $\subset$ SE.

Deciding whether two graphs have a SGE is NP-hard [6], while deciding whether three graphs have a SEFE is NP-complete [9]. However, deciding whether two graphs have a SEFE in polynomial-time remains open. We give a necessary condition in terms of forbidden minors for when pairs of graphs can have a SEFE. This leads to a polynomial-time decision algorithm in the restricted case of pairs of biconnected outerplanar graphs. We also characterize the class of biconnected outerplanar graphs that always have a SEFE with any other outerplanar graph. Finally, we characterize the graphs that always have a SEFE with any planar graph and compute a SEFE when possible.

1.1 Related Work

Any number of stars, two caterpillars (trees whose removal of all leaves gives a path) and two cycles always have a SGE, whereas three paths and two trees may not [2,10]. Which graphs always have a SGE with a path, a caterpillar, a tree, or a cycle remains unknown. For the case of SEFE, a planar graph and a tree always have a SEFE, whereas two outerplanar graphs do not [8]. This shows that the topological problem of SEFE is less restricted than the geometric problem of SGE. Note that this is unlike standard planarity where the sets of topological and geometric planar graphs are identical [5]. Planar graphs are characterized in terms of the forbidden graphs, K_5 and $K_{3,3}$, which form two minimum examples of non-planarity [12,14]. No similar characterization for SEFE in terms forbidden pairs has been given until now, even for restricted pairs of planar graphs.

1.2 Our Contribution

1. We show there exist three paths without a SEFE. We provide a necessary and sufficient condition in terms of 17 minimal forbidden pairs for when a pair of graphs whose union forms a subdivided K_5 or $K_{3,3}$ has a SEFE.

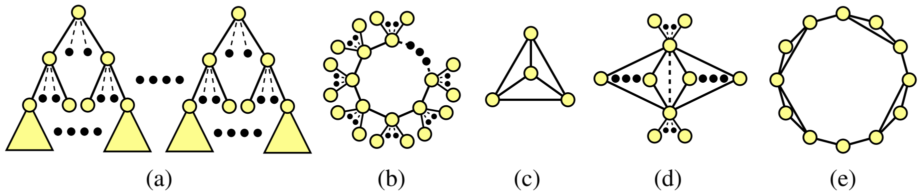


Fig. 2. Forests in (a), circular caterpillars (removal of all degree-1 vertices yields a cycle) in (b), K_4 in (c) and *subgraphs* of K_3-multiedges (an edge with any number of incident edges) in (d) have a SEFE with any planar graph. K_3-cycles (n-cycles with chords that form 3-cycles with the n-cycles) as in (e) have a SEFE with any outerplanar graph.

2. Using this condition, we characterize the class of planar graphs that have a SEFE with any planar graph to be the set of (i) forests, (ii) circular caterpillars, (iii) K_4, and (iv) subgraphs of K_3-multiedges; see Fig. 2(a)–(d). We efficiently compute a SEFE in each case. We show that any other graph not in this class contains a subgraph homeomorphic to a cycle and a disjoint edge. We provide a similar characterization for the class of biconnected outerplanar graphs that always share a SEFE with any outerplanar graph; see Fig. 2(e). Table 1 summarizes our results.
3. We determine which pairs of biconnected outerplanar graphs can have a SEFE using a forbidden outerplanar pair. This leads to a linear-time decision algorithm for this restricted case.

1.3 Preliminaries

Let P be a set of n distinct points in the plane $\mathbb{R}^2$. A *planar drawing* of $G(V,E)$ with $|V| = n$ on P consists of a bijection $\sigma : V \to P$ with a simple curve for each edge $(u,v) \in E$ drawn in the plane $\mathbb{R}^2$ connecting the points $\sigma(u)$ and $\sigma(v)$ with curves that only intersect at endpoints. Let $\mathcal{G} = \{G_1(V,E_1), G_2(V,E_2), \ldots, G_k(V,E_k)\}$. $\mathcal{G}$ has a *simultaneous embedding* (SE) if there exist planar drawings of $G_i(V,E_i)$ with the same bijection $\sigma : V \to P$. If each edge is a straight-line segment, then $\mathcal{G}$ has a *simultaneous geometric embedding* (SGE). If every

Table 1. Old and new results for SGE and SEFE pairs. The shaded pairs are new.

	SGE		SEFE				
	Path	Tree	Forest	Circular caterpillar	K_4	K_3-multiedge	K_3-cycle
Path	✓ [2]	?	✓ [8]	✓ [8]	✓ [8]	✓ [8]	✓ [8]
Caterpillar	✓ [2]	?	✓ [8]	✓ [8]	✓ [8]	✓ [8]	✓ [8]
Tree	?	✗ [10]	✓ [8]	✓ [8]	✓ [8]	✓ [8]	✓ [8]
Outerplanar	?	✗ [10]	✓	✓	✓	✓	✓
Planar	✗ [2]	✗ [2,10]	✓	✓	✓	✓	✗

common edge in $\mathcal{G}$ connecting a pair of vertices uses the same simple curve, then $\mathcal{G}$ has a *simultaneous embedding with fixed edges* (SEFE).

In a graph $G(V, E)$, *subdividing* an edge $(u, v) \in E$ replaces edge (u, v) with the pair of edges (u, w) and (w, v) in E by adding w to V. A *subdivision* of G is obtained through a series of edge subdivisions. *Contraction* of edge (u, v) replaces the vertices u and v with the vertex w that is adjacent to all the vertices that were adjacent to either u or v. A *minor* H of G is obtained through a series of edge contractions and edge deletions. A graph $G(V, E)$ is *isomorphic* to a graph $\tilde{G}(\tilde{V}, \tilde{E})$ if there exists a bijection $f : V \rightarrow \tilde{V}$ such that $(u, v) \in E$ if and only if $\big(f(u), f(v)\big) \in \tilde{E}$. A graph $G(V, E)$ is *homeomorphic* to a graph $\tilde{G}(\tilde{V}, \tilde{E})$ if the subdivisions of G and $\tilde{G}$ are isomorphic.

2 Forbidden Simultaneous Embeddings with Fixed Edges

We begin with Kuratowski's and Wagner's planar graph theorems [12,14].

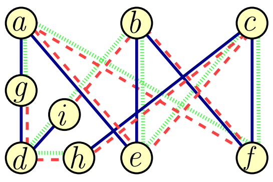
Theorem 1 (Kuratowski, Wagner). *A graph is non-planar if and only if it has a subgraph homeomorphic to K_5 or $K_{3,3}$ or has K_5 or $K_{3,3}$ as a minor.*

2.1 Forbidden Triples of Paths and Cycles

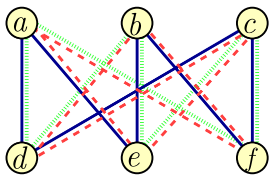
Next we show that the triples without a SGE of three paths in [2] and three cycles in [1] extend to the case of SEFE.

Theorem 2. *There exist three paths on 9 vertices and three cycles on 6 vertices without a SEFE.*

Proof. Consider the three paths $g–d–h–c–e–a–f–b–i$, $h–d–i–b–e–c–f–a–g$, and $i–d–g–a–e–b–f–c–h$ and the three cycles $a–d–c–f–b–e–a$, $a–e–c–d–b–f–a$, and $a–f–c–e–b–d–a$ shown in Fig. 3. In both cases, the union forms a subdivided $K_{3,3}$ and any drawing must have a crossing by Theorem 1. Each edge in the union belongs to two paths (or two cycles). Such a crossing must be between two pairs of paths (or cycles). Since there are only three paths (or three cycles) and fixed edges are being used, one path (or cycle) must self-intersect. □



(a) 3 paths on 9 vertices



(b) 3 cycles on 6 vertices

Fig. 3. Two graph triples without a SEFE

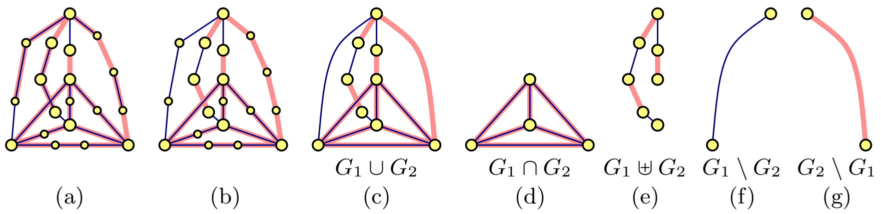


Fig. 4. Removing extraneous edges from (a) gives (b). Unsubdividing degree-2 vertices in (b) gives (c) that can be partitioned into the four subgraphs in (d)–(g).

2.2 Minimal Forbidden Pairs

Suppose a pair of graphs $G_1(V, E_1)$ and $G_2(V, E_2)$ does not have a SEFE as in Fig. 4(a). If deleting any edge from either graph allows a SEFE, then G_1 and G_2 are *edge minimal* as in Fig. 4(b). If a degree-2 vertex v (adjacent to u and w) in the union of G_1 and G_2 is not a degree-1 vertex in either G_1 or G_2, then we can *unsubdivide* the vertex by deleting v and replacing edges (u, v) and (v, w) with the edge (u, w) in G_1 and/or G_2. A pair of graphs for which this can no longer be done is *vertex minimal* as in Fig. 4(c). A *minimal forbidden pair* does not have a SEFE and is edge and vertex minimal.

We define the *union* $G_1 \cup G_2$ and the *intersection* $G_1 \cap G_2$ as having edge sets $E_1 \cup E_2$ and $E_1 \cap E_2$, respectively; see Fig. 4(c)–(d). Suppose then that $G_1 \cup G_2$ is homeomorphic to a graph G with no degree-2 vertices. Let $u \rightsquigarrow v$ in $G_1 \cup G_2$ be the path corresponding to the subdivided edge (u, v) in G. Path *$u \rightsquigarrow v$ is incident to $x \rightsquigarrow y$* in $G_1 \cup G_2$ if and only if (u, v) is incident to (x, y) in G. An *alternating edge* is a $u \rightsquigarrow v$ path in which the edges strictly alternate between being in either G_1 or G_2; see Fig. 4(e). An *exclusive edge* is a $u \rightsquigarrow v$ path composed of the edge (u, v) that is only in G_1 or G_2; see Fig. 4(f)–(g), while an *inclusive edge* is composed of the fixed edge (u, v) in $G_1 \cap G_2$; see Fig. 4(d).

Lemma 3. *Any pair of graphs $G_1(V, E_1)$ and $G_2(V, E_2)$ can be reduced to a pair in which every $u \rightsquigarrow v$ path is either an inclusive, exclusive, or alternating edge.*

Proof. We examine each $u \rightsquigarrow v$ path p in $G_1 \cup G_2$. If path p is in $G_1 \cap G_2$, we replace p with a single inclusive edge (u, v) in both G_1 and G_2. If p is in G_i but is missing edges in G_j for $i \neq j$, we replace it with the single exclusive edge (u, v) in G_i. If p is missing an edge from each graph, we make p into an alternating edge by deleting edges from p in either G_1 or G_2 until each edge along p is no longer in $G_1 \cap G_2$. Then we unsubdivide p until it is strictly alternating. We can always avoid crossings along edges of $u \rightsquigarrow v$ paths contained in $G_1 \cap G_2$ reduced in this way. Hence, neither operation changes whether the pair has a SEFE. $\square$

Suppose G_1 and G_2 are a *reduced pair*, which is a pair of graphs where all $u \rightsquigarrow v$ paths have been reduced. The *alternating edge subgraph*, $G_1 \uplus G_2$, is the subgraph of $G_1 \cup G_2$ consisting only of alternating edges. The *exclusive edge subgraph* of G_1, $G_1 \setminus G_2$, is the subgraph of $G_1 \cup G_2$ consisting of exclusive edges from G_1, where $G_2 \setminus G_1$ is defined analogously. Hence, edges of $G_1 \cup G_2$ are partitioned into $G_1 \cap G_2$, $G_1 \uplus G_2$, $G_1 \setminus G_2$, and $G_2 \setminus G_1$; see Fig. 4(d)–(g). Next we see why we only need to consider crossings between non-incident edges.

Observation 4. *Crossings between incident edges in a non-planar drawing can be removed without affecting the number of crossings of non-incident edges.*

This can be done by swapping the simple curves from the incident vertex to the first intersection point p. Separating the curves at p by a small distance eliminates the crossing without affecting the rest of the drawing. Repeating this process removes all crossings of incident edges. Hence, we only need to consider crossings of non-incident edges in a simultaneous drawing with fixed edges. Removing an edge from either K_5 or $K_{3,3}$ of Theorem 1, allows a planar embedding. Only one crossing needs to be introduced when replacing the edge, since there is at most one edge separating any pair of faces in the embedding. This fact along with Observation 4 gives the next corollary.

Corollary 5. *(a) Every drawing of K_5 or $K_{3,3}$ has a crossing between non-incident edges. (b) K_5 or $K_{3,3}$ can be drawn with only one crossing between any pair of non-incident edges.*

We use this corollary to produce a sufficient condition for SEFE.

Lemma 6. *Suppose the union $G_1 \cup G_2$ of a reduced pair (G_1, G_2) is homeomorphic to K_5 or $K_{3,3}$. Let $u \rightsquigarrow v$ and $x \rightsquigarrow y$ be non-incident paths in $G_1 \cup G_2$ but not in $G_1 \cap G_2$. If either path belongs to $G_1 \uplus G_2$ or one belongs to $G_1 \setminus G_2$ and the other belongs to $G_2 \setminus G_1$, then G_1 and G_2 have a SEFE.*

Proof. By Corollary 5(b), a K_5 or a $K_{3,3}$ can always be drawn so that only (u, v) and (x, y) cross. Hence, there is a SEFE in which an alternating edge in $G_1 \uplus G_2$ only crosses an edge in either $G_1 \setminus G_2$ or $G_2 \setminus G_1$. Likewise, an edge in $G_1 \setminus G_2$ can cross any non-incident edge in $G_2 \setminus G_1$. □

With Lemma 6 we determine when a K_5 or a $K_{3,3}$ pair has a SEFE.

Corollary 7. *Suppose the union $G_1 \cup G_2$ of a reduced pair (G_1, G_2) is homeomorphic to K_5 or $K_{3,3}$. The pair (G_1, G_2) has no SEFE if and only if (i) every non-incident edge of an alternating edge in $G_1 \uplus G_2$ is in $G_1 \cap G_2$ and (ii) every non-incident edge of an exclusive edge in $G_1 \setminus G_2$ is in G_1.*

Proof. For necessity, suppose the pair (G_1, G_2) does not have a SEFE. Consider an $x \rightsquigarrow y$ path in $G_1 \cup G_2$ that is non-incident to an alternating edge $u \rightsquigarrow v$ in $G_1 \uplus G_2$ in which $x \rightsquigarrow y$ is not in $G_1 \cap G_2$. By Lemma 6, the pair (G_1, G_2) would have a SEFE since $u \rightsquigarrow v$ is in $G_1 \uplus G_2$ and neither path is in $G_1 \cap G_2$. Next consider an $x \rightsquigarrow y$ path in $G_1 \cup G_2$ that is non-incident to an exclusive edge (u, v) in $G_1 \setminus G_2$ in which $x \rightsquigarrow y$ is not in G_1. By Lemma 6, the pair (G_1, G_2) again would have a SEFE since $x \rightsquigarrow y$ either is in $G_1 \uplus G_2$ or is in $G_2 \setminus G_1$.

For sufficiency, suppose conditions (i) and (ii) hold. Since the union forms a subdivided K_5 or $K_{3,3}$, by Corollary 5(a) at least one pair of non-incident paths

$u \rightsquigarrow v$ and $x \rightsquigarrow y$ cross. If either is in $G_1 \cap G_2$, then there must be a crossing in G_1 or G_2. If either is in $G_1 \uplus G_2$, then by (i) the other would be in $G_1 \cap G_2$, again giving a crossing in G_1 or G_2. If both are in $G_i \setminus G_j$ for $i \neq j$, then there is a crossing in G_i. Finally, (ii) prevents one edge being in $G_1 \setminus G_2$ and the other edge being in $G_2 \setminus G_1$. Hence, G_1 and G_2 do not have a SEFE. $\square$

Theorem 8. *There are 17 minimal forbidden pairs with a union homeomorphic to K_5 or $K_{3,3}$.*

Proof. Let $G_{i,j}$ denote the 17 pairs of graphs for $i \in \{1, \dots, 17\}$ and $j \in \{1, 2\}$ in Figs. 5 and 6. One can verify that all the non-incident edges of any alternating edge are in the intersection and every edge non-incident to an exclusive edge of $G_{i,1}$ is also in $G_{i,1}$. This satisfies Corollary 7 implying that none of these pairs has a SEFE. Removing any edge means either (i) the union no longer forms a K_5 or a $K_{3,3}$ or (ii) the intersection does not contain all the non-incident edges of $G_{i,1} \uplus G_{i,2}$ or of $G_{i,1} \setminus G_{i,2}$ (other than those already in $G_{i,1}$) so that Corollary 7 is no longer satisfied. This implies that all 17 forbidden pairs are minimal.

We next show that our 17 pairs are the only minimal forbidden pairs homeomorphic to K_5 or $K_{3,3}$. Assume w.l.o.g. (G_1, G_2) is a reduced minimal forbidden pair whose union forms a K_5 or a $K_{3,3}$ where G_2 has at least as many edges as G_1. We consider all the possibilities for edges to be in $G_1 \setminus G_2$ or $G_1 \uplus G_2$.

Pairs $(G_{1,1}, G_{1,2})$, $(G_{2,1}, G_{2,2})$, $(G_{12,1}, G_{12,2})$, and $(G_{13,1}, G_{13,2})$ are the only possibilities in which there is one exclusive edge in G_1 or one alternating edge in $G_1 \uplus G_2$. Two non-incident alternating edges would violate Corollary 7. The other case of two non-incident edges that are exclusive in G_1 is given by pairs $(G_{6,1}, G_{6,2})$ and $(G_{14,1}, G_{14,2})$. Three non-incident edges are only possible in a $K_{3,3}$, but adding all of their non-incident edges implies that G_1 is a $K_{3,3}$.

For the case of $G_1 \cup G_2$ homeomorphic to K_5, the pairs $(G_{3,1}, G_{3,2})$, $(G_{4,1}, G_{4,2})$, and $(G_{5,1}, G_{5,2})$ give the three possibilities of two incident edges that are exclusive and/or alternating. Two incident exclusive edges with a third

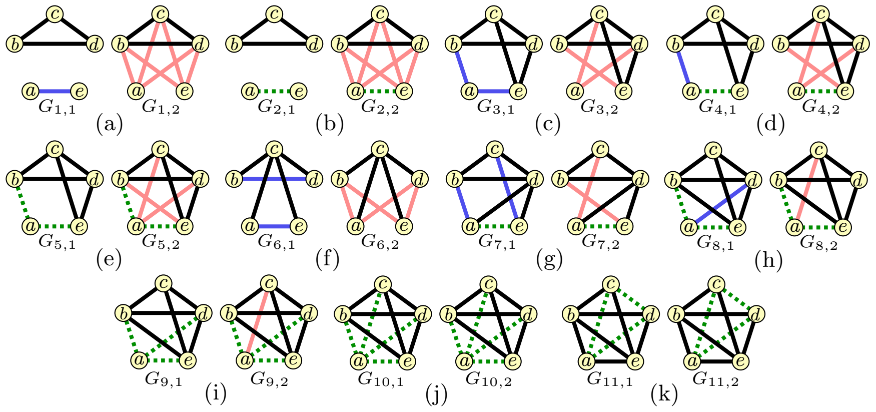


Fig. 5. Eleven K_5 minimal forbidden pairs

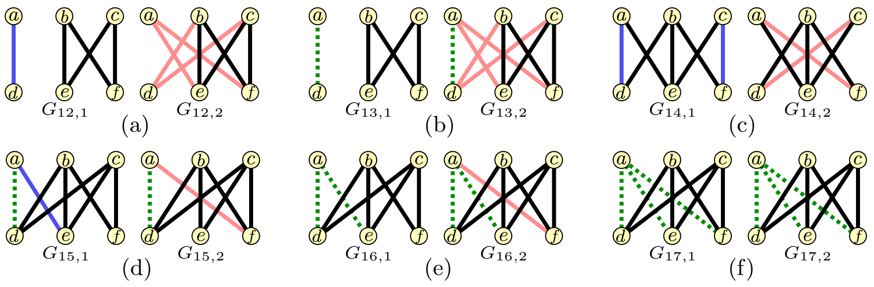


Fig. 6. Six $K_{3,3}$ minimal forbidden pairs

exclusive or alternating edge cannot happen since $G_{3,1}$ has seven edges with two incident exclusive edges. Adding another exclusive or alternating edge along with its non-incident edge would imply that $|G_2 \setminus G_1| = |G_1 \cup G_2| - |G_1| - |G_1 \setminus G_2| = 10 - 7 - 2 = 1$. This contradicts our assumption of G_2 having at least as many edges as G_1.

Two non-incident exclusive edges with a third incident alternating edge is given by the pair $(G_{7,1}, G_{7,2})$. Two or three alternating edges that are all incident with another exclusive or alternating edge are given by the pairs $(G_{8,1}, G_{8,2})$, $(G_{9,1}, G_{9,2})$ and $(G_{10,1}, G_{10,2})$, respectively. The last possibility of three alternating edges that are only pairwise incident is given by pair $(G_{11,1}, G_{11,2})$ in which all the non-incident edges of each alternating edge is in the intersection.

For the case of $G_1 \cup G_2$ homeomorphic to $K_{3,3}$, if there are two incident exclusive and/or alternating edges, then the third incident $u \rightsquigarrow v$ edge in the union is the only edge that can be in $G_2 \setminus G_1$. This is because edges non-incident to $u \rightsquigarrow v$ are also in G_1 implying that $G_2 \setminus G_1$ can only contain the edge (u, v). Hence, $|G_1 \setminus G_2| < |G_2 \setminus G_1| = 1$. Pairs $(G_{15,1}, G_{15,2})$ with one exclusive edge and one alternating edge and $(G_{16,1}, G_{16,2})$ with two alternating edges are the only possibilities for two incident edges. However, $u \rightsquigarrow v$ could be an alternating edge. The pair $G_{16,2}$ already has one exclusive edge with two incident alternating edges. This leaves three alternating edges that are all incident given by pair $(G_{17,1}, G_{17,2})$ as the final possibility. $\square$

Unlike standard planar graphs in which the set of forbidden minors is identical to the set of forbidden subdivisions by Theorem 1, the same is not true for SEFE. Fig. 7 shows three pairs with the same minor pair $(G_{7,1}, G_{7,2})$ in Fig. 7(a). Each pair is obtained by "uncontracting" vertex d to form the fixed edge (d_1, d_2) in Figs. 7(b)–(d). Fig. 7(b)–(c) are forbidden pairs, whereas, Fig. 7(d) is not.

Figs. 7(c)–(d) are examples in which a new fixed edge (a, d) is created from the exclusive edges (a, d_1) in $G_1 \setminus G_2$ and (a, d_2) in $G_2 \setminus G_1$ by contracting edge (d_1, d_2) to vertex d in Fig. 7(a). To avoid this, we define a *fixed edge minor pair* as a minor pair (H_1, H_2) of (G_1, G_2) that is obtained by only contracting edges in which no new fixed edges are created. Fig. 7(b) is an example in which Fig. 7(a) forms a fixed edge minor pair. This leads to the following corollary.

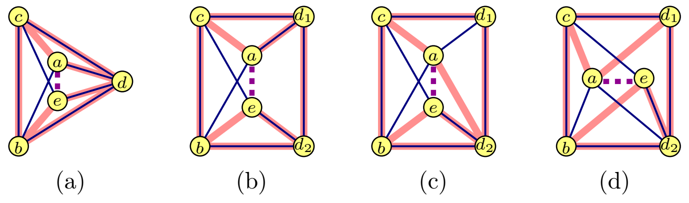


Fig. 7. The pair $(G_{7,1}, G_{7,2})$ in (a) is a minor pair of the two forbidden pairs in (b) and (c), which have no SEFE, as well as the pair in (d), which has the given SEFE

Corollary 9. *Pair (G_1, G_2) has no SEFE if the pair has a fixed edge minor pair (H_1, H_2) isomorphic to one of the 17 minimal forbidden pairs of Theorem 8.*

This forms a necessary condition for SEFE, but is insufficient since Fig. 7(c) does not have a SEFE, nor does it have any of the 17 fixed edge minor pairs.

3 Characterizing SEFE with Planar Graphs

We next determine the graphs that *always* have a SEFE with *any* planar graph and produce simultaneous drawings. Let $\mathcal{P}$ be the set of planar graphs and $\mathcal{P}_{\mathsf{SEFE}}$ be the subset of $\mathcal{P}$ containing forests, *circular caterpillars* (removal of all degree-1 vertices yields a cycle), K_4, and the subgraphs of K_3-*multiedges* (edge (x, y) with the incident edges (x, z) and/or (y, z) for each $z \in V \setminus \{x, y\}$).

Lemma 10. *G is in $\mathcal{P}_{\mathsf{SEFE}}$ if and only if G does not contain a subgraph homeomorphic to a K_3 and a disjoint edge.*

Proof. First, we show necessity. Let $G \in \mathcal{P}_{\mathsf{SEFE}}$ and let H be the graph consisting of a K_3 and a disjoint edge. A forest has no cycles unlike H. While a circular caterpillar has a cycle, all the other edges are incident to the cycle. A K_4 has four vertices while H has five. Finally, every subgraph of a K_3-multiedge with a cycle, either has a 3-cycle, $x \rightsquigarrow y \rightsquigarrow z \rightsquigarrow x$, or a 4-cycle, $x \rightsquigarrow z_1 \rightsquigarrow y \rightsquigarrow z_2 \rightsquigarrow x$, if there is no edge (x, y). In either case, every other edge is part of the cycle or is incident to x or y.

Let $\tilde{G} \in \mathcal{P} \setminus \mathcal{P}_{\mathsf{SEFE}}$. Showing that $\tilde{G}$ has a subgraph homeomorphic to H gives sufficiency. The graph $\tilde{G}$ must have a cycle since otherwise it would be a forest. Let C be a cycle in $\tilde{G}$ of maximum length, and let e be any edge in $\tilde{G} \setminus C$. Either the edge e is incident to C or the graph $\tilde{G}$ contains a subgraph homeomorphic to H. If the edge e forms a chord of C where C is a k-cycle for some $k > 4$, then there is a cycle C' formed by a path in C and the edge e. Thus, C would have a non-incident edge from the cycle C' so that $\tilde{G}$ would be homeomorphic to H.

Hence, all cycles in $\tilde{G}$ are 3-cycles or 4-cycles. Suppose C is a 3-cycle with another cycle C' in $\tilde{G}$. Either C and C' share an edge giving a longer cycle (contradicting the maximality of C) or C' has an edge non-incident to C. Hence, C must be a 4-cycle if $\tilde{G}$ has multiple cycles. If two 4-cycles C and C' only share a vertex or a single edge, then C would have a non-incident edge in C'. Hence,

C and C' must share two edges. If the two edges are non-incident, then C_1 and C_2 form a K_4. Thus, $\tilde{G}$ either forms a K_4 or all the 4-cycles share a common path consisting of the two incident edges (x, z) and (y, z). Thus, all 3-cycles have the common edge (x, y) if it exists. Any non-cycle edge e must be incident to all the cycles implying that e is either (x, z) or (y, z) for some vertex z of degree 1. Thus, if $\tilde{G}$ has multiple cycles but is not a K_4, then $\tilde{G}$ is a subgraph of some K_3-multiedge. Finally, if C is the only cycle, then all the vertices not in C have degree 1 so that $\tilde{G}$ is a circular caterpillar. □

Together Corollary 9 and Lemma 10 allow us to determine when a graph always has a SEFE with any planar graph with the following lemma:

Lemma 11. *A graph G has a SEFE with any planar graph if only if $G \in \mathcal{P}_{\mathsf{SEFE}}$.*

Proof. We prove necessity by showing that each $G_1 \in \mathcal{P} \setminus \mathcal{P}_{\mathsf{SEFE}}$ does not have a SEFE with every $G_2 \in \mathcal{P}$. In all the 17 pairs of Theorem 8, both graphs have a subgraph homeomorphic to $G_{1,1}$ that is a K_3 and a disjoint edge; see Fig. 5(a). By Lemma 10, we know that that G_1 contains a subgraph homeomorphic to $G_{1,1}$. Thus, (G_1, G_2) cannot have a SEFE by Corollary 9 in which G_2 contains a subgraph homeomorphic to $G_{1,2} \in \mathcal{P}$.

To show sufficiency, we must show that every graph in $G \in \mathcal{P}_{\mathsf{SEFE}}$ has a SEFE. We do this by showing how to efficiently compute a SEFE for the class of graphs in $\mathcal{P}_{\mathsf{SEFE}}$. Frati [8] gave an algorithm that finds a SEFE for forests and planar graphs without explicitly bounding the number of bends per edge. Our algorithm computes a SEFE by drawing each edge with a modification of the optimal Euclidean shortest path algorithm that runs in $O(n \log n)$ time [11]. The modification is to determine the shortest path among a set of line segments (that do not intersect except at endpoints) in the plane in which at least a distance (of arbitrarily small) ε is always left between the path and the endpoint of any segment. This can be done using Minkowski sums such that the minimum distance from each endpoint is $2^{n/i}\varepsilon$ in step i for $i \in [1..n]$.

For each step i, a new bend $b_{i,k}$ is either caused by an endpoint p_k of an edge or a bend $b_{j,k}$ from a previous step $2 \leq j < i$. However, for each such bend $b_{i,k}$ only at most two points in the set $\{p_k, b_{2,k}, \ldots b_{i-1,k}\}$ (the inner and outer ones) contribute—bends added more recently hide bends caused by the original point p_k in previous steps. Hence, each time we add edges, at most $O(n)$ new bends are being introduced. Since the size of the vertex set grows by $O(n)$ for each step, this gives an overall running time of $\sum_{i=1}^{n} O(i \cdot n \log i \cdot n) = O(n^2 \log n)$.

Let $G_1 \in \mathcal{P}_{\mathsf{SEFE}}$ and $G_2 \in \mathcal{P}$. First, we draw G_2 in $O(n)$ time. We then find an embedding of G_2 and draw G_2 on an $(n-2) \times (n-2)$ grid, both done in $O(n)$ time [3,5]. Some of the edges of G_1 were drawn with G_2. We can ignore the edges in $G_2 \setminus G_1$ as we draw the rest of G_1. For a forest or a circular caterpillar in which the cycle has not yet been drawn, there is a single face giving a shortest Euclidean path between any two vertices. For a circular caterpillar with the cycle already drawn, the remaining points either lie inside or outside of the cycle. All edges are incident to the cycle. Hence, a Euclidean path always exists from vertices of the cycle to vertices of degree 1. For a graph with multiple cycles, it is a K_4 or a subgraph of a K_3-multiedge with a 4-cycle C that has two vertices x and y of degree greater than 2. We finish drawing C. For K_4, one chord is drawn inside of C, while the other chord is drawn outside of C. For a K_3-multiedge, any path from x to y is either the edge (x, y) or the path $x \rightsquigarrow z \rightsquigarrow y$ from some degree-2 vertex z. The edge (x, y) can drawn inside of C to start. For the other paths, there must always exist Euclidean paths from x and y to the common vertex z that lies inside some cycle drawn so far. Any remaining edges must be incident to x or y in which a Euclidean path must also exist. □

Lemmas 10 and 11 together imply the following characterization:

Theorem 12. *The following two statements are equivalent: A graph has a* SEFE *with any planar graph if only if*

- *it does not contain a subgraph homeomorphic to a K_3 and a disjoint edge.*
- *it is either (i) a forest, (ii) a circular caterpillar, (iii) a K_4, or (iv) a subgraph of a K_3-multiedge.*

4 Characterizing SEFE with Outerplanar Graphs

We next determine which biconnected outerplanar graphs *always* have a SEFE with *any* other outerplanar graph. A *K_3-cycle* is an n-cycle C with chords such that every chord forms a 3-cycle with edges from C; see Fig. 2(e).

The following lemma provides an analogous result for biconnected outerplanar graphs with respect to the outerplanar graphs $\mathcal{O}$ that Lemma 10 does for the planar graphs $\mathcal{P}$. The omitted proof can be found in [7]. The set $\mathcal{O}_{\mathsf{SEFE}}$ of K_3-cycles is shown to be the set of biconnected outerplanar graphs that do not contain $(G_{14,1}, G_{14,2})$ as a fixed edge minor pair. This is the only pair of Theorem 8 in which both graphs are biconnected and outerplanar. The graphs $G_{14,1}$ and $G_{14,2}$ are both isomorphic to a 6-cycle with a chord that forms two 4-cycles.

Lemma 13. *G is in $\mathcal{O}_{\mathsf{SEFE}}$ if and only if G does not contain a subgraph homeomorphic to $G_{14,1}$.*

The omitted proof of the following lemma also appears in [7]. The key idea is to use Euclidean shortest paths again to draw each edge that is not in the intersection. Special care is taken for pairs of edges (x, z) and (y, z) when the chord (x, y) is in the intersection. First, the edge (y, z) is routed to x and then both edges proceed within a small distance of each other from vertex z. Remaining chords can always be drawn inside the outerface of the K_3-cycle since each has a degree-2 vertex z on the outerface that is adjacent to both endpoints.

Lemma 14. *A biconnected outerplanar graph G has a* SEFE *with any outerplanar graph if only if $G \in \mathcal{O}_{\mathsf{SEFE}}$.*

Lemmas 13 and 14 together give the following characterization:

Theorem 15. *The following two statements are equivalent: A biconnected outerplanar graph has a* SEFE *with any outerplanar graph if only if*

- *it does not contain a subgraph homeomorphic to $G_{14,1}$.*
- *it is a K_3-cycle.*

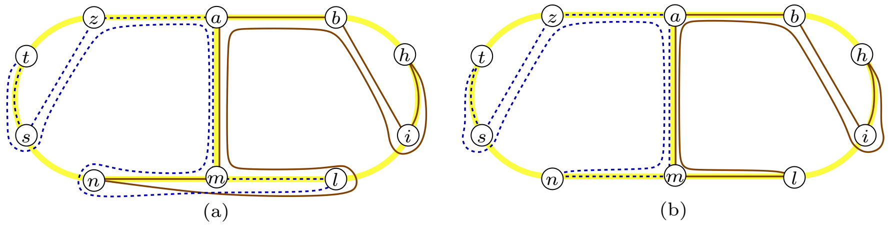


Fig. 8. Two biconnected outerplanar graphs with a common chord (a, m) do not have a SEFE in (a) given that (a, m) and its adjacent endpoints match the forbidden labeling of $(G_{14,1}, G_{14,2})$. The same pair in (b) has a SEFE since this is not the case.

5 Deciding SEFE for Biconnected Outerplanar Graphs

Corollary 9 provided a necessary but insufficient condition for the SEFE of two planar graphs. However, for the restricted case of two biconnected outerplanar graphs, we can give a necessary and sufficient condition.

Lemma 16. *The biconnected outerplanar graph pair (G_1, G_2) has a SEFE if and only if G_1 and G_2 does not have the fixed edge minor pair $(G_{14,1}, G_{14,2})$.*

The omitted proof found in [7] compares the labelings of the two outerfaces and the chords in the intersection to see if they match the forbidden labeling of the outerplanar graphs of $(G_{14,1}, G_{14,2})$; see Fig. 8. If so, the pair does not have a SEFE. Otherwise, an algorithm that runs in $O(n^2 \log n)$ time is given to produce a SEFE in which the cycles involving common chords in each graph are closed in such a way as to avoid any crossings.

Theorem 17. *Deciding whether a pair of biconnected outerplanar graphs (G_1, G_2) has a SEFE can be done in $O(n)$ time.*

The omitted proof found in [7] uses the conditions on the common chords in the intersection in the proof of Lemma 16. This condition can be checked in linear time, which yields a linear-time decision algorithm.

6 Conclusion

We gave a necessary condition for whether two graphs can have a SEFE in terms of 17 fixed edge minor pairs. This allowed us to characterize the graphs that always have a SEFE with any planar graph. We also characterized the class of biconnected outerplanar graphs that have a SEFE with any outerplanar graph. For the restricted case of two biconnected outerplanar graphs, deciding whether they have a SEFE can be done in linear-time.

While our results may be helpful in solving bigger open problems, there are still no known algorithms for testing whether a pair of planar graphs has a SEFE in polynomial time. Finding all fixed edge minor pairs of planar graphs would give a sufficient condition for their SEFE. This may lead to a polynomial-time decision algorithm, an improvement over the ILP crossing minimization algorithm in [4].

References

1. Brandes, U., Erten, C., Fowler, J., Frati, F., Geyer, M., Gutwenger, C., Hong, S., Kaufmann, M., Kobourov, S., Liotta, G., Mutzel, P., Symvonis, A.: Colored simultaneous geometric embeddings. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, pp. 254–263. Springer, Heidelberg (2007)
2. Brass, P., Cenek, E., Duncan, C.A., Efrat, A., Erten, C., Ismailescu, D., Kobourov, S.G., Lubiw, A., Mitchell, J.S.B.: On simultaneous graph embedding. Computational Geometry 36(2), 117–130 (2007)
3. Chiba, N., Nishizeki, T., Abe, S., Ozawa, T.: A linear algorithm for embedding planar graphs using PQ-trees. J. Comput. Syst. Sci. 30(1), 54–76 (1985)
4. Chimani, M., Jünger, M., Schulz, M.: Crossing minimization meets simultaneous drawing. In: IEEE Pacific Visualization Symposium 2008, pp. 33–40 (2008)
5. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10(1), 41–51 (1990)
6. Estrella-Balderrama, A., Gassner, E., Jünger, M., Percan, M., Schaefer, M., Schulz, M.: Simultaneous geometric graph embeddings. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 280–290. Springer, Heidelberg (2008)
7. Fowler, J.J., Jünger, M., Kobourov, S.G., Schulz, M.: Characterizations of restricted pairs of planar graphs allowing simultaneous embedding with fixed edges. Technical Report TR08-01, University of Arizona (2008), `ftp://ftp.cs.arizona.edu/reports/2008/TR08-01.pdf`
8. Frati, F.: Embedding graphs simultaneously with fixed edges. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 108–113. Springer, Heidelberg (2007)
9. Gassner, E., Jünger, M., Percan, M., Schaefer, M., Schulz, M.: Simultaneous graph embeddings with fixed edges. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 325–335. Springer, Heidelberg (2006)
10. Geyer, M., Kaufmann, M., Vrto, I.: Two trees which are self-intersecting when drawn simultaneously. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 201–210. Springer, Heidelberg (2006)
11. Hershberger, J., Suri, S.: An optimal algorithm for Euclidean shortest paths in the plane. SIAM Journal on Computing 28(6), 2215–2256 (1999)
12. Kuratowski, C.: Sur les problèmes des courbes gauches en Topologie. Fund. Math. 15, 271–283 (1930)
13. Pach, J., Wenger, R.: Embedding planar graphs at fixed vertex locations. In: Whitesides, S.H. (ed.) GD 1998. LNCS, vol. 1547, pp. 263–274. Springer, Heidelberg (1998)
14. Wagner, K.: Über eine Eigenschaft der ebenen Komplexe. Math. Ann. 114(1), 570–590 (1937)

A Lower Bound on the Area Requirements of Series-Parallel Graphs

Fabrizio Frati

Dipartimento di Informatica e Automazione, Università di Roma Tre
frati@dia.uniroma3.it

Abstract. We show that there exist series-parallel graphs requiring $\Omega(n \log n)$ area in any straight-line or poly-line grid drawing. Such a result is achieved by proving that, in any straight-line or poly-line drawing of $K_{2,n}$, one side of the bounding box has length $\Omega(n)$.

1 Introduction

A planar graph is a graph that can be drawn in the plane so that no two edges intersect, except, possibly, at common endpoints. Planar graphs are nicely characterized to be those graphs excluding K_5 and $K_{3,3}$ as minors [15]. Determining asymptotic bounds for the area requirements of straight-line and poly-line drawings of planar graphs is one of the classical topics in the Graph Drawing literature. Groundbreaking works of the end of the eighties have shown that every n-vertex planar graph admits a planar straight-line drawing in a $O(n) \times O(n)$ grid [5,7,12]. It turns out that such a bound is worst-case optimal, even for poly-line drawings [7,9]. Hence, it is natural to search for interesting subclasses of planar graphs admitting sub-quadratic area drawings.

Concerning *outerplanar graphs*, i.e., graphs excluding K_4 and $K_{2,3}$ as minors, Biedl [2] has shown how to construct poly-line drawings in $O(n \log n)$ area, and Di Battista and Frati [8] presented an algorithm for obtaining straight-line drawings in $O(n^{1.48})$ area. Concerning *trees*, i.e., graphs excluding K_3 as a minor, the *h-v drawing* algorithm in [6] allows to construct drawings in $O(n \log n)$ area. Both for outerplanar graphs and for trees, no super-linear area lower bound is known.

Another class of planar graphs that has been widely investigated in the Graph Theory and Graph Drawing literature is the one of *series-parallel graphs* (see, e.g., [14,10,1]). A series-parallel graph is a graph excluding K_4 as a minor. In [3] Biedl proved that every series-parallel graph admits a poly-line drawing in $O(n^{3/2})$ area. However, no sub-quadratic area upper bound is known in the case of straight-line drawings. In the same paper, Biedl proved a $\Omega(\frac{n \log n}{\log \log n})$ area lower bound for straight-line drawings of series-parallel graphs.

The $\Omega(\frac{n \log n}{\log \log n})$ area lower bound for straight-line drawings of series-parallel graphs is a direct consequence of the results in [4], where Biedl, Chan, and López-Ortiz, settling in the positive a conjecture of Felsner *et al.* [11], proved that no linear-area straight-line drawing of $K_{2,n}$ can achieve constant aspect ratio., i.e., constant ratio between the longest and the smallest side of the bounding box of the drawing. More precisely, Biedl, Chan, and López-Ortiz proved the following:

Theorem 1. (Biedl *et al.* [4]) *Every planar straight-line drawing of $K_{2,n}$ in a $W \times H$ grid with $W \geq H$ satisfies $W \log H \in \Omega(n)$.*

Corollary 1. (Biedl *et al.* [4]) *Every planar straight-line drawing of $K_{2,n}$ in a $W \times H$ grid satisfies* $\max\{W, H\} \in \Omega(n/\log n)$.

Biedl *et al.* ask whether the $\log H$ factor in Theorem 1 can be eliminated and whether the same lower bound holds even in the case of poly-line drawings.

In this paper we answer both the questions in the affirmative. Namely, we prove the following:

Theorem 2. *Every planar straight-line or poly-line drawing of $K_{2,n}$ in a $W \times H$ grid satisfies* $\max\{W, H\} \in \Omega(n)$.

As a main consequence of Theorem 2, we obtain a $\Omega(n \log n)$ lower bound on the area requirements of poly-line and straight-line drawings of series-parallel graphs. We remark that no super-linear area lower bound was previously known for poly-line drawings of series-parallel graphs and that $\Omega(\frac{n \log n}{\log \log n})$ was the best known area lower bound for straight-line drawings of series-parallel graphs [3].

Theorem 3. *There exist series-parallel graphs requiring $\Omega(n \log n)$ area in any straight-line or poly-line drawing.*

Proof. Consider any series-parallel graph S containing $K_{2,(n/2-2)}$ and a $(n/2)$-node complete ternary tree as subgraphs. A complete ternary tree that has height $h+1$ cannot be drawn on h parallel grid lines [11,13]. Since an n-node complete ternary tree has height $\log_3(2n+1)$, it follows that both sides of the drawing of a $(n/2)$-node complete ternary tree have length $\Omega(\log n)$. Hence, in any straight-line or poly-line drawing of S both sides have length $\Omega(\log n)$ and, by Theorem 2, one side has length $\Omega(n)$. The theorem follows. □

2 Preliminaries

A *grid drawing* of a graph is a mapping of each vertex to a distinct point of the plane with integer coordinates and of each edge to a Jordan curve between the endpoints of the edge. A *planar drawing* is such that no two edges intersect except, possibly, at common endpoints. In the following we always refer to planar grid drawings. A *straight-line* drawing is such that all edges are rectilinear segments. A *poly-line* drawing is such that the edges are sequences of rectilinear segments. In a poly-line drawing a *bend* is a point in which an edge changes its slope, i.e., a point common to two consecutive segments in the sequence of segments representing the edge. In a grid drawing bends have integer coordinates.

The *bounding box* of a drawing Γ is the smallest rectangle with sides parallel to the axes that covers Γ completely. The *height* (*width*) of Γ is the height (resp. width) of its bounding box. The *area* of Γ is the height of Γ times its width.

A poly-line drawing of the complete bipartite graph $K_{2,n}$ can be thought as a drawing of n paths that start and end at the same two vertices, in the following denoted by a and b, and that do not share any other vertex. In the following we will refer to such paths as to the *paths of* $K_{2,n}$.

3 Lemmata on the Geometry of $K_{2,n}$

In this section we show some lemmata that will be used to prove Theorem 2.

Lemma 1. *Consider any poly-line drawing of $K_{2,n}$, any path π of $K_{2,n}$, and any vector $\boldsymbol{v} = (v_1, v_2)$. There exists a grid point $p \in \pi$ such that $\boldsymbol{v} \cdot p \geq \boldsymbol{v} \cdot p'$, for any point $p' \in \pi$.*

Proof. If $\boldsymbol{v} \cdot a \geq \boldsymbol{v} \cdot p'$ or $\boldsymbol{v} \cdot b \geq \boldsymbol{v} \cdot p'$, for every point $p' \in \pi$, the lemma follows. Otherwise, consider the part π' of π starting at a and ending at the first point p in which $\boldsymbol{v} \cdot p \geq \boldsymbol{v} \cdot p'$, for every point $p' \in \pi$ (see Fig. 1.a). Since each point p' of π' has $\boldsymbol{v} \cdot p' < \boldsymbol{v} \cdot p$, then there exists a small disk D centered at p such that the part of π' enclosed in D is increasing in the direction determined by $\boldsymbol{v}$, when π' is oriented from a to p. On the other hand π, when oriented from a to b, cannot be increasing immediately after p in the direction determined by $\boldsymbol{v}$, otherwise there would exist a point p'' such that $\boldsymbol{v} \cdot p'' > \boldsymbol{v} \cdot p$. It follows that π changes slope at p and, by definition of poly-line grid drawing, p is a grid point. $\square$

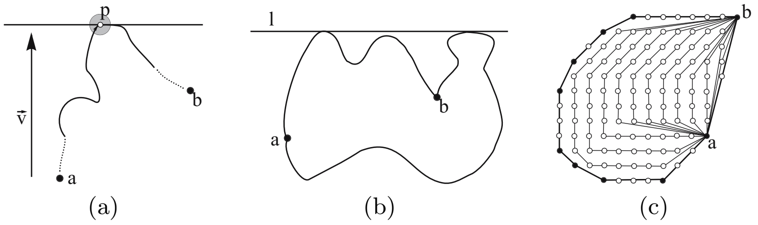


Fig. 1. (a) Illustration for the proof of Lemma 1. Disk D is the small shaded region. (b) Illustration for the proof of Lemma 2. (c) Drawing the maximum number of paths in a convex polygon with vertices (drawn as black circles) having integer coordinates.

Lemma 2. *Consider any drawing of $K_{2,n}$. Let l be any line that does not intersect or contain the open segment (a, b). There exist no three paths π_1, π_2, and π_3 of $K_{2,n}$ such that: (i) π_1, π_2, and π_3 do not intersect each other; (ii) π_1, π_2, and π_3 are entirely contained in the closed half-plane delimited by l and containing a and b; (iii) each of π_1, π_2, and π_3 touches l at least once.*

Proof. Suppose, for a contradiction, that three paths π_1, π_2, and π_3 of $K_{2,n}$ with the above properties exist. Paths π_1 and π_2 form a cycle $\mathcal{C}$. Line l is external to $\mathcal{C}$ and separates a from b in the exterior of $\mathcal{C}$ (see Fig. 1.b). Consider any path π_3 between a and b. If π_3 is internal to $\mathcal{C}$, then it can not touch l unless it intersects $\mathcal{C}$. If π_3 is external to $\mathcal{C}$, then it intersects l. If π_3 is part internal and part external to $\mathcal{C}$, then it intersects $\mathcal{C}$. In any case we have a contradiction. $\square$

Let P be any convex polygon in the plane with vertices having integer coordinates. Let G be the set of grid points in the interior or on the border of P. Let a and b be two distinct vertices of P. Let π_1^* and π_2^* be the two paths that connect a and b and that compose P. At least one out of π_1^* and π_2^*, say π_1^*, is different from segment $\overline{ab}$. Let M be the maximum number of paths connecting a and b that can be drawn as non-crossing polygonal paths inside or on the border of P.

Lemma 3. *There exists a drawing of M non-crossing polygonal paths connecting a and b such that each path is completely contained inside or on the border of P and one of such paths is drawn as π_1^*.*

Proof. Consider any drawing Γ of M non-crossing polygonal paths connecting a and b and contained inside or on the border of P. If a path of Γ is drawn as π_1^*, there is nothing to prove. Otherwise, observe that no two distinct paths π_i and π_j can pass through points of π_1^*, otherwise π_i and π_j would cross. Hence, Γ has at most one path π passing through points of π_1^*. Remove π from Γ, if π exists, and draw a path in Γ as π_1^*. Since no path different from π passes through a point of π_1^*, the resulting drawing is planar, hence proving the lemma. □

Lemma 4. *There exists a drawing of M non-crossing polygonal paths connecting a and b such that each path is completely contained inside or on the border of P and such that one of the paths is represented by segment $\overline{ab}$.*

Proof. We prove the claim by induction on M. If $M = 1$, then drawing a path as segment $\overline{ab}$ proves the claim. Suppose $M \geq 2$. By Lemma 3, there exists a drawing Γ of M non-crossing polygonal paths connecting a and b such that each path is inside or on the border of P and one of such paths, say π, is drawn as π_1^*. Remove π from Γ and all the grid points π passes through, except for a and b, from G. Consider the convex closed polygon P' that is the convex hull of the resulting grid point-set G'. The vertices of P' have integer coordinates. Further, P' is such that $M - 1$ non-crossing polygonal paths connecting a and b can be drawn with each path inside or on the border of P'. In fact Γ is a drawing having such a property. Hence, the inductive hypothesis applies and $M - 1$ paths can be drawn so that each path is inside or on the border of P' and so that one of the paths is represented by segment $\overline{ab}$. Considering the drawing of such $M - 1$ paths together with the drawing of π as π_1^* proves the lemma. □

Now assume that a and b are consecutive vertices of P (see Fig. 1.c). Let G be the set of grid points in the interior or on the border of P. As before, let π_1^* and π_2^* be the two paths that connect a and b and that compose P, where π_1^* is different from segment $\overline{ab}$. Let also M be the maximum number of paths connecting a and b that can be drawn as non-crossing polygonal paths completely contained inside or on the border of P. We iteratively draw paths $\pi_1, \pi_2, \cdots, \pi_N$ connecting a and b inside or on the border of P as follows. Path π_i is drawn when the current convex grid polygon is P_i containing in its interior or on its border a set G_i of grid points. At the first step $P_1 = P$ and $G_1 = G$. If P_i does not coincide with segment $\overline{ab}$, draw path π_i as the polygonal line that connects a and b, that lies on P_i, and that is different from segment $\overline{ab}$. Remove the grid points that lie on P_i, except for a and b, from G_i, obtaining a new set of grid points G_{i+1}. Then, P_{i+1} is the convex hull of G_{i+1}. If P_i coincides with segment $\overline{ab}$, draw the path π_i as segment $\overline{ab}$. We observe the following:

Lemma 5. *Paths $\pi_1, \pi_2, \cdots, \pi_N$ are non-crossing polygonal lines that connect a and b and that completely lie inside or on the border of P. Further, $N = M$.*

Proof. The first part of the statement is trivial. We prove that $N = M$ by induction on M. If $M = 2$, then the claim trivially holds, since π_1 is drawn as π_1^* and π_2 as $\overline{ab}$. Suppose $M \geq 3$. By Lemma 3, there exists a drawing Γ of M non-crossing polygonal paths connecting a and b such that each path is completely contained inside or on the border of P and one of such paths, say π_1, is drawn as π_1^*. Remove π_1 from Γ and all the grid points π_1 passes through from G. Consider the convex closed polygon P' that is the convex hull of the resulting grid point-set G'. Clearly, the vertices of P' have integer coordinates. Further, P' is such that $M-1$ non-crossing polygonal paths connecting a and b can be drawn such that each path is completely contained inside or on the border of P'. In fact Γ is a drawing having such a property. Hence, the inductive hypothesis applies and the drawing algorithm described before the statement of the lemma draws $M-1$ non-crossing polygonal paths inside or on the border of P'. Considering such paths together with the drawing of π_1 as π_1^* proves the lemma. □

4 Proof of Theorem 2

By definition, a straight-line drawing is also a poly-line drawing. Hence, it suffices to prove Theorem 2 for poly-line drawings. Consider any poly-line drawing of $K_{2,n}$. Let R be the minimum closed axis-parallel rectangle enclosing a and b (see Fig. 2.a). Let $l_{a,b}$ be the line through a and b. Suppose, w.l.o.g., that $y(a) \leq y(b)$. Suppose also that the slope of $l_{a,b}$ is greater or equal than 0, the case in which the slope of $l_{a,b}$ is less than 0 being analogous. Let c and d be the upper left corner and the lower right corner of R, respectively. Let h_a and v_a (h_b and v_b) be the horizontal and vertical lines through a (resp. through b), respectively. For any line l, denote by $H^+(l)$ (resp. by $H^-(l)$) the closed half-plane delimited by l and containing the normal vector of l increasing in the y-direction (resp. decreasing in the y-direction). If l is a vertical line, then $H^+(l)$ (resp. $H^-(l)$) denotes the closed half-plane delimited by l and containing the normal vector of l increasing in the x-direction (resp. decreasing in the x-direction). Let d_1 and d_2 be the horizontal and vertical distance between a and b, respectively. The width W and the height H of the drawing are such that $W \geq d_1$ and $H \geq d_2$.

Consider the half-plane $H^+(h_b)$. By Lemma 1 with $\boldsymbol{v} = (0, 1)$, for each path π intersecting $H^+(h_b)$, there exists a grid point $p \in \pi$ whose y-coordinate is maximum among the points of π. Clearly, p belongs to $H^+(h_b)$. Hence, p belongs to an horizontal grid line l that does not intersect or contain the open segment (a, b). By Lemma 2, at most two paths of $K_{2,n}$ have their points with greatest y-coordinate belonging to l. It follows that, if a linear number of paths of $K_{2,n}$ intersects $H^+(h_b)$, then their points with greatest y-coordinate belong to a linear number of distinct horizontal grid lines and hence $H \in \Omega(n)$.

Similar arguments show that, if a linear number of edges intersect $H^-(h_a)$, $H^+(v_b)$, or $H^-(v_a)$, then $H \in \Omega(n)$, $W \in \Omega(n)$, or $W \in \Omega(n)$, respectively. If

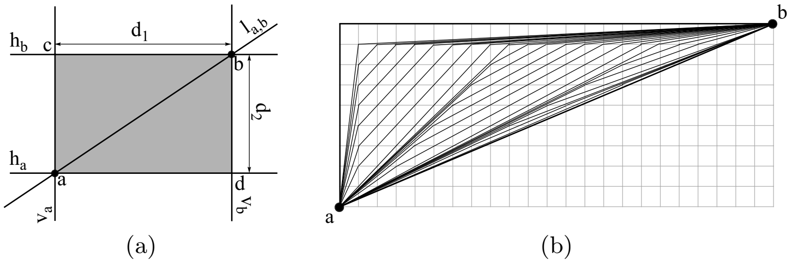


Fig. 2. (a) Illustration of the notation for the proof of Theorem 2. (b) Paths $\pi_1, \pi_2, \cdots, \pi_{M_1}$ in Π.

there exists no linear number of edges intersecting $H^+(h_b)$, $H^-(h_a)$, $H^+(v_b)$, or $H^-(v_a)$, then a linear number of edges is completely inside R. We show that this implies that $\max\{d_1, d_2\} \in \Omega(n)$, and hence that $\max\{W, H\} \in \Omega(n)$.

Let M be the maximum number of paths of $K_{2,n}$ that can be drawn inside R. By Lemma 4, there exists a drawing of M paths connecting a and b, and completely lying inside R, such that one of the paths is drawn as segment $\overline{ab}$. Since $M \in \Omega(n)$, then either a linear number of paths of $K_{2,n}$ is contained in the triangle T_1 having a, b, and c as vertices, or a linear number of paths of $K_{2,n}$ is contained in the triangle T_2 having a, b, and d as vertices. Suppose that a linear number of paths is contained into T_1, the other case being symmetric.

Let $M_1 \in \Omega(n)$ be the maximum number of paths of $K_{2,n}$ that can be drawn inside T_1, and let G_1 be the set of grid points inside or on the border of T_1. By Lemma 5, a sequence of M_1 non-crossing paths $\Pi = (\pi_1, \pi_2, \cdots, \pi_{M_1})$ connecting a and b and completely inside or on the border of T_1 can be drawn by repeating the following two operations, for $1 \leq i < M_1$: (1) consider the current convex grid polygon P_i (when $i = 1$ then $P_1 = T_1$); let G_i be the set of grid points inside or on the border of P_i; draw path π_i as the part of P_i that connects a and b, and that is different from segment $\overline{ab}$; (2) delete from G_i the grid points π_i passes through, obtaining a set of grid points G_{i+1}. Closed convex polygon P_{i+1} is the convex hull of G_{i+1}. Path π_{M_1} is drawn as segment $\overline{ab}$. See Fig. 2.b.

In order to prove that $M_1 \in \Omega(n)$ implies $\max\{d_1, d_2\} \in \Omega(n)$, we study paths $\pi_1, \pi_2, \cdots, \pi_{M_1}$. Such a study reveals interesting properties of the grid that we skecth here and detail in the following. First, we observe that each path in Π is composed by two or three segments, i.e., each path has one or two bends. A sequence of paths that are consecutive in Π and that are each composed by three segments is such that all the "second segments" of the paths have the same slope. We show that, in a sequence of paths such that the second segments of the paths have the same slope, all the bends lie on two lines, having slopes one greater and one less than the slope of segment $\overline{ab}$. The more sequences of three-segments-paths that are consecutive in Π are considered, the more the slope of the first, of the second, and of the third segment of the paths approaches to the slope of segment $\overline{ab}$. Consider a sequence of paths such that the second segments of the paths have the same slope $\frac{s_1}{s_2}$. Then, the bends of such paths lie on two lines with slopes, say, $\frac{s_3}{s_4}$ and $\frac{s_5}{s_6}$, such that $s_3 + s_5 = s_1$ and $s_4 + s_6 = s_2$.

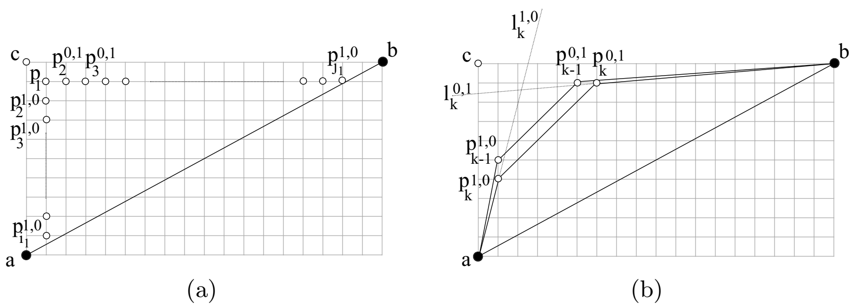


Fig. 3. (a) Sequences $S_{1,0}$ and $S_{0,1}$. (b) Path π_{k+1} is a polygonal line composed of segments $\overline{ap_k^{1,0}}$, $\overline{p_k^{1,0}p_k^{0,1}}$, $\overline{p_k^{0,1}b}$, for $k = 1, 2, \cdots, \min\{i_1, j_1\}$.

Further, the next sequence of paths whose second segments have the same slope is such that the bends of such paths lie on two lines with slopes $\frac{s_1}{s_2}$ and $\frac{s_3}{s_4}$ (or $\frac{s_1}{s_2}$ and $\frac{s_5}{s_6}$), and the second segments of such paths have slope $\frac{s_1+s_3}{s_2+s_4}$ (resp. $\frac{s_1+s_5}{s_2+s_6}$). We subdivide Π into disjoint subsequences $\Pi_1, \Pi_2, \cdots, \Pi_f$ and we argue that Π_1 has at most $\max\{d_1, d_2\}$ paths and that Π_i has at most $\max\{d_1, d_2\}/2^{i-2}$ paths, for $2 \leq i \leq f$; such bounds lead to conclude that, as long as $M_1 \in \Omega(n)$, $\max\{d_1, d_2\} \in \Omega(n)$.

Path π_1 is composed of segments $\overline{ac}$ and $\overline{cb}$. Let p_1 be the point one vertical unit below and one horizontal unit to the right of c. Consider the following two sequences of grid points (see Fig. 3.a). Sequence $S_{1,0}$ is composed of points:

$$\begin{aligned}
p_1^{1,0} &= p_1,\\
p_2^{1,0} &= (x(p_1), y(p_1) - 1),\\
p_3^{1,0} &= (x(p_1), y(p_1) - 2),\\
&\cdots,\\
p_{i_1}^{1,0} &= (x(p_1), y(p_1) - (i_1 - 1)),
\end{aligned}$$

where i_1 is the maximum such that point $(x(p_1), y(p_1) - (i_1 - 1))$ is contained inside T_1. Sequence $S_{0,1}$ is composed of points:

$$\begin{aligned}
p_1^{0,1} &= p_1,\\
p_2^{0,1} &= (x(p_1) + 1, y(p_1)),\\
p_3^{0,1} &= (x(p_1) + 2, y(p_1)),\\
&\cdots,\\
p_{j_1}^{0,1} &= (x(p_1) + (j_1 - 1), y(p_1)),
\end{aligned}$$

where j_1 is the maximum such that point $(x(p_1) + (j_1 - 1), y(p_1))$ is contained inside T_1. Notice that the points of $S_{1,0}$ lie on a line with slope $\frac{1}{0} = \infty$ and the points of $S_{0,1}$ lie on a line with slope $\frac{0}{1} = 0$. In the following, we show that a subsequence Π_1 of Π, starting at π_2 and composed of paths consecutive in Π, "consumes" the points in $S_{1,0}$ and in $S_{0,1}$, i.e., each point in $S_{1,0}$ and each point in $S_{0,1}$ is traversed by a path in Π_1; further, each path in Π_1 passes through a distinct point of the one of $S_{1,0}$ and $S_{0,1}$ that has the greater number of points.

We claim that path π_{k+1} is a polygonal line composed of segments $\overline{ap_k^{1,0}}$, $\overline{p_k^{1,0}p_k^{0,1}}$, $\overline{p_k^{0,1}b}$, for $k = 1, 2, \cdots, \min\{i_1, j_1\}$ (notice that $p_1^{1,0} = p_1^{0,1} = p_1$, hence π_2 is composed by only two segments). The claim directly implies that the second segment of path π_{k+1}, for $k = 2, 3, \cdots, \min\{i_1, j_1\}$, has slope $\frac{1}{1} = 1$. We prove the claim by induction on k. Let $l_k^{1,0}$, $l_k^{1,1}$, and $l_k^{0,1}$, be the lines through a and $p_k^{1,0}$, through $p_k^{1,0}$ and $p_k^{0,1}$, and through $p_k^{0,1}$ and b, respectively.

In the base case $k = 1$. Observe that $H^+(l_1^{1,0})$ and $H^+(l_1^{0,1})$ do not contain grid points that are inside or on the border of T_1, and that do not belong to π_1, except for p_1. Hence, π_2 is composed by segments $\overline{ap_1}$ and $\overline{p_1b}$, proving the claim in the base case. Suppose that the claim holds for paths $\pi_2, \pi_3, \cdots, \pi_k$. Then, π_k is a polygonal line composed of segments $\overline{ap_{k-1}^{1,0}}$, $\overline{p_{k-1}^{1,0}p_{k-1}^{0,1}}$, $\overline{p_{k-1}^{0,1}b}$. We prove that the claim holds for path π_{k+1} (see Fig. 3.b). It is easy to see that $H^+(l_k^{1,0})$ and $H^+(l_k^{0,1})$ do not contain grid points internal to polygon $\pi_k \cup \overline{ab}$, except for $p_k^{1,0}$ and for $p_k^{0,1}$, respectively. Further, no grid point is contained inside quadrilateral $(p_{k-1}^{0,1}, p_{k-1}^{1,0}, p_k^{1,0}, p_k^{0,1})$. Namely, any grid point of the plane lies on a line with slope $\frac{1}{1}$, and the line that has slope $\frac{1}{1}$, that contains grid points internal to $\pi_k \cup \overline{ab}$, and that is closer to $l_{k-1}^{1,1}$ is line $l_k^{1,1}$ through $p_k^{1,0}$ and $p_k^{0,1}$. Hence, path π_{k+1} is composed by segments $\overline{ap_k^{1,0}}$, $\overline{p_k^{1,0}p_k^{0,1}}$, $\overline{p_k^{0,1}b}$, proving the claim.

Three cases have to be considered, namely $i_1 = j_1$, $i_1 < j_1$, and $i_1 > j_1$. If $i_1 = j_1$, we claim that there is no grid point internal to polygon $\pi_{i_1+1} \cup \overline{ab}$. Observe that, since a is one unit to the left of the vertical line on which the points of $S^{1,0}$ lie, and since b is one unit above the horizontal line on which the points of $S^{0,1}$ lie, then, if there is any grid point internal to polygon $\pi_{i_1+1} \cup \overline{ab}$, either point $p_{i_1+1}^{1,0} = (x(p_1), y(p_1) - i_1)$ or $p_{j_1+1}^{0,1} = (x(p_1) + j_1, y(p_1))$ is internal to $\pi_{i_1+1} \cup \overline{ab}$ (see Fig. 4.a). However, by the maximality of i_1 and j_1, both $p_{i_1+1}^{1,0}$ and $p_{j_1+1}^{0,1}$ are outside or on the border of T_1, and hence they are not internal to $\pi_{i_1+1} \cup \overline{ab}$. Since there is no grid point internal to polygon $\pi_{i_1+1} \cup \overline{ab}$, then the only path of $K_{2,n}$ after $\pi_{i_1+1} = \pi_{j_1+1}$ in Π is segment $\overline{ab}$.

Now consider the case in which $i_1 < j_1$ (the case in which $i_1 > j_1$ is analogous). Sequence $S_{1,0}$ is "over", i.e., there is a path π_i passing through each point of $S_{1,0}$. Let $S_{1,1}$ be the sequence defined as follows (see Fig. 4.b):

$$\begin{aligned}
p_1^{1,1} &= p_{i_1+1}^{0,1},\\
p_2^{1,1} &= (x(p_{i_1+1}^{0,1}) - 1, y(p_{i_1+1}^{0,1}) - 1),\\
p_3^{1,1} &= (x(p_{i_1+1}^{0,1}) - 2, y(p_{i_1+1}^{0,1}) - 2),\\
&\cdots,\\
p_{i_2}^{1,1} &= (x(p_{i_1+1}^{0,1}) - (i_2 - 1), y(p_{i_1+1}^{0,1}) - (i_2 - 1)),
\end{aligned}$$

where i_2 is the maximum such that point $((x(p_{i_1+1}^{0,1}) - (i_2 - 1), y(p_{i_1+1}^{0,1}) - (i_2 - 1))$ is contained inside T_1. Sequence $S_{1,1}$ "replaces" sequence $S_{1,0}$, namely path

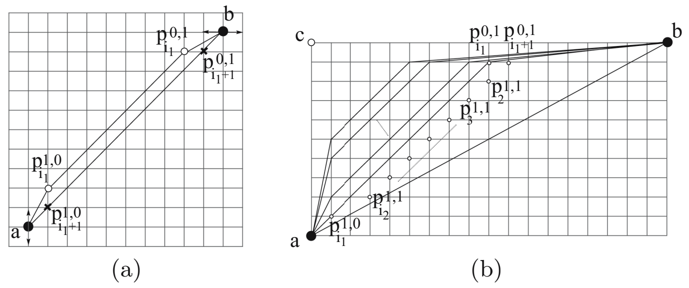


Fig. 4. (a) When $i_1 = j_1$, no grid point is internal to $\pi_{i_1+1} \cup \overline{ab}$. (b) Sequence $S_{1,1}$.

π_{i_1+k+1}, with $1 \le k \le \min\{i_2, j_1 - i_1\}$, is a polygonal line composed of segments $\overline{ap^{1,1}_k}$, $\overline{p^{1,1}_k p^{0,1}_{i_1+k}}$, $\overline{p^{0,1}_{i_1+k}b}$ (the proof of such a claim is analogous to the proof that π_{k+1} is composed by segments $\overline{ap^{1,0}_k}$, $\overline{p^{1,0}_k p^{0,1}_k}$, $\overline{p^{0,1}_k b}$, for $k = 1, 2, \cdots, \min\{i_1, j_1\}$). Notice that the bends of such paths lie on two lines with slope $\frac{1}{1} = 1$ and $\frac{0}{1} = 0$, while the second segments of such paths lie on lines with slope $\frac{1+0}{1+1} = \frac{1}{2}$.

Again, three cases have to be considered. In the first case, we have $i_2 = j_1 - i_1$. Hence, path π_{j_1+1} passes through the last point of $S_{1,1}$ and through the last point of $S_{0,1}$. Then no grid point lies inside polygon $\pi_{j_1+1} \cup \overline{ab}$ (the proof of such a claim is analogous to the one that there is no grid point internal to polygon $\pi_{i_1+1} \cup \overline{ab}$ when $i_1 = j_1$), and hence the only path of $K_{2,n}$ after $\pi_{i_1+i_2+1} = \pi_{j_1+1}$ in Π is segment $\overline{ab}$. Otherwise, one of the two sequences $S_{1,1}$ and $S_{1,0}$ ends before the other. Suppose that sequence $S_{1,1}$ ends before $S_{0,1}$. Then $S_{1,1}$ is replaced by a sequence $S_{1,2}$ of points lying on a line with slope $\frac{1}{2}$. Namely, such points have coordinates:

$$p^{1,2}_k = \left(x(p^{0,1}_{i_1+i_2+1}) - 2(k-1), x(p^{0,1}_{i_1+i_2+1}) - (k-1)\right),$$

for $1 \le k \le i_3$, where i_3 is the maximum index such that point $(x(p^{0,1}_{i_1+i_2+1}) - 2(i_3 - 1), x(p^{0,1}_{i_1+i_2+1}) - (i_3 - 1))$ is internal to T_1. Each path $\pi_{i_1+i_2+k+1}$, with $1 \le k \le \min\{i_3, j_1 - i_1 - i_2\}$, passes through point $p^{1,2}_k$ and through point $p^{0,1}_{i_1+i_2+k}$, and the second segment of each of such paths has slope $\frac{1+0}{2+1} = \frac{1}{3}$.

The above argument iterates while sequence $S_{0,1}$ is not over, i.e., while there are points of $S_{0,1}$ that are not traversed by paths in Π. From the above discussion, all paths that come after π_1 in Π pass through distinct points of $S_{0,1}$, while sequence $S_{0,1}$ is not over. Hence, supposing $i_1 \le j_1$ (the case in which $j_1 \le i_1$ being analogous), $\Pi_1 = (\pi_2, \pi_3, \cdots, \pi_{j_1+1})$ is the desired subsequence of Π passing through all points of $S_{1,0}$ and through all points of $S_{0,1}$. Further, there exists an index $l \ge 1$ such that: (1) every point in $S_{1,i}$ is traversed by a path in Π_1, for $0 \le i < l$, and (2) some points of $S_{1,l}$ are eventually traversed by a path in Π_1.

After drawing path π_{j_1+1} (that passes through the last point of $S^{0,1}$), either sequence $S_{1,l}$ is simultaneously over, i.e., $j_1 = i_1 + i_2 + \cdots + i_l$, or $S_{1,l}$ still contains points internal to T_1 and not traversed by any path in Π_1. In the former case we

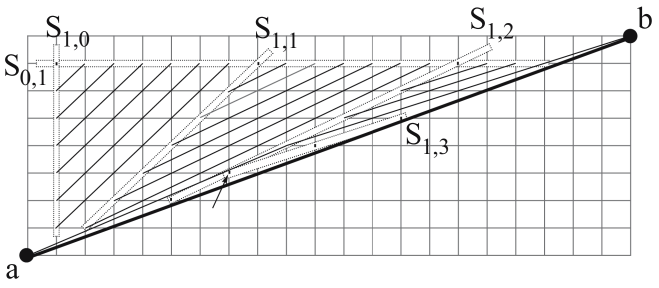


Fig. 5. The case in which sequence $S_{1,l}$ still contains at least one point after drawing path π_{j_1+1}. In this example $l = 2$. To improve the readability of the drawing, only the second segments of the paths in Π_1 are drawn, but for path π_{j_1+1}, which is entirely drawn. The arrow shows the first point of $S_{1,2}$ that is not traversed by a path in Π_1.

have that no grid point is internal to polygon $\pi_{j_1+1} \cup \{\overline{ab}\}$ and hence the only path of $K_{2,n}$ after π_{j_1+1} in Π is segment $\overline{ab}$. In the latter case, more than one path could exist in Π after π_{j_1+1}. Namely, sequence $S_{0,1}$ is now replaced by a sequence $S_{1,l+1}$, whose points lie on a line with slope $\frac{0+1}{1+l} = \frac{1}{l+1}$ passing through the first point of $S_{1,l}$ that is not traversed by a path in Π_1 (see Fig. 5).

The whole argument is now repeated again. Namely, we search for a subsequence Π_2 of Π such that Π_2 "consumes" the points in $S_{1,l}$ and the points in $S_{1,l+1}$, i.e., such that each point in $S_{1,l}$ that is not traversed by a path in Π_1 is traversed by a path in Π_2 and such that each point in $S_{1,l+1}$ is traversed by a path in Π_2; further, each path in Π_2 passes through a point of the one out of $S_{1,l}$ and $S_{1,l+1}$ that has the greater number of points. Again, Π_2 is generally found in several steps, where at each step two sequences S_{y_1,x_1} and S_{y_2,x_2} of grid points are considered (at the first step such sequences are $S_{1,l}$ and $S_{1,l+1}$, where the points of $S_{1,l}$ that are traversed by paths in Π_1 are not considered). At each step, the smallest between S_{y_1,x_1} and S_{y_2,x_2} is consumed by the paths in a subsequence of Π_2 and is replaced by a sequence of points lying on a line with slope $\frac{y_1+y_2}{x_1+x_2}$, hence starting a new step. After a certain number of steps, all points in $S_{1,l}$ and in $S_{1,l+1}$ are traversed by a path in Π_2. When the last path of Π_2 is drawn, either the currently considered sequences $S_{y_1^*,x_1^*}$ and $S_{y_2^*,x_2^*}$ are simultaneously over, or there are still points, not traversed by any path in Π_2, in the one out of $S_{y_1^*,x_1^*}$ and $S_{y_2^*,x_2^*}$ that is different from both $S_{1,l}$ and in $S_{1,l+1}$. In the former case, no grid point is inside the polygon composed of the last drawn path and of $\overline{ab}$, and hence the only path of $K_{2,n}$ after the last path of Π_2 in Π is segment $\overline{ab}$. In the latter case, the one between $S_{y_1^*,x_1^*}$ and $S_{y_2^*,x_2^*}$ that is over, say $S_{y_2^*,x_2^*}$, is replaced by a sequence $S_{y_1^*+y_2^*,x_1^*+x_2^*}$, whose grid points lie on a line with slope $\frac{y_1^*+y_2^*}{x_1^*+x_2^*}$, and the whole argument is repeated again, searching for a subsequence Π_3 of Π such that Π_3 consumes the points in $S_{y_1^*,x_1^*}$ and the points in $S_{y_1^*+y_2^*,x_1^*+x_2^*}$. Clearly, there exists an index f such that $\Pi = \{\pi_1\} \cup \Pi_1 \cup \Pi_2 \cup \cdots \Pi_f \cup \{\overline{ab}\}$.

We now compute how many paths exist in Π, as a function of d_1 and d_2. Denote by $S_{y_1^i,x_1^i}$ and by $S_{y_2^i,x_2^i}$ the sequences of grid points that are consumed by Π_i, where the grid points in $S_{y_1^i,x_1^i}$ lie on a line with slope y_1^i/x_1^i and the grid points in $S_{y_2^i,x_2^i}$ lie on a line with slope y_2^i/x_2^i. Notice that, with the above notation, $S_{y_1^1,x_1^1} = S_{1,0}$ and $S_{y_2^1,x_2^1} = S_{0,1}$, and, if $i_1 \leq j_1$, $S_{y_1^2,x_1^2} = S_{1,l}$, and $S_{y_2^2,x_2^2} = S_{1,l+1}$. It is easy to prove that $x_1^i, y_1^i, x_2^i, y_2^i \geq 2^{i-2}$, for $i \geq 2$. Notice that we already observed that such a claim holds when $i = 2$. From the above discussion, we have that y_1^i is obtained as the sum of the numerators y_a^{i-1} and y_b^{i-1} of the slopes of two lines containing grid points traversed by paths in Π_{i-1}. Inductively, $y_a^{i-1} + y_b^{i-1} \geq y_1^{i-1} + y_2^{i-1} \geq 2^{i-3} + 2^{i-3} \geq 2^{i-2}$. Analogously $y_2^i, x_1^i, x_2^i \geq 2^{i-2}$.

The number of paths in Π_i is the number of grid points in the one out of $S_{y_1^i,x_1^i}$ and $S_{y_2^i,x_2^i}$ with the greater number of points. When $i = 1$, each of $S_{1,0}$ and $S_{0,1}$ has at most $\max\{d_1, d_2\}$ grid points. Further, for $i \geq 2$, $S_{y_1^i,x_1^i}$ and $S_{y_2^i,x_2^i}$ lie on lines with slopes whose numerators and denominators are greater or equal than 2^{i-2}. Hence, each of such sequences has at most $\frac{\max\{d_1,d_2\}}{2^{i-2}} + 1$ grid points. Hence, the total number of paths in Π is at most

$$\underbrace{1}_{\pi_1} + \underbrace{\max\{d_1,d_2\}}_{\text{paths in } \Pi_1} + \underbrace{\sum_{i=2}^{f}\left(\frac{\max\{d_1,d_2\}}{2^{i-2}} + 1\right)}_{\text{paths in } \Pi_i, \text{ for } 2 \leq i \leq f} + \underbrace{1}_{\overline{ab}} \leq$$

$$\max\{d_1,d_2\} + 2\max\{d_1,d_2\} + \log_2(\max\{d_1,d_2\}) + 2 < 4\max\{d_1,d_2\} + 2.$$

Since the number of paths in Π is $\Omega(n)$, then $\max\{d_1, d_2\} \in \Omega(n)$ and hence $\max\{W, H\} \in \Omega(n)$. Theorem 2 follows.

Table 1. Summary of the best known area bounds for straight-line and poly-line drawings of planar graphs and their subclasses

	Straight-line				***Poly-line***			
Graph Class	***UB.***	***Ref.***	***LB.***	***Ref.***	***UB.***	***Ref.***	***LB.***	***Ref.***
Planar Graphs	$O(n^2)$	[7,12]	$\Omega(n^2)$	[9,7]	$O(n^2)$	[7,12]	$\Omega(n^2)$	[9,7]
Series-Parallel Graphs	$O(n^2)$	[7,12]	$\Omega(n \log n)$	*this paper*	$O(n^{3/2})$	[3]	$\Omega(n \log n)$	*this paper*
Outrplanar Graphs	$O(n^{1.48})$	[8]	$\Omega(n)$	*trivial*	$O(n \log n)$	[2]	$\Omega(n)$	*trivial*
Trees	$O(n \log n)$	[6]	$\Omega(n)$	*trivial*	$O(n \log n)$	[6]	$\Omega(n)$	*trivial*

5 Conclusions and Open Problems

In this paper we have shown that there exist series-parallel graphs requiring $\Omega(n \log n)$ area in any straight-line or poly-line drawing. As far as we know the best upper bound for the area requirements of poly-line drawings of series-parallel graphs is $O(n^{3/2})$ [3], while no sub-quadratic area upper bound is known in the case of straight-line drawings. Hence, in both cases, the gap between the upper and the lower bound is large.

We remark that, for outerplanar graphs and trees, no super-linear area lower bound is known, hence the determination of the area requirements for straight-line and poly-line drawings of such graph classes still requires further research.

Acknowledgments

Thanks to Giuseppe Di Battista for useful discussions. Work partially supported by MUR under Project MAINSTREAM Algorithms for Massive Information Structures and Data Streams.

References

1. Bertolazzi, P., Cohen, R.F., Di Battista, G., Tamassia, R., Tollis, I.G.: How to draw a series-parallel digraph. Int. J. Comp. Geom. Appl. 4(4), 385–402 (1994)
2. Biedl, T.C.: Drawing outer-planar graphs in $O(n \log n)$ area. In: Goodrich, M.T., Kobourov, S.G. (eds.) GD 2002. LNCS, vol. 2528, pp. 54–65. Springer, Heidelberg (2002)
3. Biedl, T.C.: Small poly-line drawings of series-parallel graphs. Tech. Report CS-2007-23, School of Computer Science, University of Waterloo, Canada (2005)
4. Biedl, T.C., Chan, T.M., López-Ortiz, A.: Drawing $K_{2,n}$: A lower bound. Inf. Process. Lett. 85(6), 303–305 (2003)
5. Chrobak, M., Nakano, S.: Minimum-width grid drawings of plane graphs. Comput. Geom. 11(1), 29–54 (1998)
6. Crescenzi, P., Di Battista, G., Piperno, A.: A note on optimal area algorithms for upward drawings of binary trees. Comput. Geom. 2, 187–200 (1992)
7. de Fraysseix, H., Pach, J., Pollack, R.: How to draw a planar graph on a grid. Combinatorica 10(1), 41–51 (1990)
8. Di Battista, G., Frati, F.: Small area drawings of outerplanar graphs. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 89–100. Springer, Heidelberg (2006)
9. Dolev, D., Leighton, T., Trickey, H.: Planar embeddings of planar graphs. Advances in Computing Research 2, 147–161 (1984)
10. Eppstein, D.: Parallel recognition of series-parallel graphs. Inf. Comput. 98(1), 41–55 (1992)
11. Felsner, S., Liotta, G., Wismath, S.K.: Straight-line drawings on restricted integer grids in two and three dimensions. J. Graph Algorithms Appl. 7(4), 363–398 (2003)
12. Schnyder, W.: Embedding planar graphs on the grid. In: Proc. 1st ACM-SIAM Sympos. Discr. Alg., pp. 138–148 (1990)
13. Suderman, M.: Pathwidth and layered drawings of trees. Int. J. Comp. Geom. Appl. 14(3), 203–225 (2004)
14. Valdes, J., Tarjan, R.E., Lawler, E.L.: The recognition of series parallel digraphs. SIAM J. Comput. 11(2), 298–313 (1982)
15. Wagner, K.: Uber eine Eigenschaft der ebenen Komplexe. Math. Ann. 114, 570–590 (1937)

On Independent Sets and Bicliques in Graphs*

Serge Gaspers[1], Dieter Kratsch[2], and Mathieu Liedloff[2]

[1] Department of Informatics, University of Bergen, N-5020 Bergen, Norway
serge@ii.uib.no
[2] Laboratoire d'Informatique Théorique et Appliquée,
Université Paul Verlaine - Metz, 57045 Metz Cedex 01, France
{kratsch,liedloff}@univ-metz.fr

Abstract. Bicliques of graphs have been studied extensively, partially motivated by the large number of applications. One of the main algorithmic interests is in designing algorithms to enumerate all maximal bicliques of a (bipartite) graph. Polynomial-time reductions have been used explicitly or implicitly to design polynomial delay algorithms to enumerate all maximal bicliques.

Based on polynomial-time Turing reductions, various algorithmic problems on (maximal) bicliques can be studied by considering the related problem for (maximal) independent sets. In this line of research, we improve Prisner's upper bound on the number of maximal bicliques [Combinatorica, 2000] and show that the maximum number of maximal bicliques in a graph on n vertices is exactly $3^{n/3}$ (up to a polynomial factor). The main results of this paper are $O(1.3642^n)$ time algorithms to compute the number of maximal independent sets and maximal bicliques in a graph.

1 Introduction

Bicliques. Let the vertex sets X and Y be independent sets of a graph $G = (V, E)$ such that $xy \in E$ for all $x \in X$ and all $y \in Y$. The subgraph of G induced by $X \cup Y$ is called a *biclique* of G. Furthermore depending on the context and the application area, one also calls the pair (X, Y) or the vertex set $X \cup Y$ a biclique. From a graph-theoretic point of view it is natural to consider a biclique of a graph G as a complete bipartite induced subgraph of G. For technical reasons, we prefer to consider a biclique $B \subseteq V$ of a graph $G = (V, E)$ as a vertex set inducing a complete bipartite subgraph of G.

Maximal bicliques. A biclique $B \subseteq V$ of G is a *maximal biclique* of G if B is not properly contained in another biclique of G. A lot of the research on maximal bicliques and in particular on algorithms to enumerate all maximal bicliques of (bipartite) graphs with polynomial delay is motivated by the various applications of bicliques in (bipartite) graphs. Applications of bicliques in automata and language theory, graph compression, artificial intelligence and biology are

* A large part of the research was done while Serge Gaspers was visiting the University of Metz.

discussed in [2]. An important application in data mining is based on the formal concept analysis [10] where each concept is a maximal biclique of a bipartite graph.

Previous work. The complexity of algorithmic problems on bicliques has been studied extensively. First results were mentioned by Garey and Johnson [9], among them the NP-completeness of the balanced complete bipartite subgraph problem. The maximum biclique problem is polynomial for bipartite graphs [3], and NP-hard for general graphs [25]. The maximum edge biclique problem was shown to be NP-hard by Peeters [20].

Approximation algorithms for node and edge deletion biclique problems are given by Hochbaum [12]. Enumerating maximal bicliques has attracted a lot of attention in the last decade. The algorithms in [17,18] enumerate all maximal bicliques of a bipartite graph as concepts during the construction of the concept lattice. Nowadays there are polynomial delay enumeration algorithms for maximal bicliques in bipartite graphs [5,15] and general graphs [4,15]. There are also polynomial delay algorithms to enumerate all maximal non-induced bicliques of a graph [1,5].[1]

Prisner studied various aspects of bicliques in graphs. Among others, he showed that the maximum number of maximal bicliques in a bipartite graph on n vertices is $2^{n/2}$. He established a lower bound of $3^{n/3}$ and an upper bound of 1.6181^n (up to a polynomial factor) on the maximum number of maximal bicliques in a graph on n vertices [21].

Our Results. We use a simple polynomial-time Turing reduction to transform results on maximal independent sets into results on maximal bicliques. We also improve upon Prisner's upper bound and give a simple proof that the maximum number of maximal bicliques in a graph on n vertices is at most $n \cdot 3^{n/3}$. Our main result is a $O(1.3642^n)$ time algorithm to count all maximal independent sets in a graph, which is established by using the Measure & Conquer technique (see e.g. [7]). No such algorithm was known prior to our work. We show how to use it to count all maximal bicliques of a graph within the same time bound and also provide a lower bound for the running time of this algorithm.

2 Preliminaries

All graphs in this paper are simple and undirected. For a graph $G = (V, E)$, we let $n = |V|$ and $m = |E|$. An edge between vertices u and v is denoted by uv. The set of *neighbors* of a vertex $v \in V$ is the set of all vertices adjacent to v, denoted by $N(v)$. The *closed neighborhood* of a vertex v is $N[v] = \{v\} \cup N(v)$. The *distance* between two vertices u, v is the length of the shortest path from u to v. We denote by $N^k(v)$ the set of all vertices at distance k from v, and by

[1] When the condition that X and Y are independent sets in the definition of a biclique is dropped, then (X, Y) is called a *non-induced biclique* of G. In this case a different maximality notion is used. See e.g. [1].

$N^k[v]$ the set of all vertices at distance at most k from v. The *degree* of a vertex v is $d(v) = |N(v)|$. A *clique* is a set of vertices that are all pairwise adjacent, and an *independent set* is a set of vertices that are all pairwise non-adjacent. An independent set is *maximal* if it is not properly contained in another independent set. The subgraph of G induced by a vertex set $A \subseteq V$ is denoted by $G[A]$. A graph is called *bipartite* if its vertex set can be partitioned into two independent sets V and W. The *bipartite complement* of a bipartite graph $G = (V, W, E)$ is a bipartite graph having the vertices of G as its vertex set and the non-edges of G with an endpoint in V and another in W as its edge set.

3 Polynomial-Time Reductions

There is a natural relation between independent sets (and cliques) on one hand and bicliques on the other hand. Thus it is not surprising that polynomial-time Turing reductions (in fact mainly Karp reductions) have been used in various hardness proofs for problems on bicliques [9]. The famous polynomial delay algorithm of Johnson and Papadimitriou to enumerate all maximal independent sets [14] is used explicitly or implicitly in polynomial delay algorithms to enumerate maximal (non-induced) bicliques in (bipartite) graphs [1,4,5].

The first reduction simply recalls an often used argument.

Lemma 1 (Property A). *Let $G = (V, E)$ be a bipartite graph. Let H be the bipartite complement of G. Then B is a (maximal) biclique of G if and only if B is a (maximal) independent set of H.*

The above lemma implies, among others, that any algorithm enumerating all maximal independent sets within delay $f(n)$ can be transformed into an algorithm enumerating all maximal bicliques of a bipartite graph within delay $f(n)$. The known tight bound of $2^{n/2}$ for the maximum number of maximal bicliques in a bipartite graph given in [21] follows easily from Property A and the corresponding bound for maximal independent sets in [13]. Based on this property, Yannakakis observed that the problem of finding a maximum biclique in a bipartite graph is solvable in polynomial time [25].

The following property is central for our paper.

Lemma 2 (Property B). *Let $G = (V, E)$ be a graph. For every $v \in V$, the graph H_v is the graph with vertex set $V(H_v) = N(v) \cup N^2(v)$. Its edge set $E(H_v)$ consists of the following edges:*

- *$xy \in E(H_v)$ if $xy \in E$ and $x, y \in N(v)$,*
- *$xy \in E(H_v)$ if $xy \in E$ and $x, y \in N^2(v)$,*
- *$xy \in E(H_v)$ if $xy \notin E$, $x \in N(v)$ and $y \in N^2(v)$.*

Then $B \subseteq V$ is a (maximal) biclique of G if and only if $B - v$ is a (maximal) independent set of a graph H_v for some $v \in B$.

Proof. Let B be a (maximal) biclique of G. Take some $v \in B$. Then $B \subseteq \{v\} \cup N(v) \cup N^2(v)$ in G, where the independent sets X and Y of the biclique B satisfy $X \subseteq N(v)$ and $Y \subseteq \{v\} \cup N^2(v)$. Since B is a biclique and by the construction of H_v, we obtain that $B - v$ is an independent set of H_v. On the other hand, if B' is a (maximal) independent set of H_v, for some $v \in V$, then $B' \cap N(v)$ is an independent set of $G[N(v)]$ and $B' \cap N^2(v)$ is an independent set of $G[N^2(v)]$. Hence B' is a biclique of $G - v$ and $B' \cup \{v\}$ is a biclique of G.

Finally, due to the correspondence between bicliques and independent sets, this also holds for maximality by inclusion of vertices. □

The corresponding Turing reduction does not increase the number of vertices, since $|V(H_v)| \leq |V| - 1$. Thus this reduction is useful for exponential-time algorithms.

Corollary 1. *Given an algorithm to find a maximum independent set (respectively to count all independent sets of size k) of a graph in time $c^n n^{O(1)}$, there exists an algorithm to find a maximum biclique (respectively to count all bicliques of size k) of a graph in time $c^n n^{O(1)}$.*

Proof. To find a maximum biclique of a graph $G = (V, E)$, compute a maximum independent set for each H_v, $v \in V$, constructed according to Property B and return the largest set of vertices found. To count all bicliques of size k of a graph $G = (V, E)$ on n vertices, order the vertices of G: $V = \{v_1, v_2, \dots, v_n\}$. For $i = 1, \dots, n$, compute the number of independent sets of size $k - 1$ of $H^i_{v_i}$ where $H^i_{v_i}$ is obtained from $G^i = G[V \setminus \{v_1, v_2, \dots, v_{i-1}\}]$ using Property B. Adding up the results gives the number of bicliques of size k of G. □

By this corollary and the algorithms in [22,24], a maximum biclique of a graph can be found in time $O(1.2109^n)$ and all maximum bicliques of a graph can be counted in time $O(1.2377^n)$.

Note that Corollary 1 is not directly applicable to use an algorithm for counting maximal independent sets to count the maximal bicliques of a graph. The issues are that double-counting has to be avoided at the same time as the maximality of each counted biclique has to be ensured.

4 Improving Prisner's Bound

The maximum number of maximal bicliques in a graph on n vertices has been studied by Prisner [21]. He settled the question for bipartite graphs. The maximum number of maximal bicliques in a bipartite graph on n vertices is precisely $2^{n/2}$. For general graphs the question remained open. He established a lower bound of $3^{n/3}$ and an upper bound of $(1.618034^n + o(1)) \cdot n^{5/2}$ for the maximum number of maximal bicliques in a graph on n vertices. We settle the question via an elegant proof based on Property B.

Theorem 1. *The maximum number of maximal bicliques in a graph is at most $n \cdot 3^{n/3}$.*

Proof. Let n be a positive integer and let G be any graph on n vertices. Applying Property B, for every vertex $v \in V$, there is a one-to-one correspondence between the maximal bicliques B of G satisfying $v \in B$ and the maximal independent sets $B - v$ of the graph H_v. By a well-known theorem of Moon and Moser [16], the maximum number of maximal independent sets in a graph on n vertices is $3^{n/3}$. Thus the number of maximal bicliques containing vertex v is at most $3^{n/3}$ for each $v \in V$. Consequently G has at most $n \cdot 3^{n/3}$ maximal bicliques. □

Corollary 2. *The maximum number of maximal bicliques in a graph is* $3^{n/3}$ *(up to a polynomial factor).*

5 Counting Algorithms

A problem related to enumerating all maximal bicliques of a graph is to compute the number of maximal bicliques of a graph; faster than by simply enumerating all of them. By property B, an algorithm to count all maximal independent sets of a graph could be a cornerstone to design such an algorithm. However no non-trivial algorithm for counting maximal independent sets is known. It is known that the counting problem for maximal independent sets is #P-complete even when restricted to chordal graphs [19]. Hence our goal is to construct a fast exponential-time algorithm solving this problem.

5.1 Algorithm to Count All Maximal Independent Sets

Let $G = (F, M, E)$ be a *marked graph* which are the graphs dealt with by our algorithm. Vertices of F are called *free* and vertices of M are called *marked*. Let u be a vertex of $F \cup M$. The degree of u is the number of neighbors in $F \cup M$ and is denoted by $d(u)$. Given a set $D \subseteq (F \cup M)$, the set $N(u) \cap D$ is denoted by $N_D(u)$ and its cardinality is denoted by $d_D(u)$.

The following notions are crucial for our algorithm. A set $S \subseteq F$ is a maximal independent set (or shortly, MIS) of a marked graph $G = (F, M, E)$ if S is a MIS of $G[F]$. We say that the MIS S of $G = (F, M, E)$ satisfies property Π if each vertex of M has a neighbor in S.

Given a marked graph G, our algorithm computes the number of MISs of $G = (F, M, E)$ satisfying Π. Thus, a marked vertex u is used to force that each MIS S of G counted by the algorithm contains at least one free neighbor of u. This is particularly useful to guarantee that only maximal independent sets of the input graph are counted. In the remainder of this section, we suppose that G is a connected graph, otherwise the algorithm is called for each of its connected components, and the product of the results gives the number of MISs of G satisfying Π.

Given a simple graph $G' = (V, E)$, `#MaximalIS`$\big(G = (V, \emptyset, E)\big)$ returns the number of maximal independent sets of G'. (See the next page for the description of the algorithm.)

We emphasize that all the halting ((H1)–(H2)) and reduction ((R1)–(R7)) rules are necessary for our running time analysis (see Subsection 5.3). The

```
Algorithm #MaximalIS$(G = (F, M, E))$
Input: A marked graph $G = (F, M, E)$.
Output: The number of MISs of $G$ satisfying $\Pi$.
  // Reduction rules
  if $G$ is empty then
    return 1                                                        (H1)
  if there exists $u \in M$ s.t. $d_F(u) = 0$ then
    return 0                                                        (H2)
  if there exists $u \in M$ s.t. $N_F(u) = \{v\}$ then
    return #MaximalIS$(G = (F \setminus N[v], M \setminus N(v), E))$              (R1)
  if there exists $u \in F$ s.t. $d_F(u) = 0$ then
    return #MaximalIS$(G = (F \setminus N[u], M \setminus N(u), E))$              (R2)
  if there exists $u, v \in M$ s.t. $\{u, v\} \in E$ then
    return #MaximalIS$(G = (F, M, E \setminus \{u, v\}))$                       (R3)
  if there exists $u, v \in F$ s.t. $N[u] = N[v]$ then
    count $\leftarrow$ #MaximalIS$(G = (F \setminus \{v\}, M, E))$
    Let $\mathrm{MIS}_u$ be the number of MISs computed by
    #MaximalIS$(G = (F \setminus \{v\}, M, E))$ containing $u$
    return $\mathrm{MIS}_u$ + count                                          (R4)
  if there exists $u \in M$ and $v \in N(u)$ s.t. $N[v] \subseteq N[u]$ then
    return #MaximalIS$(G = (F, M \setminus \{u\}, E))$                         (R5)
  if there exists $u, v \in M$ s.t. $N(u) = N(v)$ then
    return #MaximalIS$(G = (F, M \setminus \{v\}, E))$                         (R6)
  if there exists $u \in F \cup M$ and $v \in F$ s.t. $N(u) = N(v)$ then
    return #MaximalIS$(G = (F \setminus \{v\}, M, E))$                         (R7)
  // Branching rule                                                  (B)
  if there exists a marked vertex $u$ with $d(u) = 2$ then
    Choose $u$
  else
    Choose a vertex $u \in (F \cup M)$ such that
      (i) $u$ has minimum degree among all vertices in $F \cup M$
      (ii) among all vertices fulfilling (i), $u$ has a neighbor of maximum degree
      (iii) among all vertices fulfilling (ii), $u$ has maximum dual degree (i.e. the
      sum of the degrees of its neighbors)
  Let BL$(u) \leftarrow [v_1, \ldots, v_{d_F(u)}]$ be an ordered list of $N_F(u)$ such that:
    (i) $v_1$ is a vertex of $N_F(u)$ having a minimum number of neighbors in $V \setminus N(u)$
    (ii) append (in any order) the vertices of $N(v_1) \cap N_F(u)$ to the ordered list
    (iii) append the vertices of $N_F(u) \setminus N[v_1]$ ordered by increasing number of
    neighbors in $V \setminus N(u)$
  count $\leftarrow 0$
  if $u$ is free then // select $u$ (to be in the current MIS)
    count $\leftarrow$ #MaximalIS$(G = (F \setminus N[u], M \setminus N(u), E))$
  foreach $v_i \in BL(u)$ do // mark each vertex of $M'$ and select $v_i$
    $M' \leftarrow \{v_j \in \mathrm{BL}(u) : 1 \le j < i$ and $\{v_j, v_i\} \notin E\}$
    count $\leftarrow$ count+ #MaximalIS$(G = (F \setminus (M' \cup N[v_i]), (M \cup M') \setminus N(v_i), E))$
  return count
```

branching rule (B) selects a vertex u, orders its free neighbors in a list $[v_1, v_2, \ldots, v_{d_F(u)}]$ and makes a recursive call (i.e. a branching) counting all MISs containing u, and a recursive call for each $i = 1, 2, \ldots, d_F(u)$ where it counts all MISs containing v_i but none of $v_1, v_2, \ldots, v_{i-1}$.

The selected vertex u is chosen according to three criteria (i)–(iii). By (i), u has minimum degree, which ensures either that the algorithm makes few recursive calls or that many vertices are removed in each branching. By (ii), u has a neighbor of maximum degree among all vertices satisfying (i). If the degree of this neighbor is high, then many vertices are removed in at least one recursive call. If the degree of this vertex is low, every vertex of minimum degree has no high-degree neighbor. This property is exploited in the analysis of our algorithm, which considers a decrease in the degree of a vertex of small degree more advantageous than a decrease in the degree of a high-degree vertex. Similarly, (iii) ensures either many recursive calls where many vertices are removed or a knowledge on the degrees of the neighbors of a vertex of minimum degree. The ordered list $\mathtt{BL}(u)$ is defined in this way to ensure that for certain configurations of $N^2[u]$, reduction rule (R1) or a (fast) subsequent branching on a marked vertex of degree 2 is applied in many recursive calls.

5.2 Correctness of `#MaximalIS`

We show the correctness of the branching and reduction rules of `#MaximalIS`. **(H1)** If the input graph is empty then the only MIS is the empty set. **(H2)** If there is a marked vertex u without any free neighbor then there is no MIS satisfying Π. **(R1)** If a marked vertex u has only one free neighbor v, the vertex v has to be in the MIS to satisfy Π. **(R2)** By maximality, each free vertex without any free neighbor has to belong to all MISs. **(R3)** Since marked vertices cannot belong to any MIS, edges between two marked vertices are irrelevant and can be removed. **(R4)** Suppose $u, v \in F$ are two free vertices and $N[u] = N[v]$. Every MIS containing a neighbor of u does not contain v. Moreover, every MIS containing u can be replaced by one containing v instead of u. Thus, it is sufficient to remove v and to return the number of MISs containing a neighbor of u plus twice the number of MISs containing u. (Note that the algorithm can easily be implemented such that the number of MISs containing u is obtained from the recursive call. E.g., keep a counter to associate to each free vertex the number of MISs containing this vertex.) **(R5)** If $u \in M$ has a neighbor v such that all neighbors of v are also neighbors of u, then every MIS of $G - u$ must contain a vertex of $N[v] \setminus \{u\}$ and thus a neighbor of u in G. **(R6)** If two marked vertices have the same neighborhood then one of them is irrelevant. **(R7)** Let v be a free vertex and u a vertex such that $N(u) = N(v)$, and thus u and v are non adjacent. Hence every MIS containing a neighbor of u does not contain v and every MIS containing u (if u is free) also contains v. Thus the number of MISs is the same as for $G - v$.

(B) The algorithm considers the two possibilities that either u or at least one neighbor of u is in the current MIS. By induction and the fact that $N[u]$ is removed if the algorithm decides to add u to the current MIS, every MIS containing u is counted and it is counted only once. Consider the possibility that at least one neighbor of u is in the current MIS and let v_i be the first such neighbor in the ordered list $\texttt{BL}(u)$, containing all the free neighbors of u. That no MIS containing a vertex appearing before v_i in $\texttt{BL}(u)$ is counted, is ensured by either its deletion (because it is a neighbor of v_i) or the marking of this vertex. So, every MIS containing v_i but neither u (removed as it is a neighbor of v_i) nor a vertex appearing before v_i in $\texttt{BL}(u)$ is counted exactly once.

5.3 Running Time Analysis of #MaximalIS

We analyze the running time of our algorithm using the Measure & Conquer technique which has recently been used to establish several of today's best known exact exponential-time algorithms for NP-hard problems. For some important results and more details on the technique, we refer to [6,7,8,23]. To analyze the running time of our algorithm, we use the following measure $\mu(G)$ of a marked graph G.

$$\mu = \mu\big(G = (F, M, E)\big) = \sum_{i=1}^{n-1} w_i |V_i|$$

The weights w_i, $1 \le i \le n-1$ are real numbers taken from $[0,1]$ that will be fixed later. For $1 \le i \le n-1$, V_i denotes the set of vertices of degree i in G. The following values will be useful in the analysis.

$$\Delta w_i = \begin{cases} w_i - w_{i-1} & \text{if } 2 \le i \le n-1 \\ w_1 & \text{if } i = 1 \end{cases}$$

To further simplify the forthcoming analysis, we assume that $w_i = 1$ (for $4 \le i \le n-1$), $w_{i-1} \le w_i$ (for $2 \le i \le n-1$), and $\Delta w_i \ge \Delta w_{i+1}$ (for $1 \le i \le n-1$). It is not hard to see that an application of a reduction rule will not increase the measure of the marked graph. Furthermore no reduction rule can be applied more than n times, respectively m times for (R3). Finally, each reduction rule can be implemented to run in polynomial time, and thus for each subproblem the running time of our algorithm, excluding the recursive calls by branching rule (B), is polynomial. Consequently we need to analyze the maximum number of such recursive calls, i.e. the maximum number of subproblems generated by a recursive call by (B), during the execution of our algorithm on a marked graph of measure μ, which we denote by $T(\mu)$.

We only have to analyze the changes in measure when applying branching rule (B).

Case 1: (B) is applied to a marked vertex u with $d(u) = 2$.
Let v_1 and v_2 be its two neighbors. By (R3), i.e. since (R3) could not be applied, $v_1, v_2 \in F$, and by (R2), $d(v_1), d(v_2) \ge 2$.

(a) Suppose $d(v_1) = d(v_2) = 2$. For $i \in \{1, 2\}$, let x_i be the other neighbor of v_i. If $d(x_1) = d(x_2) = 1$ then the algorithm deals with a component of constant size, and the number of MISs of such a component can be computed in constant time. Suppose now that $d(x_1) \ge 2$. In the first branch (or subproblem) u, v_1 and x_1 are removed. In the second branch u, v_2 and x_2 are removed. Thus, the corresponding recurrence is majorized by $T(\mu) \leq T(\mu - 3w_2) + T(\mu - w_1 - 2w_2)$.

(b) Suppose $d(v_1) \geq 3$ and $d(v_2) \geq 2$. In the first branch u, v_1 and at least two other neighbors of v_1 are removed. In the second branch u, v_2 and the other neighbors of v_2, at least one, are removed. Thus, the corresponding recurrence is majorized by $T(\mu) \leq T(\mu - 2w_1 - w_2 - w_3) + T(\mu - w_1 - 2w_2)$. Since $w_2 \leq w_3$ and $w_2 \leq 2w_1$ (recall that $\Delta w_1 \geq \Delta w_2$), it follows that $3w_2 \leq 2w_1 + w_2 + w_3$ and thus the solution of the recurrence in case (b) is not worse than the one of case (a).

Case 2: Vertex u is chosen by the *else* statement of (B).
Thus u satisfies the conditions (i), (ii) and (iii). Let $[v_1, \dots, v_{d_F(u)}]$ be the *Branching List*, short $\mathtt{BL}(u)$, built by the algorithm. Given a vertex v_i, $1 \leq i \leq d_F(u)$, of $\mathtt{BL}(u)$, we denote by $\mathtt{Op}(v_i)$ the operation of adding v_i to the current MIS, removing $N[v_i]$ and marking the vertices $v_1, \dots, v_{i-1}$ that are not adjacent to v_i.

Let Δ_u denote the gain on the measure obtained by adding u to the current MIS. Removing u and its neighbors from the graph decreases $\mu(G)$ by $w_{d(u)} + \sum_{v \in N(u)} w_{d(v)}$. Moreover, the decrease of the degrees of vertices in $N^2(u)$ implies a gain of $\sum_{x \in N^2(u)} (w_{d(x)} - w_{d(x) - d_{N(u)}(x)})$. Thus, $\Delta_u = w_{d(u)} + \sum_{v \in N(u)} w_{d(v)} + \sum_{x \in N^2(u)} (w_{d(x)} - w_{d(x) - d_{N(u)}(x)})$.

Let $\Delta_{\mathtt{Op}(v_i)}$ denote the gain on the measure when $v_i \in \mathtt{BL}(u)$, $1 \leq i \leq d_F(u)$, is selected and added to the maximal independent set. Again, by selecting vertex v_i the vertices of $N[v_i]$ are removed and thus a gain of $w_{d(v_i)} + \sum_{x \in N(v_i)} w_{d(x)}$ is obtained. Since neighbors of vertices of $N^2(v_i)$ have been removed, we gain $\sum_{y \in N^2(v_i)} (w_{d(y)} - w_{d(y) - d_{N(v_i)}(y)})$. The measure further decreases whenever among the marked vertices of $\{v_1, \dots, v_{i-1}\}$, some of them have only one remaining free neighbor after the deletion of $N[v_i]$. By direct application of reduction rule (R1), these vertices and their neighbors are also removed from the graph. We denote this *extra* gain by $\text{marked}_1(\mathtt{Op}(v_i))$ Thus, $\Delta_{\mathtt{Op}(v_i)} = w_{d(v_i)} + \sum_{x \in N(v_i)} w_{d(x)} + \sum_{y \in N^2(v_i)} (w_{d(y)} - w_{d(y) - d_{N(v_i)}(y)}) + \text{marked}_1(\mathtt{Op}(v_i))$.

Putting all together, we obtain the following general recurrence for case 2:

$$T(\mu) \leq T(\mu - \Delta_u) + \sum_{v_i \in \mathtt{BL}(u)} T(\mu - \Delta_{\mathtt{Op}(v_i)})$$

Finally, we conclude the time analysis by Measure & Conquer. We solve the corresponding system of linear recurrences and establish an upper bound on the worst case running time of our algorithm. Moreover, for some cases where a marked vertex of degree 2 appears, we combine the analysis of the case with the subsequent branching on this vertex. The key step is to choose the weights w_1, w_2 and w_3 such that the worst-case solution taken over all recurrences is minimized (see e.g. [7,8]). Using the weights $w_1 = 0.8512$, $w_2 = 0.9194$ and $w_3 = 0.9877$, we obtain:

Theorem 2. *Algorithm* `#MaximalIS` *counts all MISs of a given graph G in time $O(1.3642^n)$, where n is the number of vertices of G.*

Typically using a computer program, first the collection of recurrences that are obtained for all possible cases of vertices, degrees, etc. in the general recurrence are computed and then the optimal values of the weights are computed. Although for our problem the number of recurrences is still rather moderate, due to space limitations the detailed analysis is not given in this extended abstract.

Remark 1. Given a marked graph of maximum degree 2, `#MaximalIS` takes exponential time. A dynamic programming approach can also be used to count in polynomial time all MISs of a such marked graph. Adding this polynomial time procedure to `#MaximalIS` is likely to be of help in implementations of the algorithm, however it does not improve its worst case running time.

For most non-trivial Branch-and-Reduce algorithms, it is not known whether the upper bound of the running time provided by the currently available analysis is tight or not. A lower bound for the worst case running time of such algorithms is therefore desirable (see e.g. [7,8]).

Theorem 3. *There exists an infinite family of graphs for which* `#MaximalIS` *takes time $\Omega(1.3247^n)$, and thus its worst case running time is $\Omega(1.3247^n)$.*

Proof. The lower bound for the running time of Algorithm `#MaximalIS` established here uses the same family of graphs as the lower bound for an algorithm computing a minimum independent dominating set [11].

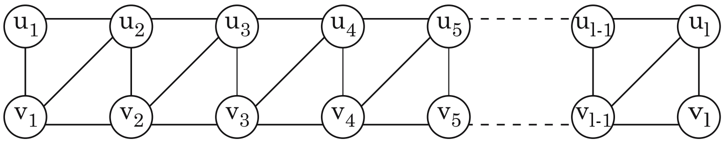


Fig. 1. The graph G_l

Consider the graph G_l of Figure 1. It has $n = 2l$ vertices. Note that none of the reduction or halting rules are applicable to G_l. The first branching of `#MaximalIS` is on vertex u_1 or vertex v_l. Without loss of generality, suppose the algorithm always chooses the vertex with smallest index when it has more than one choice (i.e. it chooses u_1 for the first recursive call).

The branching rule (B) then makes recursive calls on graphs with $n-3$, $n-4$ and $n-5$ vertices, not marking any vertex. The structure of all resulting graphs is similar to G_l: either isomorphic to G_{l-2} or equal to $G_l \setminus N[u_1]$ or $G_l \setminus N[u_2]$. The subsequent recursive calls again remove 3, 4 and 5 vertices in each case and do not mark any vertices. Unless the graph has at most 4 vertices, each application of branching rule (B) satisfies the recurrence $T(n) = T(n-3)+T(n-4)+T(n-5)$ for this graph and therefore the running time for this class of graphs is $\Omega(\alpha^n)$ where α is the positive root of $x^{-3}+x^{-4}+x^{-5}-1$ (i.e. $1.3247 < \alpha < 1.3248$). □

5.4 Algorithm to Count All Maximal Bicliques

Finally, we consider the problem of counting all maximal bicliques of a graph $G = (V, E)$. Let $G' = (V', E')$ be a copy of G. Let $G'' = (V'', E'')$ where $V'' = V \cup V'$ and $E'' = E \cup E' \cup \{xy' : x = y \text{ or } xy \notin E\}$.

Lemma 3. *The number of MISs of G'' equals twice the number of maximal bicliques of G.*

Proof. We show that there is a one-to-one correspondence between the bicliques of G and the symmetric pairs of independent sets of G''.

Let $X \cup Y$ be a biclique of G. Clearly, X, Y are independent sets in G and their copies X', Y' are independent sets in G'. Let $x \in X$ and $y \in Y$. Then $xy, x'y' \in E''$ and $xy', x'y \notin E''$. So, $X \cup Y'$ and $X' \cup Y$ are independent sets in G''.

Let $X, Y \subseteq V$ be such that $X \cup Y'$ is an independent set in G'' where X', Y' are the copies of X, Y. Hence X, Y are independent sets in G. Let $x \in X$ and $y' \in Y'$. Then $xy \in E$. So, $X \cup Y$ is a biclique in G. By the symmetry of G'', the independent set $X' \cup Y$ in G'' also corresponds to the biclique $X \cup Y$ in G.

Clearly, this correspondence also holds for maximality by inclusion of vertices. This implies that $X \cup Y$ is a maximal biclique of G iff $X \cup Y'$, and thus also $Y \cup X'$, are MISs of G''. $\square$

With this construction and the algorithm for counting all MISs of a graph, we are now able to give an algorithm for counting all maximal bicliques of a graph.

Theorem 4. *There is an algorithm that counts all maximal bicliques of a graph in time $O(1.3642^n)$.*

Proof. The algorithm simply calls `#MaximalIS`$\big((V'', \emptyset, E'')\big)$ and divides the result by 2. Notice that G'' has $2n$ vertices and that every vertex of G'' has degree n. The first application of branching rule (B) makes $n+1$ recursive calls and in each one, $n+1$ vertices are removed from the marked graph. Thus the running time is $(n+1)(c^{n-1})n^{O(1)}$ where $c^n n^{O(1)}$ is the running time of `#MaximalIS` on a graph with n vertices. The constant $c = 1.3642$ was rounded to derive the running time for `#MaximalIS`, and thus the running time of the algorithm to count maximal bicliques is $O(1.3642^n)$. $\square$

References

1. Alexe, G., Alexe, S., Crama, Y., Foldes, S., Hammer, P.L., Simeone, B.: Consensus algorithms for the generation of all maximal bicliques. Discrete Appl. Math. 145, 11–21 (2004)
2. Amilhastre, J., Vilarem, M.C., Janssen, P.: Complexity of minimum biclique cover and minimum biclique decomposition for bipartite dominofree graphs. Discrete Appl. Math. 86, 125–144 (1998)
3. Dawande, M., Swaminathan, J., Keskinocak, P., Tayur, S.: On bipartite and multipartite clique problems. J. Algorithms 41, 388–403 (2001)

4. Dias, V.M.F., Herrera de Figueiredo, C.M., Szwarcfiter, J.L.: Generating bicliques of a graph in lexicographic order. Theoret. Comput. Sci. 337, 240–248 (2005)
5. Dias, V.M.F., Herrera de Figueiredo, C.M., Szwarcfiter, J.L.: On the generation of bicliques of a graph. Discrete Appl. Math. 155, 1826–1832 (2007)
6. Fomin, F.V., Gaspers, S., Pyatkin, A.V., Razgon, I.: On the minimum feedback vertex set problem: Exact and enumeration algorithms. Algorithmica, Special Issue on Parameterized and Exact Algorithms (to appear)
7. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and conquer: Domination – A case study. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 192–203. Springer, Heidelberg (2005)
8. Fomin, F.V., Grandoni, F., Kratsch, D.: Measure and conquer: A simple $O(2^{0.288n})$ independent set algorithm. In: SODA 2006, pp. 18–25. ACM Press, New York (2006)
9. Garey, M.R., Johnson, D.S.: Computers and Intractability: A guide to the Theory of NP-completeness. Freeman, New York (1979)
10. Ganter, B., Wille, R.: Formal Concept Analysis, Mathematical Foundations. Springer, Berlin (1996)
11. Gaspers, S., Liedloff, M.: A branch-and-reduce algorithm for finding a minimum independent dominating set in graphs. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 78–89. Springer, Heidelberg (2006)
12. Hochbaum, D.S.: Approximating clique and biclique problems. J. Algorithms 29, 174–200 (1998)
13. Hujter, M., Tuza, Z.: The number of maximal independent sets in triangle-free graphs. SIAM J. Discrete Math. 6, 284–288 (1993)
14. Johnson, D.S., Papadimitriou, C.H.: On generating all maximal independent sets. Inf. Process. Lett. 27, 119–123 (1988)
15. Makino, K., Uno, T.: New algorithms for enumerating all maximal cliques. In: Hagerup, T., Katajainen, J. (eds.) SWAT 2004. LNCS, vol. 3111, pp. 260–272. Springer, Heidelberg (2004)
16. Moon, J.W., Moser, L.: On cliques in graphs. Israel J. Math. 3, 23–28 (1965)
17. Nourine, L., Raynaud, O.: A Fast Algorithm for Building Lattices. Inf. Process. Lett. 71, 199–204 (1999)
18. Nourine, L., Raynaud, O.: A fast incremental algorithm for building lattices. J. Exp. Theor. Artif. Intell. 14, 217–227 (2002)
19. Okamoto, Y., Uno, T., Uehara, R.: Linear-Time Counting Algorithms for Independent Sets in Chordal Graphs. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 433–444. Springer, Heidelberg (2005)
20. Peeters, R.: The maximum edge biclique problem is NP-complete. Discrete Appl. Math. 131, 651–654 (2003)
21. Prisner, E.: Bicliques in Graphs I: Bounds on Their Number. Combinatorica 20, 109–117 (2000)
22. Robson, J.M.: Algorithms for maximum independent sets. J. Algorithms 7, 425–440 (1986)
23. van Rooij, J.M.M., Bodlaender, H.L.: Design by measure and conquer: a faster exact algorithm for dominating set. In: Albers, S., Weil, P. (eds.) STACS 2008. IBFI, Schloss Dagstuhl, Germany, pp. 657–668 (2008)
24. Wahlström, M.: A tighter bound for counting max-weight solutions to 2SAT instances. In: Grohe, M., Niedermeier, R. (eds.) IWPEC 2008. LNCS, vol. 5018, pp. 202–213. Springer, Heidelberg (2008)
25. Yannakakis, M.: Node and edge deletion NP-complete problems. In: STOC 1978, pp. 253–264. ACM, New York (1978)

Evaluations of Graph Polynomials

Benny Godlin, Tomer Kotek, and Johann A. Makowsky[⋆]

Department of Computer Science
Technion–Israel Institute of Technology, Haifa, Israel
{bgodlin,tkotek,janos}@cs.technion.ac.il

Abstract. A graph polynomial $p(G, \bar{X})$ can code numeric information about the underlying graph G in various ways: as its degree, as one of its specific coefficients or as evaluations at specific points $\bar{X} = \bar{x}_0$. In this paper we study the question how to prove that a given graph parameter, say $\omega(G)$, the size of the maximal clique of G, cannot be a fixed coefficient or the evaluation at any point of the Tutte polynomial, the interlace polynomial, or any graph polynomial of some infinite family of graph polynomials.

Our result is very general. We give a sufficient condition in terms of the connection matrix of graph parameter $f(G)$ which implies that it cannot be the evaluation of any graph polynomial which is invariantly definable in $CMSOL$, the Monadic Second Order Logic augmented with modular counting quantifiers. This criterion covers most of the graph polynomials known from the literature.

1 Introduction

1.1 Graph Parameters and Graph Polynomials

Graph parameters (also called numeric graph invariants) f are functions from the class of all finite graphs $\mathcal{G}$ to some numeric domain which is a commutative ring with 0 and 1, usually the integers $\mathbb{Z}$, the rational numbers $\mathbb{Q}$ or the reals $\mathbb{R}$. Graph properties are a special case where the value is 0 or 1.

Graph polynomials are functions p from $\mathcal{G}$ into a polynomial ring, usually $\mathbb{Z}[\bar{X}]$, where $\bar{X}$ is a fixed finite set of indeterminates. Graph polynomials are a way to encode infinitely many graph parameters. Every evaluation of the polynomial $p(G; \bar{X})$ at some point $\bar{X} = \bar{x_0}$ is a graph parameter. So are the coefficients of $p(G; \bar{X})$, the total degree or the degree of monomials where the coefficient satisfies certain properties, and the zeros of $p(G; \bar{X})$.

A particular graph polynomial is considered interesting if it encodes many useful graph parameters. Let $G = (V(G), E(G))$ be a graph. The characteristic polynomial $P(G, X)$ of a graph is defined as the characteristic polynomial (in

[⋆] Partially supported by the Israel Science Foundation for the project "Model Theoretic Interpretations of Counting Functions" (2007-2010) and the Grant for Promotion of Research by the Technion–Israel Institute of Technology.

the sense of linear algebra) of the adjacency matrix A_G of G. The coefficients of $P(G, X)$ are defined by

$$det(X \cdot \mathbf{1} - A_G) = \sum_{i=0}^{n} c_i(G) \cdot X^i.$$

It is well known that $n = |V(G)|$, $-c_2(G) = |E(G)|$, and $-c_3(G)$ equals twice the number of triangles of G. The second largest zero $\lambda_2(G)$ of $P(G; X)$ gives a lower bound to the conductivity of G, cf. [GR01]. All these are graph parameters.

The Tutte polynomial of G is defined as

$$T(G; X, Y) = \sum_{F \subseteq E(G)} (X-1)^{r\langle E\rangle - r\langle F\rangle}(Y-1)^{n\langle F\rangle} \tag{1}$$

where $k\langle F\rangle$ is the number of connected components of the spanning subgraph defined by F, $r\langle F\rangle = |V| - k\langle F\rangle$ is its rank and $n\langle F\rangle = |F| - |V| + k\langle F\rangle$ is its nullity.

The Tutte polynomial $T(G; X, Y)$ has remarkable evaluations which count certain configurations of the graph G, cf. [Wel93].

(i) $T(G; 1, 1)$ is the number of spanning trees of G,
(ii) $T(G; 1, 2)$ is the number of connected spanning subgraphs of G,
(iii) $T(G; 2, 1)$ is the number of spanning forest of G,
(iv) $T(G; 2, 2) = 2^{|E|}$ is the number of spanning subgraphs of G,
(v) $T(G; 1-k, 0)$ is the number of proper k-vertex colorings of G,
(vi) $T(G; 2, 0)$ is the number of acyclic orientations of G,
(vii) $T(G; 0, -2)$ is the number of Eulerian orientations of G.

All these are also graph parameters which take values in $\mathbb{N}$. More sophisticated evaluations of the Tutte polynomial can be found in [Goo06, Goo08].

We shall show, 4, that the maximal size of a clique in G, $\omega(G)$, is not an evaluation of the Tutte polynomial. The same can be shown for the number of cliques of maximal size.

1.2 Coefficients and Evaluation of Graph Polynomials

In this paper we study the question whether a given graph parameter can occur as specific coefficient or the evaluation of a graph polynomial. We do not deal with zeros of graph polynomials. We are mostly interested in negative results: how does one prove that a given graph parameter cannot be a specific coefficient or the evaluation of a family of graph polynomials?

As a simple example, we look at the graph parameter f which is *additive* with respect to the disjoint union, in other words $f(G_1 \sqcup G_2) = f(G_1) + f(G_2)$. This is true for $|V(G)|, |E(G)|$ and also for $k(G)$, $b(G)$, number of connected components and number of blocks (doubly connected components), respectively. They are evaluations of the graph polynomials

$$V(G;X) = \sum_{v \in V(G)} X \quad \text{and} \quad E(G;X) = \sum_{e \in E(G)} X$$

$$C(G;X) = \sum_{U \subseteq V(G): U \in Co} X \quad \text{and} \quad B(G;X) = \sum_{U \subseteq V(G): U \in Bl} X$$

respectively, for $X = 1$. Here Co is the set of connected components, and Bl is the set of blocks respectively.

On the other hand, many graph polynomials $p(G, \bar{X})$, like the Tutte polynomial, are *multiplicative*, i.e., $p(G_1 \sqcup G_2, \bar{X}) = p(G_1, \bar{X}) \cdot p(G_2, \bar{X})$. Clearly, an additive graph parameter cannot be the evaluation of a multiplicative graph polynomial. If we consider instead the graph parameters $2^{|V(G)|}, 2^{|E(G)|}$, we can see that they are evaluations of the, admittedly uninteresting, multiplicative graph polynomial $X^{|V(G)|} \cdot Y^{|E(G)|}$.

More interesting[1] integer-valued graph parameters have the following property: There exist infinite sequences of graphs $(G_i)_{i \in \mathbb{N}}, (H_i)_{i \in \mathbb{N}}$ such that

(i) for all $(i,j) \in \mathbb{N}^2$ $f(G_i \sqcup H_j) = \max f(G_i), f(H_j)$ or $f(G_i \sqcup H_j) = \min f(G_i), f(H_j)$, and
(ii) for all $i \in \mathbb{N}$ the sequence $f_j = f(G_i \sqcup H_j)$ is strictly monotone increasing.

An integer-valued graph parameter f which has this property is called *maximizing*, respectively *minimizing*. In the above property we can replace the disjoint union by the join of two graphs $G_1 \bowtie G_2$ and consider the corresponding property. In this case we say that f is *join-maximizing*, respectively *join-minimizing*. Similarly we speak of *join-additive* and *join-multiplicative* graph parameters.

Example 1. (i) Among the maximizing graph parameters we have: the chromatic number $\chi(G)$, the edge chromatic number $\chi_e(G)$, and the total coloring number $\chi_t(G)$, the size of a maximal clique $\omega(G)$, the size of the maximal degree $\Delta(G)$, the tree-width $tw(G)$, and the clique-width $cw(G)$.
(ii) Among the minimizing graph parameters we have: the minimum degree $\delta(G)$, and the girth $g(G)$, which is the minimum length of a cycle in G.
(iii) $\omega(G)$ is join-additive and maximizing.
(iv) $\alpha(G)$, the size of the maximal independent set, is additive and join-maximizing.

How can we decide whether any of these do or do not occur as the evaluation of a graph polynomial? What about the average degree

$$d(G) = \frac{1}{|V(G)|} \cdot \sum_{v \in V(G)} d_G(v),$$

where $d_G(v)$ denotes the degree of a vertex v of G, which behaves differently than the example listed above?

[1] Almost all graph parameters discussed are taken from [Die96]. One exception is the clique-width, which was introduced in [CO00], and, in connection to graph polynomials, in [Mak04].

1.3 Connection Matrices

In [FLS07] the *connection matrix* $M(f,0)$ of a graph parameter f was introduced. Let $(G_i)_{i\in\mathbb{N}}$ be an enumeration of all finite graphs (up to isomorphism). $M(f,0)$ is an infinite matrix where the columns and rows are labeled by finite graphs G_i. Then the entry $M(f,0)_{i,j}$ is defined by $M(f,0)_{i,j} = f(G_i \sqcup G_j)$. We study the matrix $M(f,0)$ as a matrix over the reals $\mathbb{R}$. Similarly, we denote by $M(f,\bowtie)$ the matrix with entries $M(f,\bowtie)_{i,j}$ defined by $M(f,\bowtie)_{i,j} = f(G_i \bowtie G_j)$. Let us denote by $r(f,0)$ and $r(f,\bowtie)$ the rank of $M(f,0)$, respectively of $M(f,\bowtie)$.

Proposition 1. *(i) If f is multiplicative, $r(f,0)=1$.*
(ii) If f is additive, $r(f,0)=2$, unless $M(f,0)$ is the zero matrix.
(iii) If f is maximizing or minimizing, $r(f,0)$ is infinite.
(iv) Similarly for join-multiplicative, join-additive, join-maximizing and join-minimizing where the matrix is replaced by $M(f,\bowtie)$.
(v) For the average degree $d(G)$ of a graph, $r(d,0)$ is infinite.

Proof. The first three statements are easy. For $f=d(G)$ we have

$$M(d,0)_{i,j} = 2\frac{|E_i|+|E_j|}{|V_i|+|V_j|}.$$

This contains, for graphs with a fixed number e of edges, the Cauchy matrix $(\frac{2e}{i+j})_{i,j}$, hence $r(d,0)$ is infinite. □

In [FLS07] it is stated that for the Tutte polynomial $T(G;X,Y)$ and integers m,n the rank $r(T(G,m,n),0)$ is finite. Therefore, no integer valued graph parameter f with $r(f,0)$ infinite is an integer-valued evaluation of the $T(G;X,Y)$. What about rational values, like in the case of $d(G)$? What about other graph polynomials from the vast literature? Can we extend this argument to infinite families of graph polynomials $\mathcal{P}$ and to all their real evaluations?

In [FLS07], for every $k\in\mathbb{N}$ a more general connection matrix $M(f,k)$ and its rank $r(f,k)$ is introduced, and the finiteness of this rank is established for many graph parameters. We shall discuss this in Section 2. Our goal is to find a general criterion which allows us to conclude that for a graph parameter f the rank $r(f,k)$ is finite.

1.4 Main Result

In [Mak04, Mak07], the class of graph polynomials invariantly definable in Monadic Second Order Logic with Modular Counting, *CMSOL*, was introduced. A precise definition is given in Section 3 . For now it suffices to know that the Tutte polynomial, the matching polynomial, the characteristic polynomial, the interlace polynomial, and virtually all graph polynomials in the literature, are invariantly *CMSOL*-definable.

Theorem 2. *Let $p(G,\bar{X})$ be an invariantly CMSOL-definable graph polynomial with values in $\mathbb{R}[\bar{X}]$ with m indeterminates. There are numbers $\gamma(k)\in\mathbb{N}$ depending on the polynomial p and k only, and $\beta\in\mathbb{N}$, such that for all $\bar{x}_0\in\mathbb{R}^m$, we have $r(p(G,\bar{x}_0),k)\le\gamma(k)$. and $r(p(G,\bar{x}_0),\bowtie)\le\beta$.*

Theorem 3. *Let $p(G, \bar{X})$ be an invariantly CMSOL-definable graph polynomial with values in $\mathbb{R}[\bar{X}]$ with m indeterminates $X_1, \ldots, X_m$, and let*

$$X_1^{\alpha_1} \cdot X_2^{\alpha_2} \cdot \ldots \cdot X_m^{\alpha_m}$$

be a specific monomial of $p(G, \bar{X})$ with coefficient $c_\alpha(G)$, where $\alpha = (\alpha_1, \ldots, \alpha_m)$.

Then there is an invariantly CMSOL-definable graph polynomial $p_\alpha(G, \bar{X})$ such that $c_\alpha(G)$ is an evaluation of $p_\alpha(G, \bar{X})$.

Remark 1. Theorem 3 is also valid for monomials of the form

$$X_1^{n_1(G)-\alpha_1} \cdot X_2^{n_2(G)-\alpha_2} \cdot \ldots \cdot X_m^{n_m(G)-\alpha_m},$$

where $n_i(G) = |V(G)|$ or $n_i(G) = |E(G)|$. This can be used to get, for example, the coefficient of $X^{|V(G)|-3}$ of the characteristic polynomial.

We want to apply Theorems 2 and 3 to graph parameters which are discussed in, say, [Die96]. To do so we use Proposition 1.

Corollary 4. *The following graph parameters are not a specific coefficient nor an evaluation of some CMSOL-definable graph polynomial.*

(i) $d(G)$, the average rank of a graph G.
(ii) Any graph parameter f which is maximizing, minimizing, join-maximizing or join-minimizing.

Remark 2. The degree of a graph polynomial is a graph parameter which behaves differently than evaluations or coefficients. The degree of the characteristic polynomial $P(G; X)$ is $|V(G)|$.

(i) On the other hand, let $\omega_i(G)$ be the number of induced subgraphs of size i which are cliques. Consider the polynomial $\Omega(G; X) = \sum_i \omega_i(G) \cdot X^i$. Then the degree of $\Omega(G; X)$ is $\omega(G)$. However, $\omega(G)$ is maximizing, so by Theorems 2 and 3, $\omega(G)$ cannot be a fixed coefficient or evaluation of any invariantly *CMSOL*-definable polynomial.
(ii) Similarly, the size of a maximal independent set $\alpha(G)$, is the degree of the polynomial $Ind(G; X) = \sum_i ind_i(G) \cdot X^i$ where $ind_i(G)$ is the number of independent sets of size i. However, $\alpha(G)$ is join-maximizing, so by Theorems 2 and 3, $\alpha(G)$ cannot be a fixed coefficient or evaluation of any invariantly *CMSOL*-definable polynomial.

Both $\Omega(G; X)$ and $Ind(G; X)$ are both *CMSOL*-definable polynomials, as we shall see in Section 3.

The remainder of the paper is organized as follows: In the next section we discuss further generalizations of connection matrices and their use, and in Section 3 we present the necessary material on *CMSOL*-definable polynomials. In Section 4 we discuss further research.

2 Connection Matrices $M(f,k)$

In [FLS07] more general connection matrices $M(f,k)$ are introduced for every natural number k. Instead of looking at all graphs and their disjoint unions, they look at graphs with k vertices labeled with distinct labels from the set $[k]=\{0,\ldots,k-1\}$. Let $\mathcal{G}_k$ denote the class of k-labeled graphs. On $\mathcal{G}_k$ they define the operation $G_1 \sqcup_k G_2$, which is like the disjoint union, but vertices with corresponding labels are identified. This may create multiple edges. Let $(G_i^k)_{i\in\mathbb{N}}$ be an enumeration of all graphs from $\mathcal{G}_k$. The matrix $M(f,k)$ now has the entries $M(f,k)_{i,j}=f(G_i^k \sqcup_k G_j^k)$, and $r(f,k)$ denotes its rank.

Example 2. (i) For $\lambda\in\mathbb{N}^+$, let $\chi(G;\lambda)$ denote the number of proper λ-colorings of G. In [FLS07] it is shown that $r(\chi(G;\lambda),k)=B_{k,\lambda}$, where $B_{k,\lambda}$, denotes the number of partitions of a k-element set into at most λ parts.

(ii) Let $\omega_{conn}(G)$ be defined by

$$\omega_{conn}(G)=\begin{cases}\omega(G) & \text{if } G \text{ is connected}\\ 0 & \text{else}\end{cases}$$

where $\omega(G)$ is the size of the maximal clique of G. It is easy to see that $r(\omega_{conn},0)=2$, but

$$\omega_{conn}(G_0\sqcup_1 G_1)=\max\{\omega_{conn}(G_0),\omega_{conn}(G_1)\},$$

holds if both G_0 and G_1 are connected, hence, using Proposition 1, $r(\omega_{conn},1)$ is infinite.

(iii) If we replace connected by m-connected, we get a graph parameter ω_{m-conn} with $r(\omega_{m-conn},i)$ finite for $i\leq m$, and infinite otherwise.
This shows that the $r(f,k)$ can switch from finite to infinite at any stage.

Problem 1. What can we say in general about the behavior of $r(f,k)$ as a function of k?

Definition 1. *Let f be a graph parameter. f is k-*additive *if for all k-labeled graphs $G_1,G_2\in\mathcal{G}_k$ we have $f(G_1\sqcup_k G_2)=f(G_1)+f(G_2)$.*
*Similarly, f is k-*multiplicative, k-maximizing *or* k-minimizing *if the corresponding properties hold with the disjoint union replace by $\sqcup_k$.*

Similarly to Proposition 1 we have

Proposition 5. *If f is k-multiplicative, $r(f,k)=1$.*
If f is k-additive and $M(f,k)$ is not the zero matrix, $r(f,k)=2$.
If f is k-maximizing or k-minimizing, $r(f,k)$ is infinite.

Let $pm(G)$ denote the number of perfect matchings of G. In [FLS07], it is shown that $r(pm,k)=2^k$. For the proof they define auxiliary graph parameters $pm_A(G)$ for k-graphs as follows: Let $A\subseteq[k]$ be a set of labels. Denote by

$pm_A(G)$ the number of matchings in G that match all the unlabeled vertices and the vertices with label in A, but not any of the other labeled vertices. Then we have

$$pm(G_1 \sqcup_k G_2) = \sum_{A_1 \cap A_2 = \emptyset, A_1 \cup A_2 = [k]} pm_{A_1}(G_1) \cdot pm_{A_2}(G_2)$$

We generalize this as follows:

Definition 2. *A graph parameter f is* weakly (k, γ)-multiplicative*, if there exists a finite set of graph parameters $f_i : i \leq \gamma$ with $i, \gamma \in \mathbb{N}$ with $f = f_0$, and a matrix $M^k \in \mathbb{R}^{\gamma \times \gamma}$, such that $f_0(G_1 \sqcup_k G_2) = \sum_{i,j} f_i(G_1) M^k_{ij} f_j(G_2)$.*
In other words, $f(G_1 \sqcup_k G_2)$ is given by a quadratic form defined by $M^k_{i,j}$ of rank at most γ. The number $\gamma = \gamma(k)$ may depend on k. Similarly we define weakly $(\bowtie, \gamma)$-multiplicative*, where sqcup$_k$ is replaced by the join and the quadratic form is given by $M^{\bowtie}$.*

Proposition 6. *Let f be a graph parameter which is weakly (k, γ)-multiplicative. Then $r(f, k) \leq \gamma$ where $\gamma = \gamma(k)$ depends on k.*

The following theorem is proven in [Mak04, Theorem 6.4]:

Theorem 7. *Let f be a graph parameter which is the evaluation $f(G) = p(G, \bar{x}_0)$ of an invariantly CMSOL definable graph polynomial $p(G, \bar{X})$. Then f is weakly $(k, \gamma(k))$-multiplicative and weakly $(\bowtie, \beta)$-multiplicative for $\gamma : \mathbb{N} \to \mathbb{N}$ and $\beta \in \mathbb{N}$, which depend on the polynomial p, but not on $\bar{x}_0$.*

Remark 3. The function $\gamma(k)$ may grow super-exponentially. The best general upper bounds known contains iterated exponentials, cf. [Mak04].

Theorem 2 now follows immediately from Theorem 7 and Proposition 6.

3 *CMSOL*-Definable Graph Polynomials

In this section we give the definition of invariantly *CMSOL*-definable polynomials *in normal form*. A full treatment is given in [Mak04] . There we give a more general definition of invariantly *CMSOL*-definable polynomials, and show that every such polynomial can be written in normal form. The more general definition makes it easier to verify that a given graph polynomial is indeed invariantly *CMSOL*-definable. All we need for the proof of our two main theorems is the definition of this normal form and Theorem 7.

3.1 The Logic *CMSOL*

We consider now *ordered k-graphs* of the form $G = (V, E, R, \leq, a_0, \ldots, a_{k-1})$, where V and E are finite sets of vertices and edges, respectively, $\leq$ is a linear order on V, and $R \subseteq V \times E \times V$ is the graph incidence relation. This allows us also to have directed graphs with multiple edges. $a_i : i \in [k]$ are the labeled elements of V.

We define the logic *CMSOL* for these graphs inductively. We have first order variables $x_i : i \in \mathbb{N}$ which range over elements of $V \cup E$, and (monadic) second order variables $U_i : i \in \mathbb{N}$, which range over subsets of $V \cup E$. Terms $t, t', \ldots$ are either first order variables or one of the constants $a_i : i \in [k]$. Atomic formulas are of the form $t = t'$, $t \leq t'$, $R(t, t', t'')$, $U_i(t)$ and have the natural interpretation. Formulas are built inductively using the connectives $\vee, \wedge, \rightarrow, \leftrightarrow, \neg$, and the quantifiers $\forall x_i$, $\exists x_i$, $\forall U_i$, $\exists U_i$ with their natural interpretation. Additionally we have quantifiers $C_{a,b} x_i$ for each $a, b \in \mathbb{N}$. The formula $C_{a,b} x_i \phi(x_i)$ is interpreted by the statement "the number of elements satisfying $\phi(x_i)$ equals a modulo b".

The following can be written as *CMSOL*-formulas:

Example 3. (i) The formula $\Phi_{match}(U)$ which says that U is a matching.
(ii) The formula $\Phi_{pm}(U)$ which says that U is a perfect matching.
(iii) The formula $\Phi_{fconn}(U, x)$ which says that x is the first element of some connected component of the spanning subgraph $[U]_G$ with edge set $U \subseteq E$.
(iv) The formula Φ_{euler} which says that every vertex has even degree and the graph is connected.
(v) The formula $\Phi_{ham}(U)$ which says that U is a Hamiltonian path.

3.2 Simple Invariantly *CMSOL*-Definable Polynomials

A *simple CMSOL*-definable polynomial in one indeterminate X is of the form

$$\sum_{U:\Phi(U)} \prod_{x:U(x)} X$$

where $\Phi(U)$ is an *CMSOL*-formula. Φ is called an iteration formula. It is *invariantly CMSOL*-definable, if its value does not depend on the ordering $\leq$ of G.

Remark 4. A formula is *order invariant* if its truth does not depend on the specific order. It is easy to see that a formula with quantifiers $C_{a,b}$ is equivalent to some order-invariant formula without such quantifiers. For a polynomial to be invariantly *CMSOL*-definable, it is not required that the iteration formulas are all order invariant.

Example 4. (i) The polynomials $Ind(G; X)$ and $\Omega(G; X)$, as defined in Remark 2, are *CMSOL*-definable.
(ii) For $\Phi = \Phi_{pm}(U)$ we get the matching polynomial. For $|V|$ even, its coefficient of $X^{\frac{|V|}{2}}$ is the number of perfect matchings $pm(G)$. Now $pm(G)$ can be written as

$$\sum_{U:\Phi_{pm}(U)} \prod_{x:U(x)} 1$$

which is an evaluation of a simple *CMSOL*-definable polynomial.

(iii) The polynomial $X^{k(G)}$, where $k(G)$ is the number of connected components of G, can be written as

$$\sum_{U:U=E} \prod_{x:\Phi_{fconn}(U,x)} X$$

It is an invariantly *CMSOL*-definable polynomial.

3.3 Invariantly *CMSOL*-Definable Polynomials in Normal Form

A *CMSOL*-definable polynomial in indeterminates $X_1, \ldots, X_\ell$ in *normal form* has the form

$$\sum_{U_1:\Phi_1(U_1)} \sum_{U_2:\Phi_2(U_2)} \cdots \sum_{U_{\ell_1}:\Phi_{\ell_1}(U_{\ell_1})} \left(\prod_{\bar{x}_1:\phi_1(\bar{x}_1)} X_1 \prod_{\bar{x}_2:\phi_2(\bar{x}_2)} X_2 \ldots \prod_{\bar{x}_\ell:\phi_\ell(\bar{x}_\ell)} X_\ell \right)$$

where all the formulas Φ_i and ϕ_i are *CMSOL*-formulas with the iteration variables indicated. There may be additional parameters in the formulas. However, Φ_i may not contain the variables U_j for $j > i$, and ϕ_i may not contain $\bar{x}_j$ for $j > i$. Both Φ_i and ϕ_i are referred to as iteration formulas. It is invariantly *CMSOL*-definable if its values do not depend on the order of G.

Example 5. (i) The Tutte polynomial, as defined by (1), is not given in a good way to show that it is invariantly *CMSOL*-definable. We look instead at the Tutte polynomial in its form as a partition function

$$Z(G, X, Y) = \sum_{U:U\subseteq E} X^{k[U]_G} Y^{|U|} = \sum_{U:U\subseteq E} \left(\prod_{x_1:\Phi_{fconn}(U,x_1)} X \prod_{x_2:U(x_2)} Y \right)$$

which shows that it is invariantly *CMSOL*-definable. It is related to the Tutte polynomial by the equation

$$T(G; X, Y) = (X-1)^{-k(G)} \cdot (Y-1)^{-|V(G)|} \cdot Z(G; (X-1)(Y-1), (Y-1)).$$

Another way to prove that $T(G; X, Y)$ is invariantly *CMSOL*-definable, is to use its definition via its spanning tree expansion, cf. [Bol99, Chapter 10] and [Mak05].

(ii) To see that the characteristic polynomial $P(G; X)$ is of the required form, we need a few definitions. An *elementary subgraph* of a graph G is a subgraph (not necessarily induced) which consists only of isolated vertices and cycles. If H is an elementary subgraph of G, we denote by $k(H)$ its number of connected components, and $c(H)$ the number of its cycles. With this notation we have, [Big93, Proposition 7.4], that the coefficients of $P(G; X) = \sum_i c_i X^{n-i}$ can be expressed as

$$c_i = (-1)^i \cdot \sum_{H:|V(H)|=i} (-1)^{k(H)} \cdot 2^{c(H)}$$

where the summation is over all elementary subgraphs $H = (V(H), E(H))$ of G of size i. Therefore we have

$$P(G;X) = \sum_{V(H)} \sum_{E(H)} (-1)^{|V(H)|} \cdot (-1)^{k(H)} \cdot 2^{c(H)} \cdot \prod_{v \notin V(H)} X$$

which is an evaluation of

$$\hat{P}(G;W,X,Y,Z) = \sum_{V(H),E(H)} \left(W^{|V(H)|} \cdot Y^{k(H)} \cdot Z^{c(H)} \cdot \prod_{v \notin V(H)} X \right)$$

at $W = -1, Y = -1, Z = 2$, and which is *CMSOL*-definable.

(iii) The interlace polynomial defined in [ABS04] is also a *CMSOL*-definable polynomial, but to see this one has to use the quantifier $C_{0,2}$ which says that the cardinality of a certain set is even, cf. [Cou].

3.4 Proof of Theorem 3

We are left to prove Theorem 3. Let a graph polynomial $p(G;X,Y)$ be given in normal form with two indeterminates X, Y, and two iteration formulas for summation. The general case is similar.

$$p(G;X,Y) = \sum_{U_1:\Phi_1(U_1)} \sum_{U_2:\Phi_2(U_2)} \left(\prod_{\bar{x}_1:\phi_1(U_1,U_2,\bar{x}_1)} X \prod_{\bar{x}_2:\phi_2(U_1,U_2,\bar{x}_2)} Y \right)$$

Let $X^a \cdot Y^b$ be a monomial with coefficient $c_{a,b}$. Then we have

$$c_{a,b} = \sum_{U_1:\Phi_1(U_1)} \sum_{U_2:\Phi_2(U_2)} \left(\sum_{A:(A=\emptyset)\wedge\psi_{1,a}(U_1,U_2)\wedge\psi_{2,b}(U_1,U_2)} 1 \right),$$

where $\psi_{i,c}(U_1, U_2)$ says "There are exactly c many tuples $\bar{x}$ satisfying $\phi_i(U_1, U_2, \bar{x})$". The summation over $A = \emptyset$ ensures that the last sum contains at most one summand. Clearly, this is an evaluation of an invariantly *CMSOL*−definable polynomial. □

3.5 Not Invariantly *CMSOL*-Definable Graph Polynomials

Here we give an example of a graph polynomial which is not invariantly *CMSOL*-definable.

Definition 3. *(i) A proper vertex coloring is* harmonious*, if each pair of colors appears at most once along an edge. We denote by $\chi_{harm}(G)$ the least k such that G has a harmonious proper k-coloring.*

(ii) A proper vertex coloring is complete*, if each pair of colors appears at least once along an edge. We denote by $\chi_{comp}(G)$ the largest k such that G has a complete proper k-coloring.*

(iii) Let $\chi_{harm}(G; k)$ and $\chi_{comp}(G; k)$ denote the number of harmonious, respectively complete proper k-colorings of G.

Proposition 8. *(i) $\chi_{harm}(G; k)$ is a polynomial in k.*
(ii) $\chi_{comp}(G; k)$ is not a polynomial in k.

Proof. (i) follows from [MZ06], but it is not difficult to prove it directly.
(ii) $\chi_{comp}(G; k) = 0$ for large enough k. □

Theorem 9. *$\chi_{harm}(G)$ and $\chi_{comp}(G)$ are graph parameters which are not evaluations of invariantly CMSOL-definable graph polynomials.*

Proof. $\chi_{comp}(G)$ is maximizing, so we can apply Proposition 4.

For $\chi_{harm}(G)$ we observe that, for stars S_n, a set of n edges which meet all in one single vertex, we have

$$\chi_{harm}(S_n \sqcup S_m) = \max\{\chi_{harm}(S_m), \chi_{harm}(S_n)\} + 1.$$

Now the argument proceeds like in the case a maximizing graph parameter.

Theorem 10. *$\chi_{harm}(G; k)$ is not an invariantly CMSOL-definable graph polynomial.*

Proof. Let L_i denote the graph which consists of i vertex disjoint edges. We look at $M(\chi_{harm}(G, k), 0)$ restricted to the graphs $L_i, i \in \mathbb{N}$, which we denote by $M_L(k)$ and its rank by $r_L(k)$. We note that $\chi_{harm}(L_i \sqcup L_j) = 0$ iff $i + j > \binom{k}{2}$. Therefore, $r_L(k) = \binom{k}{2}$ which is not bounded, contradicting Theorem 2.

Remark 1. *It is shown in [EM95], that computing $\chi_{harm}(G)$ is* **NP***-complete already for trees. This, together with the fact, proven in [Mak05], that evaluations of invariantly CMSOL-definable graph polynomials are polynomial time for graphs of tree-width at most k, shows that $\chi_{harm}(G; X)$ is not invariantly CMSOL-definable, unless* $\mathbf{P} = \mathbf{NP}$. *Our proof above eliminates the complexity theoretic hypothesis* $\mathbf{P} = \mathbf{NP}$.

4 Conclusions and Open Problems

We have shown that many standard graph parameters studied in the literature cannot appear as evaluations of *CMSOL*-definable graph polynomials, which covers most of the graph polynomials studied in the literature, cf. [Mak07]. We have also exhibited for the first a natural graph polynomial which is not *CMSOL*-definable.

In [FLS07] the graph parameters f which can occur as evaluations of partition functions arising from counting weighted graph homomorphisms are characterized as exactly those parameters which are multiplicative, and such that for all $k \in \mathbb{N}$ we have that $r(f, k) \leq |V|^k$ and the matrices $M(f, k)$ are positive semi-definite. It is easy to see that the partition functions are evaluations of invariantly *CMSOL*-definable graph polynomials of a very special kind.

It remains a challenging problem to characterize those graph parameters which do occur as evaluations of invariantly *CMSOL*-definable polynomials.

Acknowledgments. We wish to thank I. Averbouch, E. Fischer and L. Lovász for some comments during the preparation of this paper, and to the anonymous referees for their suggestions.

References

[ABS04] Arratia, R., Bollobás, B., Sorkin, G.B.: The interlace polynomial of a graph. Journal of Combinatorial Theory, Series B 92, 199–233 (2004)

[Big93] Biggs, N.: Algebraic Graph Theory, 2nd edn. Cambridge University Press, Cambridge (1993)

[Bol99] Bollobás, B.: Modern Graph Theory. Springer, Heidelberg (1999)

[CO00] Courcelle, B., Olariu, S.: Upper bounds to the clique–width of graphs. Discrete Applied Mathematics 101, 77–114 (2000)

[Cou] Courcelle, B.: A multivariate interlace polynomial and its computation for graphs of bouded clique-width. Electronic Journal of Combinatorics 15(1), R69 (2008)

[Die96] Diestel, R.: Graph Theory. Graduate Texts in Mathematics. Springer, Heidelberg (1996)

[EM95] Edwards, K., McDiarmid, C.: The complexity of harmonious colouring for trees. Discrete Appl. Math. 57(2-3), 133–144 (1995)

[FLS07] Freedman, M., Lovász, L., Schrijver, A.: Reflection positivity, rank connectivity, and homomorphisms of graphs. Journal of AMS 20, 37–51 (2007)

[Goo06] Goodall, A.J.: Some new evaluations of the Tutte polynomial. Journal of Combinatorial Theory, Series B 96, 207–224 (2006)

[Goo08] Goodall, A.J.: Parity, eulerian subgraphs and the Tutte polynomial. Journal of Combinatorial Theory, Series B 98(3), 599–628 (2008)

[GR01] Godsil, C., Royle, G.: Algebraic Graph Theory. Graduate Texts in Mathematics. Springer, Heidelberg (2001)

[Mak04] Makowsky, J.A.: Algorithmic uses of the Feferman-Vaught theorem. Annals of Pure and Applied Logic 126(1–3), 159–213 (2004)

[Mak05] Makowsky, J.A.: Colored Tutte polynomials and Kauffman brackets on graphs of bounded tree width. Disc. Appl. Math. 145(2), 276–290 (2005)

[Mak07] Makowsky, J.A.: From a zoo to a zoology: Towards a general theory of graph polynomials. Theory of Computing Systems (July 2007), `http://dx.doi.org/10.1017/s00224-007-9022-9`

[MZ06] Makowsky, J.A., Zilber, B.: Polynomial invariants of graphs and totally categorical theories. MODNET Preprint No. 21 (2006), `http://www.logique.jussieu.fr/modnet/Publications/Preprint%20server`

[Wel93] Welsh, D.J.A.: Complexity: Knots, Colourings and Counting. London Mathematical Society Lecture Notes Series, vol. 186. Cambridge University Press, Cambridge (1993)

Parameterized Complexity for Domination Problems on Degenerate Graphs[⋆]

Petr A. Golovach and Yngve Villanger

Department of Informatics, University of Bergen, PB 7803, 5020 Bergen, Norway
{petr.golovach,yngve.villanger}@ii.uib.no

Abstract. Domination problems are one of the classical types of problems in computer science. These problems are considered in many different versions and on different classes of graphs. We explore the boundary between fixed-parameter tractable and W-hard problems on sparse graphs. More precisely, we expand the list of domination problems which are fixed-parameter tractable(FPT) for degenerate graphs by proving that CONNECTED k-DOMINATING SET and k-DOMINATING THRESHOLD SET are FPT. From the other side we prove that there are domination problems which are difficult (W[1] or W[2]-hard) for this graph class. The PARTIAL k-DOMINATING SET and (k, r)-CENTER for $r \geq 2$ are examples of such problems. It is also remarked that domination problems become difficult for graphs of bounded average degree.

Keywords: Parameterized complexity, algorithms, degenerate graphs, domination.

1 Introduction and Overview of Results

Domination problems are fundamental and widely studied problems in algorithms and complexity theory. This paper deals with parameterized complexity of different domination problems for sparse graphs (we refer the reader to monographs [13,14] for general information on parameterized complexity and algorithms). It is well known that these problems are W[1]-complete (like k-PERFECT CODE [6,12]) or W[2]-complete (like k-DOMINATING SET [11]) for general graphs. It is then natural to investigate complexity of these problems for restricted graph classes. By now there is an extensive literature about parameterized complexity of domination problems for different families of graphs.

Most versions of domination problems becomes fixed-parameter tractable (FPT) for planar graphs. The first such result was established by Alber et al. [1]. Later, these results were generalized for other families of sparse graphs. The newly developed theory of bidimensionality (see e.g. survey [10]) was used for construction of fixed-parameter algorithms for broad graph classes. By using this theory it is possible to construct such algorithms for different domination problems on apex-minor-free graphs. Moreover, these results can be extended

⋆ Supported by Norwegian Research Council.

to even larger classes of graphs. Particularly, it was proved by Demaine et al. that the k-Dominating set problem is FPT for H-minor-free graphs [9], (k, r)-Center if FPT for map graphs [8] and apex-minor-free graphs. By developing these ideas and applying new branching techniques Amini et al. proved that Partial k-Dominating set, Partial weighted k-Dominating set and Partial (k, r)-center are FPT for H-minor-free graphs [3]. Even more general results for probelms which can be expressed in the first-order logic were received by Dawar et al. [7].

A graph G is called *d-degenerate* (with d being a positive integer) if every induced subgraph of G has a vertex of degree at most d. For example, trees are exactly connected 1-degenerate graphs, and every planar graph is 5-degenerate. Moreover, it is known that all H-minor-free graphs are degenerate (see [18,19,20]).

An ordering of vertices of a graph G $v_1, v_2, \ldots, v_n$ is called a *d-degenerate ordering* if every vertex v_i has at most d neighbors among the vertices $v_1, v_2, \ldots, v_{i-1}$. It is well known that a graph is d-degenerate if and only if it allows a d-degenerate ordering of its vertices. By considering vertices in the backward d-degenerate ordering and using the method of bounded search trees it can be proved that k-Perfect code [5] is FPT for degenerate graphs. Similar fact can be easily established for Independent k-dominating set.

In [2] Alon and Gutner proved that k-Dominating set is FPT for d-degenerate graphs. They used the method of bounded search trees and the fact that degenerate graphs have bounded average degree. They also have shown that Weighted k-dominating set is FPT for degenerate graphs. Using same techniques it can be easily proved that some other domination problems (for example, Roman k-dominating set) are also FPT for this family of graphs.

All these results lead us to the following question: which domination problems are FPT for degenerate graphs, and which problems are W[1] or W[2]-hard?

Our results. Building on the ideas of Alon and Gutner we prove that it is possible to construct FPT-algorithms for some other domination problems with additional restrictions. Particularly, we prove that Connected k-dominating set is FPT for degenerate graphs. Because of the additional restrictions (which is connectivity of the dominating set in this case) it is impossible to apply the results of [2] directly. By using a similar approach we also show that k-dominating threshold set is FPT. Next we prove that there are domination problems which are more difficult for degenerate graphs. For instance we show that Partial k-dominating set is W[1]-hard, and (k, r)-Center is W[2]-hard for this class. We conclude our paper by the observation that domination problems become difficult for graphs of bounded average degree.

2 Preliminaries

We consider finite undirected graphs without loops or multiple edges. The vertex set of a graph G is denoted by $V(G)$ and its edge set by $E(G)$ (or simply V and E if it does not create a confusion). For $S \subseteq V(G)$ we denote by $G[S]$ the subgraph of G induced by S. A set $S \subseteq V(G)$ is called *connected* if $G[S]$ is a connected graph.

The *open neighborhood* of a vertex is denoted by $N_G(v) = \{x\colon xv \in E(G)\}$. For a positive integer r we define *closed r-neighborhood* of a vertex v, denoted $N^r_G[v]$, to be the set of vertices of G at distance at most r from v. If $U \subset V(G)$, then $N^r_G[U]$ denotes the set $\bigcup\limits_{v\in U} N^r_G[v]$. We use notations $N_G[v]$ and $N_G[U]$ for $r = 1$. Degree of a vertex v is denoted by $\deg_G v$, $\Delta(G) = \max\{deg_G v\colon v \in V(G)\}$. We omit a subscript G if it does not create a confusion. The *average degree* of G is defined as $\frac{1}{|V(G)|} \sum\limits_{v\in V(G)} \deg v = \frac{2|E(G)|}{|V(G)|}$. It is known (end easy to see) that every d-degenerate graph has the average degree no more than $2d$.

It is said that a vertex v is *dominated* by the vertex u if $v \in N[u]$, and it is said that the vertex v is *dominated* by set $S \subseteq V(G)$ if $v \in N[S]$. Recall that a set $S \subseteq V(G)$ is called a *dominating set* if every vertex of G is dominated by S.

The k-Dominating set is the following problem:

INSTANCE: A graph G.
PARAMETER: A positive integer k.
QUESTION: Is there a dominating set $S \subseteq V(G)$ such that $|S| \le k$?

Recently Alon and Gutner [2] proved k-Dominating set to be FPT on d-degenerate graphs by proving the proposition below. This proposition will be required in both the FPT algorithms presented in the next sections. In order to apply the proposition the problem is rephrased in terms of black and white vertices. It is supposed that the vertex set V is partitioned into two sets W (white vertices) and B (black vertices). The goal is to find a set $S \subset W \cup B$ which dominates B.

Proposition 1. *([2]) Let $G = (B \cup W, E)$ be a d-degenerate black and white graph. If $|B| > (4d + 2)r$, then there are at most $(4d + 2)r$ vertices in G that dominate at least $|B|/r$ vertices of B.*

This proposition enable us to apply the method of bounded search trees for the k-Dominating set problem.

3 FPT Algorithms on d-Degenerate Graphs

3.1 Connected Domination

This subsection deals with the problem of finding a connected dominating set of size at most k in a d-degenerate graph which we call Connected k-Dominating set. Formally we define the problem as follows:

INSTANCE: A d-degenerate graph G.
PARAMETER: Positive integers d and k.
QUESTION: Does there exist a dominating set $S \subseteq V$ such that $G[S]$ is connected and $|S| \le k$?

It is well known that finding a connected dominating set of size k is $W[2]$ hard on general graphs (see e.g. [13]). The rest of this section presents an FPT algorithm for this problem.

We assume without loss of generality that G is a connected graph with n vertices and $n \geq k$. Like in the paper by Alon and Gutner [2] a new problem instance based on black and white vertices is created. Let (S, W, B, q) be a problem instance for the graph G, where S contains vertices that must be contained in the dominating set, the set $W \subseteq V \setminus S$ (white vertices) contains vertices that are dominated by S, the set B (black vertices) contains the undominated vertices $V \setminus (S \cup W)$, and $q = k - |S|$. Our goal is to decide if q vertices from $W \cup B$ can be added to S, such that S becomes a connected dominating vertex set of size k. The initial problem instance for a graph G will be $(S = \emptyset, W = \emptyset, B = V, q = k)$. Each time a vertex in $W \cup B$ is moved to S the parameter q will be reduced by one. From the initial problem instance, we will grow a tree of problem instances, where the final leafs either contains a solution, or claims that the choices made on the path from the root to the leaf is not consistent with any solution.

Connected or not, every vertex set of cardinality q in $W \cup B$ that dominates B, contains a vertex that dominates at least $|B|/q$ of the vertices in B. By Proposition 1, $|B| \leq (4d+2)q$ or there exists at most $(4d+2)q$ vertices in $W \cup B$ that dominate at least $|B|/q$ vertices in B. As long as $|B| > (4d+2)q$, add a leaf $(S \cup \{u\}, W \cup (N(u) \cap B) \setminus \{u\}, B \setminus N[u], q-1)$ to the problem instance (S, W, B, q) for every vertex $u \in W \cup B$ where $|N[u] \cap B| \geq |B|/q$. By Proposition 1 we can now conclude that problem instance (S, W, B, q) contains a solution if and only if one of the added leafs contains a solution. Thus, the initial problem instance (S, W, B, q) can be ignored from this point on. This process of reducing the size of the problem instance by creating several new problem instances where one of them contains a solution if and only if the initial problem instance contained a solution will be referred to as *branching*. The number of new problem instances is called the degree of the branching.

Like the algorithm for dominating set on d-degenerate graphs [2] we branch on the at most $(4d+2)q$ vertices that dominates $|B|/q$ vertices in B, until $q = 0$ or $|B| \leq (4d+2)q$. If $q = 0$ and S is a connected dominating set, return the vertex set S as the solution, otherwise mark the problem instance with no solution.

Let us now consider the case, where $q \geq 1$ and $|B| \leq (4d+2)q$. Denote by $v_1, v_2, ..., v_q$ the vertices of the connected dominating set, which are not contained in S. Clearly, the subgraph of G induced by vertices from S and $v_1, v_2, ..., v_q$ must have some spanning tree T. Label the vertices of S and $v_1, v_2, ..., v_q$ in any order, this makes T into a labeled spanning tree. We will now extend the our problem instance by adding a labeled tree containing k vertices. By Cayley's theorem the number of labeled spanning trees containing k vertices is k^{k-2}. Add k^{k-2} leafs to the problem instance (S, W, B, q), one for each possible labeled tree T over k vertices. Notice that one of these will be equal to the labeled tree obtained by the spanning tree of a connected dominating set and the labelling assigned to S and $v_1, v_2, ..., v_q$. From now on a problem instance (S, W, B, q, T) is considered.

In any connected dominating set of size k, the neighborhood of v_i in $v_1, v_2, ..., v_q$ will define a subset V_i of B which is dominated by v_i. Branch on the no more than $q^{(4d+2)q}$ different ways of putting the vertices of B into the q vertex sets $V_1, V_2, ..., V_q$ (some sets can be empty).

Now a dynamic programming algorithm is used to find vertices $v_1, v_2, \ldots, v_q$ or prove that there are no such vertices. Choose a vertex z of T as a root of the tree. This induces a parent-child relation in the tree, which defines the set of leafs. For every vertex $x \in V(T)$ a vertex set $U(x)$ is defined. For every leaf x of T, the vertex set $U(x) = \{x\}$ if $x \in S$ and $U(x) = \{v \in W \cup B \colon V_i \subseteq N_G[v]\}$ if $x = v_i$ for $i \in \{1, 2, ..., q\}$. Assume now that x in T has children $y_1, y_2, \ldots, y_t$ such that $U(y_i)$ is defined for $i \in \{1, 2, \ldots, t\}$. There are two cases, let us first assume that $x \in S$. If $N_G[x] \cap U(y_j) \neq \emptyset$ for all $j \in \{1, 2, \ldots, t\}$ then $U(x) = \{x\}$, otherwise $U(x) = \emptyset$. Let us now assume that $x = v_i$ for $i \in \{1, 2, ..., q\}$. Then, $U(x) = \{v \in W \cup B \colon V_i \subseteq N_G[v]$ and $N_G[v] \cap U(y_j) \neq \emptyset$ for all $j \in \{1, 2, \ldots, t\}\}$. If $U(z) = \emptyset$ then the considered problem instance has no solution, and if $U(z) \neq \emptyset$ then by the dynamic programming it is easy to chose vertices from $U(v_1), U(v_2), \ldots, U(v_q)$ so that they together with vertices of S compose a connected dominating set of cardinality no more than k. Note that it is possible that same vertices are chosen from different sets $U(v_i)$, but it only means that we have a connected dominating set of lesser cardinality.

The properties of our algorithm are summarized in the following theorem.

Theorem 1. *The described algorithm decides in $O(k^{O(dk)} \cdot n^{O(1)})$ time if a d-degenerate graph contains a connected dominating set of size k.*

Proof. The correctness of the algorithm follows from the description above. From the initial problem instance the algorithm above create a branching tree, which has the following properties. For every internal vertex of the branching tree, there exists a child that has a solution if and only if the parent contains a solution. For every leaf instance, we can decide in $O(n^{O(1)})$ time if there exists a solution to the instance, and every problem instance is created in $O(n^{O(1)})$ time.

It remains to bound the number of instances in the branching tree, let us count them. At most $((4d+2)k)^k$ problem instances are created when branching on vertices that dominates at least $|B|/k$ of the vertices in B. There exists k^{k-2} trees containing k vertices, and these trees can be listed with complexity $O(k^{k-1})$ [15]. At most $k^{(4d+2)k}$ problem instances are created when distributing vertices of B into the vertex sets $V_1, V_2, ..., V_k$. Total number of created problem instances is obtained by multiplying these numbers, which give the total $O(k^{O(dk)})$. Thus, the running time is $O(k^{O(dk)} \cdot n^{O(1)})$.

3.2 Dominating Threshold Set

For a given graph $G = (V, E)$, and integers k and r, a vertex set $S \subset V$ is a *dominating threshold set* if the closed neighborhood $N[v]$ contains at least r

vertices in S for every vertex $v \in V$. Formally the k-Dominating threshold set problem is defined as follows:

INSTANCE: A graph G and a positive integer r.
PARAMETER: A positive integer k.
QUESTION: Is there a set $S \subseteq V(G)$ such that $|S| \leq k$ and for every vertex $u \in V(G)$ $|N[u] \cap S| \geq r$?

This problem can be solved on d-degenerate graphs in a similar way as the Connected k-dominating set problem was solved on d-degenerate graphs.

We assume that $n = |V|$ and $k \leq n$. If $r > d+1$ or $r > k$ then the graph has no dominating threshold set. So we suppose that $r \leq min(d+1, k)$.

Let us rephrase the problem in terms of black and white vertices. Consider the problem instance (S, W, B, q), where S are vertices in the dominating threshold set, W are vertices that are dominated by at least r vertices in S, $B = V \setminus W$, and $q = k - |S|$. Notice that even vertices in S has to be dominated by r vertices, so adding a vertex v to S, do not enable us to remove it from B. The initial problem instance will be $(S = \emptyset, W = \emptyset, B = V, q = k)$.

Dominating threshold set are also dominating sets, so Proposition 1 applies here as well. Thus, $|B| \leq (4d+2)q$ or there exists at most $(4d+2)q$ vertices in $W \cup B$ that dominates at least $|B|/q$ of the vertices in B. While $|B| \geq (4d+2)q$ we branch and create one new problem instance $(S \cup \{u\}, W \cup U, B \setminus U, q-1)$ for each vertex $u \in (W \cup B) \setminus S$ that dominates at least $|B|/q$ vertices in B, where U is the set of vertices in $N[u] \cap B$ that are dominated by $r-1$ vertices in S. Repeat this branching until $q = 0$ or $|B| \leq (4d+2)q$. If $q = 0$ and $B = \emptyset$ return S as the solution, otherwise if $B \neq \emptyset$ then mark the problem instance with no solution.

Consider now the case where $|B| \leq (4d+2)q$. It is not enough to find a dominating set of B in this case since every vertex requires r neighbors in S. Like the connected domination set problem we define the vertex sets $V_1, V_2, ..., V_q \subseteq B$, but this time every vertex of B can be added to several sets. The reason for this is that it might be missing more than one dominator.

Now, branch on the at most $q^{(4d+2)qr}$ different ways of adding the at most $(4d+2)q$ vertices of B to the vertex sets $V_1, V_2, ..., V_q$ in such a way that for every vertex $v \in B$ $|\{i \colon v \in V_i\}| + |\{u \in S \colon v \in N[u]\}| \geq r$. If there are no such sets then the problem instance has no solution. Otherwise for every new instance we are trying to add to our set S vertices $v_1, v_2, \ldots, v_q$ such that v_i dominates exactly set V_i.

Define $U_i = \{w \colon w \in (W \cup B) \setminus S \text{ and } V_i = N[w] \cap B\}$ for $i \in \{1, 2, \ldots, q\}$. Clearly there is no solution if some vertex set U_i is empty. Also if $V_i \neq V_j$ then $U_i \cap U_j = \emptyset$. Let $s_i = |\{j \colon V_j = V_i\}|$ for every $i \in \{1, 2, \ldots, q\}$. If $|U_i| < s_i$ for some $i \in \{1, 2, \ldots, q\}$ then the problem instance has no solution. Otherwise we consider all pairwise different sets U_i. From every such set U_i we chose s_i different vertices and add them to the set S.

Theorem 2. *The described algorithm decides in $O(k^{O(dkr)} \cdot n^{O(1)})$ time if a d-degenerate graph contains a dominating r-threshold set of size k.*

Proof. The correctness of the algorithm follows from the description above. From the initial problem instance the algorithm create a branching tree, which has the following properties. For every internal vertex of the branching tree, there exists a child that has a solution if and only if the parent contains a solution. For every leaf instance, we can decide in $O(n^{O(1)})$ time if there exists a solution to the instance, and every instance is created in $O(n^{O(1)})$ time.

It remains to bound the number of instances in the branching tree. At most $((4d+2)k)^k$ problem instances are created when branching on vertices that dominates at least $|B|/k$ of the vertices in B. There exists no more than $k^{(4d+2)kr} = k^{O(dkr)}$ different ways of adding vertices from B to the vertex sets $V_1, V_2, ..., V_q$. The total number of created problem instances is then $O(k^{O(dkr)})$. Thus, the running time is $O(k^{O(dkr)} \cdot n^{O(1)})$.

4 Partial Domination

Here we consider a variant of domination problem, in which it is not necessary to dominate all vertices of a graph, but at least the given number of vertices. The Partial k-dominating set problem is formulated as follows:

INSTANCE: A graph G and a positive integer N.
PARAMETER: A positive integer k.
QUESTION: Is there a set $S \subseteq V(G)$ such that $|S| \leq k$ and which dominates at least N vertices?

It can be easily seen that this problem is W[2]-complete on general graphs(if $N = |V(G)|$ then Partial k-dominating set is the k-Dominating set problem). Note that here N is a part of the instance, but is not a parameter of the problem. If N is supposed to be a parameter of the problem then it is FPT [17]. Recall that Amini et al. proved that Partial k-dominating set is FPT for H-minor-free graphs [3]. We prove that Partial k-dominating set is difficult for degenerate graphs.

Theorem 3. Partial k-dominating set *is W[1]-hard for 2-degenerate graphs.*

Proof. We reduce the k-Perfect code problem. A *perfect code* in a graph G is a set of vertices $S \subseteq V(G)$ with the property that for every vertex $v \in V(G)$, there is exactly one vertex from S in $N[v]$. The k-Perfect code is the following problem:

INSTANCE: A graph G.
PARAMETER: A positive integer k.
QUESTION: Is there a perfect code $S \subseteq V(G)$ of size k?

It is known [6,12] that this problem is W[1]-complete.

Let G be a graph with n vertices and m edges. It can be assumed without loss of generality that this graph is connected and has at least 2 vertices. We construct graph G' starting with the vertex set $V = V(G)$. Let $t = n^3$ and

$r = \Delta(G)^2 t$. If vertices $u, v \in V$ are adjacent in G or are at distance 2 in this graph then u and v are joined by t paths of length two. Then for every vertex $v \in V$ add $r - t(|N_G^2[v]| - 1)$ adjacent pendant vertices. Denote by U the set of vertices of degree 1 or 2 which were included to the vertex set of G' at this stage of our construction. Now for every vertex $v \in V$ we execute the following operation: for every two different vertices $x, y \in N_G[v]$ a x, y-path of length two is added to G'. Denote by $W(v)$ the set of vertices of degree 2 which were added during this operation for the vertex v, and let $W = \bigcup_{v \in V} W(v)$. Note that some vertices can be joined by several paths after these operations for all vertices of V, but since any two different vertices of V can belong to closed neighborhoods of no more than n vertices, the number of such paths is no more than n. Then $|W| \leq \frac{n^2(n-1)}{2}$. Clearly G' is 2-degenerate, and our construction of G' is polynomial. Now we define $N = (r + 1)k + 2m$.

Suppose that $S \subseteq V(G)$ is a perfect code in G. It can be easily seen that $N_{G'}[S] \cap V = S$. Since S is a perfect code, vertices of S are at distance at least 3 in the graph G. It follows immediately that $|N_{G'}[S] \cap U| = kr$. For every vertex $v \in V$ exactly one vertex $x \in N_G[v]$ belongs to S. Then $N_{G'}[S] \cap W(v) = N_{G'}(x) \cap W(v)$, and $|N_{G'}(x) \cap W(v)| = \deg_G v$. So $|N_{G'}[S] \cap W| = \sum_{v \in V} \deg_G v = 2m$. Now $|N_{G'}[S]| = |N_{G'}[S] \cap V| + |N_{G'}[S] \cap U| + |N_{G'}[S] \cap W| = (1 + r)k + 2m = N$.

Assume now that $S \subseteq V(G')$, $|S| \leq k$ and $|N_{G'}[S]| \geq N$. Suppose that $|S| < k$. Then $|N_{G'}[S]| = |N_{G'}[S] \cap (V \cup U)| + |N_{G'}[S] \cap W| \leq |S|(r+1) + |W| \leq |S|(r + 1) + \frac{n^2(n-1)}{2} < (|S| + 1)(r + 1) \leq N$. So $|S| = k$. If the set S contains a vertex from $U \cup W$ then $|N_{G'}[S]| = |N_{G'}[S] \cap (V \cup U)| + |N_{G'}[S] \cap W| \leq (k - 1)(r + 1) + |W| + 3 \leq (k - 1)(r + 1) + \frac{n^2(n-1)}{2} + 3 < k(r + 1) \leq N$. This contradiction means that $S \subseteq V$. Suppose that S contains vertices that are adjacent or 2-distant in G. In this case $|N_{G'}[S]| = |N_{G'}[S] \cap (V \cup U)| + |N_{G'}[S] \cap W| \leq k(r + 1) - t + |W| \leq k(r + 1) + \frac{n^2(n-1)}{2} - t < k(r + 1) \leq N$, and we conclude that for every $v \in V$ $N_G[v]$ contains no more than one vertex from S. If there is a vertex $v \in V$ such that there are no vertices from S in $N_G[v]$ then $|N_{G'}[S]| = |N_{G'}[S] \cap (V \cup U)| + |N_{G'}[S] \cap W| \leq k(r + 1) + 2m - \deg_G v < N$. It follows immediately that S is a perfect code of the size k in G.

5 (k, r)-Center Problem

The (k, r)-Center (see e.g. [4] for the background of this problem) is another example of domination problem which becomes difficult for degenerate graphs. Let r be a positive integer. The set $S \subseteq V(G)$ is called a *r-center* if $N^r[S] = V(G)$. The (k, r)-Center is the following problem:

INSTANCE: A graph G.
PARAMETER: Positive integers k and r.
QUESTION: Is there a r-center $S \subseteq V(G)$ such that $|S| \leq k$?

Clearly k-Dominating set is a special case of this problem for $r = 1$, hence (k, r)-Center is W[2]-hard for general graphs. We prove that for $r \geq 2$ results by Amini et al. [3] can not be extended for degenerate graphs if FPT $\neq$ W[2].

Theorem 4. *For any $r \geq 2$ the (k, r)-Center is W[2]-hard for 2-degenerate graphs.*

Proof. We reduce k-Dominating set problem. Let G be a connected nonempty graph, and $V = V(G)$. Every edge of G is replaced by the path of length r. We call a vertex x of such a path the *central* vertex if it is at distance $\lfloor \frac{r}{2} \rfloor$ from one of endpoints (if r is even every path has one central vertex, and there are two central vertices if r is odd). Then a new vertex u is introduced and joined by paths of length $\lceil \frac{r}{2} \rceil + 1$ with all central vertices. At the final stage of the construction a vertex v is added and joined with u by the path P of length r. Denote the obtained graph by G'. Clearly G' is 2-degenerate.

We prove that G has a dominating set of a size at most k if and only if G has a r-center of a size at most $k + 1$. Suppose that S is the dominating set in G and $|S| \leq k$. It can be easily seen that $S \cup \{u\}$ is a r-center of G' and $|S| \leq k + 1$. Assume now that S' is a r-center of G' and $|S'| \leq k + 1$. At least one vertex of the path P is included to S'. Without loss of generality we can assume that u is a unique vertex of this path which belongs to S'. Note that $V(G') \setminus V = N^r_{G'}[u]$. Let $S = S' \setminus \{u\}$. Suppose that there is a vertex $x \in S$ such that $x \notin V$. Then either x is a vertex of the path which replaced some edge $ab \in E(G)$ or it belongs to the path which connects u with the central vertex of such a path. Only vertices a and b in V are at distance at most r from x. Then we can replace x in S by a or b. It means that we can assume that $S \subseteq V$. It can be easily seen that S is a dominating set of G, and $|S| \leq k$.

6 Domination Problems for Graphs of Bounded Average Degree

It is known that some graph covering problems (like k-Independent set) are FPT for graphs of bounded average degree, but it can be simply proved that domination problems are W[1] or W[2]-hard for this class.

Proposition 2. *The k-Dominating set problem is W[2]-hard for graph of bounded average degree.*

Proof. We reduce k-Dominating set for general graphs. Let G be a graph with n vertices and m edges. Define G' as a union of G and a star $K_{1,r}$ for $r = n^2$. The average degree of G' is equal to $\frac{2m+2r}{n+r+1} \leq \frac{3n^2}{n^2} = 3$, i.e. this graph has bounded average degree. It can be easily seen that G has a dominating set of a size k if and only G' has a $k + 1$-element dominating set.

By same reduction it can be easily proved that k-Perfect code is W[1]-hard for graphs of bounded average degree and Independent k-dominating set is W[2]-hard for this class. For connected dominating set reduction is slightly different.

Proposition 3. *The* Connected k-dominating set *problem is W[2]-hard for graph of bounded average degree.*

Proof. We reduce k-Dominating set for general graphs. Let G be a graph with n vertices and m edges. Suppose that $V(G) = \{v_1, v_2, \ldots, v_n\}$. We introduce two copies of the set $V(G)$: $U = \{u_1, u_2, \ldots, u_n\}$ and $W = \{w_1, w_2, \ldots, w_n\}$. Vertices u_i and w_j are joined by an edge if $i = j$ or $v_iv_j \in E(G)$. Then a new vertex z is added and joined by edges with all vertices of W. At the final stage of the construction n^2 pendant vertices adjacent to z are added. Denote the obtained graph G'. The average degree of G' is equal to $\frac{4m+2n+2n^2}{2n+1+n^2} \leq 4$, i.e. this graph has bounded average degree. It can be easily seen that G has a dominating set of a size k if and only G' has a $k+1$-element connected dominating set. If S is a dominating set in G then $S' = \{w_i : v_i \in S\} \cup \{z\}$ is a connected dominating set in G'. Suppose that S' is a connected dominating set in G' of size at most $k + 1$. Clearly, $z \in S'$, and we can assume that all other vertices of this set belong to $U \cup W$. If some vertex $u_i \in S'$ then it can be replaced by w_i in our dominating set. So we can also assume that $S' \setminus \{z\} \subseteq W$. We have only note that $S = \{v_i \in V(G) : w_i \in S' \setminus \{z\}\}$ is a dominating set in G.

7 Conclusion

We proved that the k-domination problem remains FPT for degenerate graphs, even if additional restrictions like connectivity or a threshold boundary is added. On the other side the k-domination problem becomes W[1] or W[2]-hard on degenerate graphs, when a partial or r-center domination is required. It could be interesting to obtain a sharper boundary between the FPT and W-hardness for different classes of sparse graphs. For example, it easily follows from the results of [3] that the Partial k-vertex cover problem is FPT for degenerate graphs, but this problem is W[1]-complete for general graphs [16]. By using the same reduction as in Theorem 2, the Partial k-vertex cover is W[1]-hard for graphs of bounded average degree. At the same time it is well known that the k-Independent set which is W[1]-hard for general graphs is FPT for this class. Another interesting problem is a construction of more efficient FPT-algorithms for domination problems on d-degenerate graphs.

References

1. Alber, J., Bodlaender, H.L., Fernau, H., Kloks, T., Niedermeier, R.: Fixed parameter algorithms for dominating set and related problems on planar graphs. Algorithmica 33, 461–493 (2002)
2. Alon, N., Gutner, S.: Linear time algorithms for finding a dominating set of fixed size in degenerated graphs. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, pp. 394–405. Springer, Heidelberg (2007)
3. Amini, O., Fomin, F.V., Saurabh, S.: Parameterized algorithms for partial cover problems (submitted, 2008)

4. Bar-Ilan, J., Kortsarz, G., Peleg, D.: How to allocate network centers. J. Algorithms 15, 385–415 (1993)
5. Cai, L., Kloks, T.: Parameterized tractability of some (efficient) Y -domination variants for planar graphs and t-degenerate graphs. In: International Computer Symposium (ICS), Taiwan (2000)
6. Cesati, M.: Perfect code is W[1]-complete. Inform. Process. Lett. 81, 163–168 (2002)
7. Dawar, A., Grohe, M., Kreutzer, S.: Locally excluding a minor. In: LICS, pp. 270–279. IEEE Computer Society, Los Alamitos (2007)
8. Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Fixed-parameter algorithms for (k,r)-center in planar graphs and map graphs, ACM Trans. Algorithms 1, 33–47 (2005)
9. Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. J. ACM 52, 866–893 (2005) (electronic)
10. Demaine, E.D., Hajiaghayi1, M.T.: The bidimensionality theory and its algorithmic applications. The Computer Journal (2007)
11. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness. I. Basic results. SIAM J. Comput., 24, 873–921 (1995)
12. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness. II. On completeness for W[1]. Theoret. Comput. Sci. 141, 109–131 (1995)
13. Downey, R.G., Fellows, M.R.: Parameterized complexity, Monographs in Computer Science. Springer, New York (1999)
14. Flum, J., Grohe, M.: Parameterized complexity theory, Texts in Theoretical Computer Science. EATCS Series. Springer, Berlin (2006)
15. Gabow, H.N., Myers, E.W.: Finding all spanning trees of directed and undirected graphs. SIAM J. Comput. 7, 280–287 (1978)
16. Guo, J., Niedermeier, R., Wernicke, S.: Parameterized complexity of vertex cover variants. Theory Comput. Syst. 41, 501–520 (2007)
17. Kneis, J., Mölle, D., Rossmanith, P.: Partial vs. complete domination: t-dominating set. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 367–376. Springer, Heidelberg (2007)
18. Kostochka, A.V.: The minimum Hadwiger number for graphs with a given mean degree of vertices, Metody Diskret., Analiz, pp. 37–58 (1982)
19. Thomason, A.: An extremal function for contractions of graphs. Math. Proc. Cambridge Philos. Soc. 95, 261–265 (1984)
20. Thomason, A.: The extremal function for complete minors. J. Combin. Theory Ser. B 81, 318–338 (2001)

An Algorithm for Finding Input-Output Constrained Convex Sets in an Acyclic Digraph

Gregory Gutin, Adrian Johnstone, Joseph Reddington, Elizabeth Scott, and Anders Yeo

Department of Computer Science, Royal Holloway, University of London
Egham, Surrey, TW20 0EX, UK

Abstract. A set X of vertices of an acyclic graph is convex if any vertex on a directed path between elements of X is itself in X. We construct an algorithm for generating all input-output constrained convex (IOCC) sets in an acyclic digraph, which uses several novel ideas. We show that our algorithm is more efficient than algorithms described in the literature in both the worst case and computational experiments. IOCC sets of acyclic digraphs are of interest in the area of modern embedded processor technology.

1 Introduction

In this paper we consider an algorithm for generating all input-output constrained convex sets in an acyclic digraph N. There is an immediate application for this algorithm in the field of embedded systems design. One of the major design choices for any new processor is the selection of the machine instruction set. In an embedded system, the processor will only execute a single fixed program during its lifetime, and significant efficiency gains can be made by choosing the machine instruction set, and associated hardware, to support the program that will be executed.

In particular there exist *extensible* general purpose processors such as the ARM OptimoDE, the MIPS Pro Series and the Tensilica Xtensa that can be customized for specific applications by the addition of custom-designed machine instructions and supporting hardware. The approach is to choose a set of application specific machine instructions by examination of the target program; candidate instructions are likely to involve the combination of several basic computations. For example, a program solving simultaneous linear equations may find it useful to have a single instruction to perform matrix inversion on a set of values held in registers.

Candidate instruction identification is carried out on *data dependency graphs* (DDGs), which are obtained from the application program by first splitting it into *basic blocks*, regions of sequential computation with no control transfer into their bodies, and then creating vertices for each instruction. There is an arc to each vertex u from those vertices whose instructions compute input operands of u. The resulting DDGs are acyclic and any convex subset of vertices is a candidate for a custom instruction which could be implemented in hardware. (A vertex set X is *convex* if it has the property that any vertex which lies on a path between vertices in X is itself in X, and convexity ensures that all of the inputs for the proposed instruction are available at the start of the instruction execution.)

We have given [4] an algorithm which efficiently finds all the connected convex vertex sets of an acyclic digraph N. However, in practice a given hardware application will have specific, and usually small, input and output constraints. This significantly reduces the size of the solution space and thus presents an opportunity for a more efficient enumeration algorithm. Furthermore, certain instructions, such as writes to main memory, cannot be combined into a custom instruction, thus certain vertices in the acyclic digraph can be designated as *forbidden* from the point of view of inclusion in a candidate set. Thus we are interested in finding all convex sets which have specified bounds, n_{in} and n_{out}, on the numbers of input and output vertices and which do not contain any vertices from a specified forbidden set F. For a convex set S, a vertex $i \in V(N) - S$ ($o \in S$) is called an *input vertex* (*output vertex*) if there is an arc from i to a vertex in S (there is an arc from o to a vertex not in S).

Bonzini and Pozzi [1] and Chen, Maskell and Sun [2] proved that with the two constraints above there are only polynomial number, $O(n^{n_{in}+n_{out}})$, of valid convex sets in an acyclic digraph N with n vertices provided n_{in} and n_{out} are constants (as they are in practice). The algorithm given in [1], the BP algorithm, has running time $O(n^{n_{in}+n_{out}+1})$. For an acyclic digraph N with unique source s (which is a vertex of in-degree zero) and a vertex set Q, a vertex set C is a *generalized dominator of* Q if each path from s to Q passes through a vertex in C, and for each vertex $c \in C$ there is a path from s to Q which contains only c and no other members of C. It was observed in [1] that if C is a generalized dominator of B in N then there is a convex set S in N with the set of input vertices C and the set of output vertices containing B. However, the converse it not true and, as a result, the BP algorithm does not generate all valid convex sets (in our experiments up to 25% of all valid convex sets were not generated by the BP algorithm); for a more detailed discussion, see [6].

Moreover, the BP algorithm is efficient only when the number $c(N)$ of valid convex sets in N is close to $\Theta(n^{n_{in}+n_{out}})$. In practice many acyclic digraphs N have significantly fewer valid convex sets. In such cases our valid convex set generation algorithm $\mathcal{A}$ described below, which is of time complexity $O(m \cdot n_{in}^2(c(N) + n^{n_{out}}) + m)$, is significantly faster (m is the number of arcs in N) than the BP algorithm. More importantly, $\mathcal{A}$ generates *all* valid convex sets.

In computational experiments, we have compared $\mathcal{A}$ with the state-of-the-art algorithm of Chen, Maskell and Sun [2] (CMS algorithm) and the well-known algorithm of Atasu, Pozzi and Ienne [5] (API algorithm). Our experiments clearly demonstrate that $\mathcal{A}$ is significantly faster than both the CMS and API algorithms.

For more information on modern embedded processors technology and convex set generating algorithms, see, e.g. [1,2,3,4,5].

In what follows, N denotes the acyclic digraph under consideration and F_0 is the initial set of forbidden vertices. By adding extra vertices to N and F_0 if necessary, without loss of generality we shall assume that N has a unique vertex s (*source*) with in-degree zero and a unique vertex t (*sink*) with out-degree zero. We assume that $s, t \in F_0$. Thus, every vertex lies on a directed path between two elements of F_0.

For a fixed pair n_{in}, n_{out} of positive integers and a set F of forbidden vertices, we say that a convex set S is *valid* if $S \cap F = \emptyset$ and the numbers of its input and output vertices are at most n_{in} and n_{out}, respectively. For vertex sets Y, Z, an arc yz with $y \in Y$ and $z \in Z$ is called an (Y, Z)-*arc* and a path (walk) starting in a vertex of Y and terminating in a vertex of Z is called a (Y, Z)-*path* ((Y, Z)-*walk*). In this paper, all walks (and, thus, paths and cycles) are directed.

2 Preliminary Results

Let O be an arbitrary set of vertices such that $|O| \le n_{out}$ and the following holds: for every vertex $o \in O$ there is an (o, F)-path in $N - (O - \{o\})$. The condition guarantees that there is no convex set containing O with the output set $O' \subset O$.

Definition 1. *For a set Y of vertices in a digraph D, the* convex closure Y^{cl}_D *(or just Y^{cl} if D is clear from the context) is defined as $Y^{cl}_D = \{u \mid \exists (u, Y) - path \ \& \ (Y, u) - path\}$. That is, Y^{cl}_D contains all vertices with a path in D into Y and a path in D from Y. In particular $Y \subseteq Y^{cl}$.*

Let X and F be arbitrary sets of vertices in N, such that $O^{cl}_N \subseteq X$, $F_0 \subseteq F$ and $X \cap F = \emptyset$. We will give a recursive algorithm that finds all convex sets S with O as the output vertices, with at most n_{in} input vertices and with $X \subseteq S \subseteq V(N) - F$. However, before doing this we need the following definitions and lemmas.

Definition 2. *Given the above definitions, let N^* be obtained from N by deleting all arcs out of the vertices in O. Let D be obtained from N^* by coloring every arc $xy \in A(N^*)$ red and adding the blue-colored arc yx.*

Given a multiset $\mathcal{B}$ of arcs in D, let $D_{\mathcal{B}}$ denote the directed multigraph with $V(D_{\mathcal{B}}) = V(D)$ and $A(D_{\mathcal{B}}) = \mathcal{B}$. Note that $\mathcal{B}$ may contain several copies of the same arc in D and $D_{\mathcal{B}}$ may therefore contain parallel arcs.

Definition 3. *A multiset $\mathcal{W}$ of arcs in D is* $(D; F, X)$-feasible *if the following conditions hold.*

(i) $d^+_{D_{\mathcal{W}}}(f) \ge d^-_{D_{\mathcal{W}}}(f)$ *for all* $f \in F$;

(ii) $d^-_{D_{\mathcal{W}}}(x) \ge d^+_{D_{\mathcal{W}}}(x)$ *for all* $x \in X$;

(iii) *$d^-_{D_\mathcal{W}}(y) = d^+_{D_\mathcal{W}}(y)$ for all $y \in V(D_\mathcal{W}) - F - X$;*
(iv) *No distinct red arcs in $D_\mathcal{W}$ have the same initial vertex;*
(v) *There are no 2-cycles in $D_\mathcal{W}$.*

Define $R(\mathcal{W})$ as follows.

$$R(\mathcal{W}) = \sum_{f \in F}(d^+_{D_\mathcal{W}}(f) - d^-_{D_\mathcal{W}}(f)) = \sum_{x \in X}(d^-_{D_\mathcal{W}}(x) - d^+_{D_\mathcal{W}}(x))$$

Definition 4. *Let $\mathcal{W}$ be a $(D; F, X)$-feasible multiset of arcs in D and let $W = w_1w_2 \dots w_k$ be a walk in D. Let $\mathcal{R}$ denote all vertices in D with a red arc out of them in $\mathcal{W}$ and let $RED(D)$ denote all red arcs in D. A vertex $w_i \in V(W)$ is said to be either $(\mathcal{W}, W)$-special, $(\mathcal{W}, W)$-normal or $(\mathcal{W}, W)$-forbidden depending on the following.*

(a) *w_i is $(\mathcal{W}, W)$-special if and only if $1 < i < k$, $w_iw_{i-1} \in \mathcal{W} \cap RED(D)$ and $w_iw_{i+1} \in RED(D)$.*
(b) *w_i is $(\mathcal{W}, W)$-normal if and only if w_i is not $(\mathcal{W}, W)$-special and the following holds: $i = k$ or $w_iw_{i+1} \notin RED(D)$ or $w_{i+1}w_i \in \mathcal{W}$ or $w_i \notin \mathcal{R}$.*
(c) *w_i is $(\mathcal{W}, W)$-forbidden if it is not $(\mathcal{W}, W)$-special or $(\mathcal{W}, W)$-normal. In other words, w_i is $(\mathcal{W}, W)$-forbidden if and only if $i < k$ and $w_iw_{i-1} \notin \mathcal{W} \cap RED(D)$ (or $i = 1$) and $w_iw_{i+1} \in RED(D)$ and $w_{i+1}w_i \notin \mathcal{W}$ and $w_i \in \mathcal{R}$.*

*We now define a $(D; \mathcal{W}; F, X)$-*feasible walk*, W, in D as any (F, X)-walk where for every vertex $x \in V(D)$, x appears at most once on W as a $(\mathcal{W}, W)$-special vertex, it appears at most once on W as a $(\mathcal{W}, W)$-normal vertex and it does not appear at all as a $(\mathcal{W}, W)$-forbidden vertex.*

Lemma 1. *Let $\mathcal{W}$ be a $(D; F, X)$-feasible multiset of arcs in D and let $W = w_1w_2 \dots w_k$ be an (F, X)-walk in D. If no vertex on W is $(\mathcal{W}, W)$-forbidden then there exists a $(D; \mathcal{W}; F, X)$-feasible walk, W', in D from w_1 to w_k.*

Proof: Assume without loss of generality that W is the shortest walk from w_1 to w_k in D without any $(\mathcal{W}, W)$-forbidden vertices. For the sake of contradiction assume that W is not a $(D; \mathcal{W}; F, X)$-feasible walk, which implies that some vertex $x \in V(D)$ appears on W at least twice as a $(\mathcal{W}, W)$-special vertex or at least twice as a $(\mathcal{W}, W)$-normal vertex.

First assume that $x = w_i = w_j$ and $1 < i < j < k$ and both w_i and w_j are $(\mathcal{W}, W)$-special. This means that $w_1w_2 \dots w_iw_{j+1}w_{j+2} \dots w_k$ is a walk from w_1 to w_k containing no $(\mathcal{W}, W)$-forbidden vertices (as w_i is still $(\mathcal{W}, W)$-special and no other vertex changes status). This contradicts the minimality of W.

So now assume that $x = w_i = w_j$ and $1 \le i < j \le k$ and both w_i and w_j are $(\mathcal{W}, W)$-normal. Again we note that $W' = w_1w_2 \dots w_iw_{j+1}w_{j+2} \dots w_k$ is a walk from w_1 to w_k containing no $(\mathcal{W}, W)$-forbidden vertices (if $j = k$, then $W' = w_1w_2 \dots w_i$). This again contradicts the minimality of W. These contradictions imply that W is a $(D;\mathcal{W};F,X)$-feasible walk. □

Corollary 1. *Let $\mathcal{W}$ be a $(D;F,X)$-feasible multiset of arcs in D. If $W = w_1w_2 \dots w_k$ and $W' = w_kw_{k+1} \dots w_l$ are $(D;\mathcal{W};F,X)$-feasible walks in D, then there exists a $(D;\mathcal{W};F,X)$-feasible walk from w_1 to w_l in D.*

Proof: Let $W^* = w_1w_2 \dots w_kw_{k+1} \dots w_l$. As w_k is not a $(\mathcal{W}, W^*)$-forbidden vertex on W^* (as otherwise w_k would be $(\mathcal{W}, W')$-forbidden on W') we note that there are no $(\mathcal{W}, W^*)$-forbidden vertices on W^*. We are now done by Lemma 1. □

Recall that if $\mathcal{W}$ is a multiset of arcs and W is a walk, then if some arc appears i times in $\mathcal{W}$ and j times in $A(W)$ then it appears $i+j$ times in $A(W) \cup \mathcal{W}$.

Lemma 2. *Let $\mathcal{W}$ be a $(D;F,X)$-feasible multiset of arcs in D and let W be a $(D;\mathcal{W};F,X)$-feasible walk in D. Let $\mathcal{W}'$ be obtained from $A(W) \cup \mathcal{W}$ after deleting pairs xy, yx of arcs until there is no 2-cycles anymore. Then $\mathcal{W}'$ is a $(D;F,X)$-feasible multiset of arcs in D with $R(\mathcal{W}') = R(\mathcal{W}) + 1$.*

Proof: Let $\mathcal{W}'$ be defined as in the statement of the lemma and let $\mathcal{W}'' = A(W) \cup \mathcal{W}$. As W is a walk from F to X we note that (i), (ii) and (iii) in Definition 3 hold for $\mathcal{W}''$ and $R(\mathcal{W}'') = R(\mathcal{W}) + 1$. However this implies that (i), (ii) and (iii) in Definition 3 also hold for $\mathcal{W}'$ and $R(\mathcal{W}') = R(\mathcal{W}) + 1$, as deleting 2-cycles have no effect on $d^-(y) - d^+(y)$ for any $y \in V(D)$. By the definition of a $(D;\mathcal{W};F,X)$-feasible walk in D we note that (iv) in Definition 3 holds for $\mathcal{W}'$ (as the only way a vertex can increase the number of red arcs leaving it is if x is a $(\mathcal{W}, W)$-normal vertex in W and x did not have any red arcs leaving it in $\mathcal{W}$). By the construction of $\mathcal{W}'$ we also note that (v) in Definition 3 holds for $\mathcal{W}'$. □

We say that a set $\mathcal{S} = \{Q_1, Q_2, \dots, Q_p\}$ of paths and cycles in a directed multigraph M is a *decomposition* of M if each arc of M belongs to exactly one element of $\mathcal{S}$.

Lemma 3. *Let $\mathcal{W}$ be a $(D;F,X)$-feasible multiset of arcs in D. Then $D_{\mathcal{W}}$ can be decomposed into $W_1, W_2, \dots, W_{R(\mathcal{W})}$, $C_1, C_2, \dots, C_k$ (for some $k \geq 0$), such that W_i is a path from F to X in $D_{\mathcal{W}}$ for all $i = 1, 2, \dots, R(\mathcal{W})$ and C_j is a cycle in $D_{\mathcal{W}}$ for all $j = 1, 2, \dots, k$.*

Proof: If $R(\mathcal{W}) = 0$, $D_{\mathcal{W}}$ is eulerian and it is well-known that $D_{\mathcal{W}}$ can be decomposed into a number of cycles. So assume that $R(\mathcal{W}) > 0$. We will use induction on $|A(D_{\mathcal{W}})|$. Let $u_1 \in F$ be any vertex with $d^+_{D_{\mathcal{W}}}(u_1) > d^-_{D_{\mathcal{W}}}(u_1)$. Starting at u_1 and moving to a successor vertex until we reach an already visited vertex, in which case we obtain a cycle, or we reach a vertex in X, in which case we obtain an (F,X)-path in $D_{\mathcal{W}}$. Remove the arcs of this cycle or path from $\mathcal{W}$ and use induction if $R(\mathcal{W}) > 0$ or the above case when $R(\mathcal{W}) = 0$. It is not difficult to see that this results in the desired decomposition, with exactly $R(\mathcal{W})$ (F,X)-paths in $D_{\mathcal{W}}$. □

Lemma 4. *Let $\mathcal{W}$ be a $(D; F, X)$-feasible multiset of arcs in D, such that $R(\mathcal{W}) \leq n_{in}$. Assume that $O_N^{cl} \subseteq X$ and that for every vertex $o \in O$ there is an (o, F)-path in $N - (O - \{o\})$. Let Q be a set of vertices such that $X \subseteq Q \subseteq V(D) - F$. If there is no $(D; \mathcal{W}; V(D) - Q, Q)$-feasible walk in D then Q is a convex set in N satisfying the input and output constraints. Furthermore, O is the set of output vertices of Q.*

Proof: Let Q be defined as stated in the lemma. Assume that Q is not convex, implying that there is a $w \notin Q$ such that there exists a (w, Q)-path, P, in N and a (Q, w)-path, P', in N. Let P'' be the reverse of P' and note that if no vertex of P' belongs to O, then P'' is a blue path in D from $w \in V(D) - Q$ to Q. As there is no $(D; \mathcal{W}; V(D) - Q, Q)$-feasible walk in D we must have that some vertex on P' does indeed belong to O. Let $o_1 \in V(P') \cap O$ be arbitrary.

Let p be the terminal vertex of P and note that $p \in Q$. As there is a path, P^*, in N from p to F (recall that there exist (u, F)-paths for all $u \in V(N)$), we get a blue path from F to Q unless some vertex on the P^* belongs to O. Let this vertex be $o_2 \in V(P^*) \cap O$. By the above construction, we have an (o_1, w)-path in N and an (w, o_2)-walk in N (by merging P and part of P^*). However this implies that $w \in O_N^{cl} \subseteq X \subseteq Q$, a contradiction.

We will now prove that the input and output constraints are satisfied. Assume that there is some arc, xy, out of Q in N where $x \notin O$. Thus, yx is a blue arc in D and yx is a $(D; \mathcal{W}; V(D) - Q, Q)$-feasible walk in D, a contradiction. So the only arcs out of Q in N come from O. Let $o \in O$ be arbitrary and recall that there is an (o, F)-path in $N - (O - \{o\})$, which implies that some vertex on this path is an output vertex for Q. Therefore, this vertex must be o, so we have now shown that O is exactly the output vertices for Q.

Assume that the input constraint is not satisfied and that $\{x_1, x_2, \ldots, x_r\}$ is a set of vertices in $V(N) - Q$ with arcs into Q in N and $r > n_{in}$. Let $\{y_1, y_2, \ldots, y_r\}$ be defined such that $x_i y_i$ is an $(V(D) - Q, Q)$-arc in N for all $i \in \{1, 2, \ldots, r\}$. By Lemma 3 let $W_1, W_2, \ldots, W_{R(\mathcal{W})}$, $C_1, C_2, \ldots, C_k$ be a decomposition of $\mathcal{W}$, such that W_i is a path from F to X in $D_{\mathcal{W}}$ for all $i = 1, 2, \ldots, R(\mathcal{W})$ and C_j is a cycle in $D_{\mathcal{W}}$ for all $j = 1, 2, \ldots, k$.

Assume that there is some $x_i \in \{x_1, x_2, \ldots, x_r\}$ such that there is no red (x_i, Q)-arc in $\mathcal{W}$. If there is no red arc out of x_i in $\mathcal{W}$ at all, then the arc $x_i y_i$ contradicts the fact that there is no $(D; \mathcal{W}; V(D) - Q, Q)$-feasible walk in D. So let $x_i u$ be a red arc in $\mathcal{W}$ where $u \notin Q$. However the path $u x_i y_i$ again contradicts the fact that there is no $(D; \mathcal{W}; V(D) - Q, Q)$-feasible walk in D. So for every vertex in $\{x_1, x_2, \ldots, x_r\}$ there exists a red (x_i, Q)-arc in $\mathcal{W}$. Without loss of generality we may assume that $\{y_1, y_2, \ldots, y_r\}$ was chosen such that $x_i y_i$ is red and $x_i y_i \in \mathcal{W}$ for all $i = 1, 2, \ldots, k$. If for some $i \in \{1, 2, \ldots, k\}$ the arc $x_i y_i$ belongs to a cycle $C_a \in \{C_1, C_2, \ldots, C_k\}$, then there is a $(Q, V(D) - Q)$-arc, uv, in C_a. However, the path vu is a $(D; \mathcal{W}; V(D) - Q, Q)$-feasible walk in D, a contradiction.

As $r > n_{in} \geq R(\mathcal{W})$ there must be some path in $\{W_1, W_2, \ldots, W_{R(\mathcal{W})}\}$ that contains at least two arcs from $\{x_1 y_1, x_2 y_2, \ldots, x_r y_r\}$. Without loss of generality assume that $x_1 y_1$ is the first such arc on W_1 and $x_2 y_2$ is the second such arc on W_1. This implies that there is a walk from $y_1 \in Q$ to $x_2 \notin Q$, containing a $(Q, V(D) - Q)$-arc, uv. However the path vu is a $(D; \mathcal{W}; V(D) - Q, Q)$-feasible walk in D, a contradiction. This contradiction against $r > n_{in}$ implies that the input constraint is satisfied. □

Lemma 5. *Let $\mathcal{W}$ be a $(D; F, X)$-feasible multiset of arcs in D, such that $R(\mathcal{W}) > n_{in}$. Then there is no convex set of vertices, Q, in N, such that $X \subseteq Q \subseteq V(D) - F$ and Q satisfies the input constraint and has O as its output vertices.*

Proof: For the sake of contradiction assume that there exists a convex set of vertices, Q, in N, such that $X \subseteq Q \subseteq V(D) - F$ and Q satisfies the input constraint and has O as its output vertices. Let $I = \{i_1, i_2, \ldots, i_r\}$ be the input vertices for Q and $r \leq n_{in}$. By Lemma 3 let $W_1, W_2, \ldots, W_{R(\mathcal{W})}, C_1, C_2, \ldots, C_k$ be a decomposition of $\mathcal{W}$, such that W_i is a path from F to X in $D_{\mathcal{W}}$ for all $i = 1, 2, \ldots, R(\mathcal{W})$ and C_j is a cycle in $D_{\mathcal{W}}$ for all $j = 1, 2, \ldots, k$. As $r \leq n_{in} < R(\mathcal{W})$ there must be some path W_i, without loss of generality say W_1, which does not contain a red arc out of any vertex in I (as each vertex in I has at most one red arc out of it in $\mathcal{W}$). Let uv be a $(V(D) - Q, Q)$-arc on W_1. If uv is a red arc then $u \notin I$, contradicting the fact that I is the set of input vertices. So uv is a blue $(V(D) - Q, Q)$-arc in D. Hence $u \in N^*$ and $v \in O$ contrary to the definition of N^*. □

Lemma 6. *Let $\mathcal{W}$ be a $(D; F, X)$-feasible multiset of arcs in D. In time $O(|V(N)| + |A(N)|)$ we can find a $(D; \mathcal{W}; F, X)$-feasible walk in D if it exists or determine that it does not exist. If it does not exist we can also determine the following two sets:*

$$S = \{u \mid \textit{there is a } (D; \mathcal{W}; F, \{u\})\textit{-feasible walk in } D\}$$
$$T = \{v \mid \textit{there is a } (D; \mathcal{W}; \{v\}, X)\textit{-feasible walk in } D\}$$

Proof: We will define a digraph D' as follows. Let $\mathcal{R}$ contain all vertices in D which have a red arc out of them in $\mathcal{W}$. Let the vertex set of D' be $V(D') = V(D) \cup \{r' \mid r \in \mathcal{R}\}$ (that is we duplicate all vertices in $\mathcal{R}$). For all arcs $uv \in A(D)$ add the following arcs to D'.

(R1) If uv is red and $vu \notin \mathcal{W}$ and $u \in \mathcal{R}$, then add $u'v$ to D'.
(R2) If uv is red and $vu \in \mathcal{W}$ or $u \notin \mathcal{R}$, then add uv to D'.
(B1) If uv is blue and $vu \in \mathcal{W}$, then add uv and uv' to D'.
(B2) If uv is blue and $vu \notin \mathcal{W}$, then add uv to D'.

Now use any algorithm such as depth (or breadth) first search to find an (F, X)-path in D' if it exists (we start and end in vertices of the form u and not u'). First assume that such a path $P' = x_1 x_2 \ldots x_k$ exists. Replacing vertices of the form u' with u, we obtain a walk $W = w_1 w_2 \ldots w_k$ in D, which we will show is a $(D; \mathcal{W}; F, X)$-feasible walk. If $x_i = w_i'$ then, since there are no arcs of the form $y'z'$ in D', we must have $x_{i-1} = w_{i-1}$ and $x_{i+1} = w_{i+1}$. Thus, it is not difficult to see that w_i a $(\mathcal{W}, W)$-special vertex in W. Whereas if $w_i = x_i$ then either $i = k$ or **R2**, **B1** or **B2** holds for $w_i w_{i+1}$ and so w_i is $(\mathcal{W}, W)$-normal in W. Thus, since P' is a path, W is indeed a $(D; \mathcal{W}; F, X)$-feasible walk in D.

Now assume that some $(D; \mathcal{W}; F, X)$-feasible walk, W, in D exists. Let $W = w_1 w_2 w_3 \cdots w_l$. If w_i is $(\mathcal{W}, W)$-special in W then change it to w_i' in D'. After doing this for all special vertices we note that we get a path from F to X in D'. So we have now shown that there exists a $(D; \mathcal{W}; F, X)$-feasible walk in D if and only if there exists an (F, X)-path in D'. This gives us the correct time complexity as $|V(D')| \le 2|V(D)|$ and $|A(D')| \le 2|A(D)|$.

If there is no (F, X)-path in D' then it is not difficult to find the set of vertices $Z \subseteq V(D')$, such that there is an (F, z)-path in D' if and only if $z \in Z$. Now let $S = \{u \in V(D) \mid u \in Z\}$ (note that $\{u \in V(D) \mid u \in Z \text{ or } u' \in Z\}$ is an equivalent definition of S). It is not difficult to see that S is the set of vertices in $V(D)$ for which there exists a $(D; \mathcal{W}; F, \{s\})$-feasible walk in D. Analogously we can find T. □

3 The Algorithm

The algorithm $\mathcal{A}(N, F)$ described below makes a call to $\mathcal{B}(\emptyset, F, X)$ for all possible output sets O (see A.2) where $X = O_N^{cl}$. The procedure $\mathcal{B}(\mathcal{W}, F, X)$ will then find all convex sets, Q, satisfying the input constraint and having O as the output vertices and satisfying $X \subseteq Q \subseteq V(N) - F$.

Lemma 7. *The sets saved by $\mathcal{A}(N, F)$ are precisely the valid convex sets of N and furthermore no such set is saved more than once.*

Proof: We only save solutions in B.4 or B.5.2. In both cases, using Corollary 1 and Lemma 4, it can be seen that the saved set is a valid convex set.

Now let Q' be a valid convex set. Let O' be the output vertices of Q'. Assume that for some $o' \in O'$ there is no (o', F)-path in $N - (O' - \{o'\})$. Let $y \in N_N^+(o')$ be arbitrary and assume that $y \notin Q'$. However there is no (y, F)-path in $N - (O' - \{o'\})$ but there is a (y, F)-path in N. This implies that there is a $(y, O' - \{o'\})$-path in N. Therefore $y \in Q'$ (as Q' is convex), a contradiction. So for every $o' \in O'$ there is a (o', F)-path in $N - (O' - \{o'\})$. Note that $X = (O')_N^{cl} \subseteq Q'$, as Q' is convex and we will make a call to $\mathcal{B}(\emptyset, F, X)$ in A.2.3 of $\mathcal{A}(N, F)$.

If we return in B.1, then we did not have $X \subseteq Q' \subseteq V(D) - F$, by Lemma 5. If we make recursive calls in B.3, then the desired recursive call is $\mathcal{B}(\mathcal{W}, F, (X \cup \{u\})_N^{cl})$ if $u \in Q'$ (note that if $X \cup \{u\} \subseteq Q'$ and Q' is convex, then $(X \cup \{u\})_N^{cl} \subseteq Q'$) and $\mathcal{B}(\mathcal{W}, F \cup \{u\}, X)$ if $u \notin Q'$. Now assume that T is saved in B.4. We note that $S \cap V(Q') = \emptyset$ by Lemma 5 (as otherwise we obtain a $(D; F, Q')$-feasible multiset, $\mathcal{W}'$, with $R(\mathcal{W}') > n_{in}$, by adding a $(D; \mathcal{W}; F, \{s\})$-feasible walk to $\mathcal{W}$, where $s \in S \cap V(Q')$). Analogously $T - V(Q') = \emptyset$ by Lemma 5 (as again we obtain a $(D; V(D) - Q', Q')$-feasible multiset, $\mathcal{W}'$, with $R(\mathcal{W}') > n_{in}$, by adding a $(D; \mathcal{W}; \{t\}, X)$-feasible walk to $\mathcal{W}$, where $t \in T - V(Q')$). Therefore $Q' = T$ and Q' is saved.

```
Algorithm $\mathcal{A}(N, F)$
  A.1 Find an acyclic order $u_1, u_2, \ldots, u_n$ of the vertices in $N$ (that
      is, if $u_i u_j \in A(N)$ then $i < j$).
  A.2 For all sets $O \subseteq V(N)$, where for every vertex $o \in O$ there is
      an $(o, F)$-path in $N - (O - \{o\})$ and $|O| \leq n_{out}$ do the following.
    A.2.1 Find $N^*$ and $D$ as in Definition 2.
    A.2.2 Let $X = O_N^{cl}$ (see Definition 1).
    A.2.3 Make a call to $\mathcal{B}(\emptyset, F, X)$ (see below).
```

```
Algorithm $\mathcal{B}(\mathcal{W}, F, X)$
  B.1 If $R(\mathcal{W}) > n_{in}$ or if $X \cap F \neq \emptyset$, then there is no solution so return.
  B.2 Use Lemma 6 to determine if there is a $(D; \mathcal{W}; F, X)$-feasible walk in $D$.
      If there is and $R(\mathcal{W}) \leq n_{in}$ then add it to $\mathcal{W}$ using the approach in Lemma
      2 and go to B.1. Otherwise determine $S$ and $T$ as in Lemma 6.
  B.3 If $V(D) \neq S \cup T$ then let $u \in V(D) - S - T$ be arbitrary. Now make
      recursive calls $\mathcal{B}(\mathcal{W}, F, (X \cup \{u\})_N^{cl})$ and $\mathcal{B}(\mathcal{W}, F \cup \{u\}, X)$ and return.
  B.4 If $R(\mathcal{W}) = n_{in}$, then by B.3 we have $V(D) = S \cup T$. In this case save $T$
      as a solution and return.
  B.5 If $R(\mathcal{W}) < n_{in}$ then consider the following possibilities. Note that $V(D) =$
      $S \cup T$.
    B.5.1 If $X \neq T$ then let $u_a \in T - X$ be chosen such that $a$ is minimum.
      Make the recursive calls $\mathcal{B}(\mathcal{W}, F, (X \cup \{u_a\})_N^{cl})$ and $\mathcal{B}(\mathcal{W}, F \cup \{u_a\}, X)$ and
      return.
    B.5.2 If $X = T$, then save $T$ as a solution. Let $i_1, i_2, \ldots, i_k$ be the vertices
      in $V(D) - T$ with red arcs into $T$ in $D$. Make the following recursive calls.
      $\mathcal{B}(\mathcal{W}, F, (X \cup \{i_1\})_N^{cl})$,
      $\mathcal{B}(\mathcal{W}, F \cup \{i_1\}, (X \cup \{i_2\})_N^{cl})$,
      $\mathcal{B}(\mathcal{W}, F \cup \{i_1, i_2\}, (X \cup \{i_3\})_N^{cl})$,....,
      $\mathcal{B}(\mathcal{W}, F \cup \{i_1, i_2, \ldots, i_{k-1}\}, (X \cup \{i_k\})_N^{cl})$.
```

If we make recursive calls in B.5.1, then the desired recursive call is $\mathcal{B}(\mathcal{W}, F, (X \cup \{u_a\})_N^{cl})$ if $u_a \in Q'$ and $\mathcal{B}(\mathcal{W}, F \cup \{u_a\}, X)$ if $u_a \notin Q'$. If we perform B.5.2, then we return Q' if $\{i_1, i_2, \ldots, i_k\} \cap Q' = \emptyset$. If $\{i_1, i_2, \ldots, i_k\} \cap Q' \neq \emptyset$ then let j be the minimum index such that $i_j \in \{i_1, i_2, \ldots, i_k\} \cap Q'$ and note that the desired recursive call is $\mathcal{B}(\mathcal{W}, F \cup \{i_1, i_2, \ldots, i_{j-1}\}, (X \cup \{i_j\})_N^{cl})$.

We have shown that Q' will be saved and we will now prove that Q' cannot be saved twice. As we only consider sets O with the property that for every vertex $o \in O$ there is an (o, F)-path in $N - (O - \{o\})$, we note that Q' cannot be saved in two distinct calls in A.2.3 (only when O is exactly the output set of Q'). Furthermore as all recursive calls either add a vertex to the forbidden set or that same vertex to X the same set cannot be saved in two different recursive calls (in B.5.2. we noted above exactly in which recursive call Q' would be saved if $\{i_1, i_2, \ldots, i_k\} \cap Q' \neq \emptyset$ and otherwise it would not be saved in any recursive call, but only in the current call). $\square$

We omit proof of the following due to the limited space.

Lemma 8. *If N is a connected acyclic digraph of order n, size m and containing $c(N)$ valid convex sets, then $\mathcal{A}(N, F)$ has time complexity $O(m \cdot N_{in}^2(c(N) + n^{N_{out}}) + m)$.*

The last two lemmas imply the following:

Theorem 1. *Let N be an acyclic digraph with n vertices and m arcs. The algorithm $\mathcal{A}(N, F)$ finds all valid convex sets in N in time $O(m \cdot n_{in}^2(c(N) + n^{n_{out}}) + m)$, where $c(N)$ is the number of valid convex sets.*

4 Experiments

We have implemented $\mathcal{A}$ and tested it against the state-of-the-art algorithm of Chen, Maskell and Sun [2] (the CMS algorithm) and the well-known algorithm of Atasu, Pozzi and Ienne [5] (the API algorithm) using both synthetic examples and DDGs generated from real world applications. The source of the CMS implementation was kindly provided by its authors. All algorithms were implemented in C++ and experimental data were produced using Dual Core AMD Opteron 265 1.8GHz processors with 4Gbyte of RAM, running 64-bit SUSE Linux 10.2.

Figure 1 shows the performance of these algorithms on synthetic tree and acyclic lattice digraphs with $n_{in} = 3$ and $n_{out} = 2$. In both cases, algorithm $\mathcal{A}$ consistently outperforms the current state of the art with the performance of CMS only slightly superior to the API algorithm on tree-like graphs.

Table 1 shows results from five real world C++ programs in the MiBench benchmark suite [3]. We selected a large (150–1800 lines of intermediate code) basic block from within a critical loop of each program: typically the compiler will have unrolled this block to some degree. The resulting DDGs were augmented

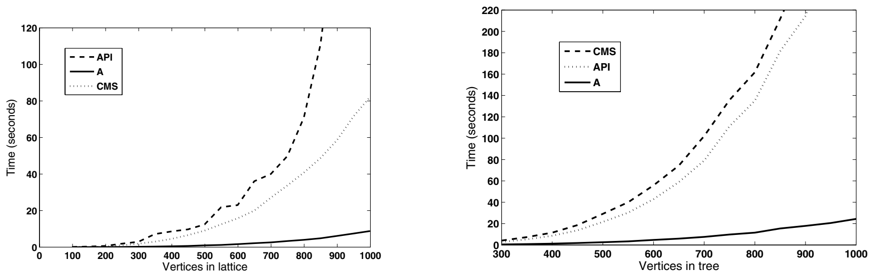


Fig. 1. Performance on lattice and tree synthetic digraphs under I/O constraints

Table 1. Comparative performance on real world examples

Input	n_{in}	n_{out}	$c(N)$ sets	Time CMS	Time API06	Time $\mathcal{A}$	Calls CMS	Calls API06	Calls $\mathcal{A}$
bf	2	1	482	0.04	2.78	0.01	24,009	796,775	631
	4	1	1,920	0.07	15.71	0.04	34,084	4,467,923	3,507
	6	1	7,669	0.11	34.61	0.17	55,374	9,816,778	15,005
	3	2	7,831	0.35	91.51	0.15	176,631	25,169,197	12,302
	5	2	40,714	0.79	352.95	0.92	383,570	101,091,122	75,376
	4	3	105,599	2.31	DNF	2.81	1,122,520	DNF	189,037
	6	3	570,197	7.02	DNF	14.57	3,342,391	DNF	1,085,505
	8	3	2,155,103	17.37	DNF	71.95	8,329,766	DNF	4,253,251
cjpeg	2	1	406	0.02	0.10	0.00	21,907	61,832	694
	4	1	544	0.02	0.10	0.00	22,003	70,216	970
	6	1	550	0.01	0.11	0.00	22,003	70,216	982
	3	2	41,363	0.61	13.86	0.28	677,813	9,880,064	76,889
	5	2	113,611	0.82	19.30	1.02	875,155	13,460,590	220,929
	7	2	140,335	0.94	20.15	1.53	896,688	13,721,462	274,377
	4	3	2,201,568	20.50	DNF	18.78	18,454,621	DNF	4,236,388
rijndael	2	1	1241	2.79	51.43	0.05	697778	5473096	1636
	4	1	4,787	3.51	253.30	0.17	786,732	27,471,175	8,728
	6	1	15,236	4.09	DNF	0.59	878,083	DNF	29,626
	3	2	75,241	83.96	DNF	3.98	11,575,641	DNF	145,477
	5	2	648,748	201.41	DNF	23.88	31,777,459	DNF	1,207,733
sha	2	1	1,546	3.76	DNF	0.13	300,752	DNF	1,632
	4	1	4,372	4.23	DNF	0.24	345,994	DNF	7,284
	6	1	10,152	5.30	DNF	0.49	432,350	DNF	18,844
	3	2	78,132	85.14	DNF	6.75	6,450,724	DNF	117,159
	5	2	293,259	164.97	DNF	15.37	12,652,418	DNF	494,521
md5	2	1	893	1.54	DNF	0.04	329,373	DNF	969
	4	1	2,304	1.66	DNF	0.08	342,133	DNF	3,791
	6	1	3,546	1.70	DNF	0.12	349,486	DNF	6,275
	3	2	54,476	51.45	DNF	3.24	6,109,809	DNF	102,389

with forbidden vertices, which represent values external to the basic block to give the following examples: the BlowFish encryption algorithm (bf) with 467 vertices of which 134 are forbidden; JPEG image compression (cjpeg) with 152 vertices, 34 forbidden; AES encryption (rijndael) with 1237 vertices, 391 forbidden; secure message digest hashing (sha) with 1811 vertices, 351 forbidden; and MD5 with 1170 vertices, 353 forbidden.

The API algorithm is not competitive with either $\mathcal{A}$ or CMS for these graphs, supporting the conclusions in [2]. Although the CMS algorithm performs better on some small to medium examples, on large examples the effect of the better asymptotic complexity is clear — on the rijindel, sha and md5 benchmarks, $\mathcal{A}$ clearly has a performance advantage. We also note that in every case, the number of recursions made by $\mathcal{A}$ was significantly lower.

References

1. Bonzini, P., Pozzi, L.: Polynomial-time subgraph enumeration for automated instruction set extension. In: DATE 2007, pp. 1331–1336 (2007)
2. Chen, X., Maskell, D.L., Sun, Y.: Fast identification of custom instructions for extensible processors. IEEE Trans. Computer-Aided Design Integr. Circuits Syst. 26, 359–368 (2007)
3. Guthaus, M.R., Ringenberg, J.S., Ernst, D., Austin, T.M., Mudge, T., Brown, R.B.: MiBench: A free, commercially representative embedded benchmark suite. In: Proceedings of the WWC-4, 2001 IEEE International Workshop on Workload Characterization, pp. 3–14 (2001)
4. Gutin, G., Johnstone, A., Reddington, J., Scott, E., Soleimanfallah, A., Yeo, A.: An algorithm for finding connected convex subgraphs of an acyclic digraph. In: Algorithms and Complexity in Durham, 2007. College Publications (2008)
5. Pozzi, L., Atasu, K., Ienne, P.: Exact and approximate algorithms for the extension of embedded processor instruction sets. IEEE Trans. on CAD of Integrated Circuits and Systems 25, 1209–1229 (2006)
6. Reddington, J.: Improvements to instruction identification for custom instruction set design. PhD Thesis, Royal Holloway, University of London (2008)

Cutwidth of Split Graphs, Threshold Graphs, and Proper Interval Graphs

Pinar Heggernes, Daniel Lokshtanov, Rodica Mihai, and Charis Papadopoulos

Department of Informatics, University of Bergen
pinar@ii.uib.no, daniello@ii.uib.no,
rodica@ii.uib.no, charis@cs.uoi.gr

Abstract. We give a linear-time algorithm to compute the cutwidth of threshold graphs, thereby resolving the computational complexity of cutwidth on this graph class. Although our algorithm is simple and intuitive, its correctness proof relies on a series of non-trivial structural results, and turns out to be surprisingly complex. Threshold graphs are a well-studied subclass of interval graphs and of split graphs, both of which are unrelated subclasses of chordal graphs. To complement our result, we show that cutwidth is NP-complete on split graphs, and consequently also on chordal graphs. In addition, we show that cutwidth is trivial on proper interval graphs, another subclass of interval graphs. The cutwidth of interval graphs is open, and only very few graph classes are known so far on which polynomial-time cutwidth algorithms exist. Thus we contribute to define the border between graph classes on which cutwidth is polynomially solvable and on which it remains NP-complete.

1 Introduction

The cutwidth problem asks, given a graph G, and a positive integer k, whether there exists a linear layout of the vertices of G so that any line inserted between two consecutive vertices of the layout cuts (intersects with) at most k edges. The cutwidth of the input graph is the smallest integer for which the question can be answered positively. This important graph layout problem was first proposed as a model to minimize the number of channels in a circuit [1,19], and more recently it has found applications in areas like protein engineering [3], network reliability [16], automatic graph drawing [21], information retrieval [4], and as a subroutine in the cutting plane algorithm for TSP [14].

Like many other interesting graph problems, cutwidth is NP-complete [8], even when input graphs are restricted to planar graphs of maximum degree three [20], unit disk graphs, partial grids [9], and consequently bipartite graphs.

Coping with the NP-completeness of the problem has been mainly channeled via approximation algorithms and fixed parameter algorithms. There is a polynomial-time $O(\log^2 n)$-approximation algorithm for general graphs [17], and a polynomial-time constant factor approximation algorithm for dense graphs [2]. The best known parameterized algorithm for cutwidth so far runs in linear time (but of course exponential in the parameter k) [22].

Polynomial-time algorithms for the exact computation of cutwidth are known only for very few graph classes. For certain trivial graph classes, like meshes or complete p-partite graphs, there exist closed formulas for their cutwidth (see [10]). The cutwidth of trees can be computed in $O(n \log n)$ time by a sophisticated and technical algorithm [24] (see also [6]). The cutwidth of graphs both of whose treewidth *and* maximum degree can be bounded by constants, can be computed in polynomial time by advanced methods [23]. The computational complexity of cutwidth on threshold graphs has been open until now [10].

In this paper, we present an $O(n)$-time algorithm for computing the cutwidth of threshold graphs with n vertices. Threshold graphs are a well-studied graph class with a variety of theoretical applications [18], and they are both split graphs and interval graphs [5,12]. Split and interval graphs are two unrelated subclasses of the widely-known class of chordal graphs. Before presenting our algorithm for threshold graphs, we show that the cutwidth problem remains NP-complete on split graphs (even on a very restricted type of split graphs), and hence also on chordal graphs. As a complementary result, we show that for another subclass of interval graphs, proper interval graphs, the cutwidth problem has a trivial solution, which does not work for interval graphs in general or for threshold graphs. Our findings are summarized in Figure 1.

The algorithm that we present for threshold graphs is simple and intuitive, and interestingly its execution does not at all depend on properties of threshold graphs; thus it can also be run on general graphs as a heuristic. For the proof of correctness of this algorithm on threshold graphs, we study the properties of a possible minimal counterexample through a series of structural results, and we show that the assumption of the existence of such a counterexample leads to a contradiction. (In this extended abstract some proofs have been omitted.) Although threshold graphs can be viewed as a quite restricted graph class, designing an algorithm for their cutwidth and proving its correctness proved to be a much more challenging task than expected. Extending our results even to trivially perfect graphs, which is a superclass of threshold graphs and a subclass of interval graphs, seems to be a non-trivial problem.

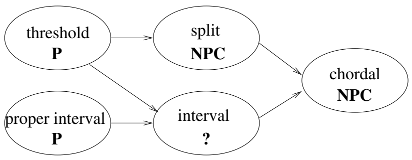


Fig. 1. The graph classes studied in this paper, and the complexity of cutwidth on each class according to our results. P means polynomial and NPC means NP-complete. The arrow represents the subset relation.

2 Preliminaries

We consider labeled undirected finite graphs with no loops or multiple edges. For a graph $G = (V, E)$, we denote its vertex and edge set by V and E, respectively, with $n = |V|$. Every vertex $v \in V$ has a distinct label, $label(v)$, between 1 and n. We say that a vertex u *is smaller than* v if $label(u) < label(v)$. For a vertex subset $S \subseteq V$, the subgraph of G induced by S is denoted by $G[S]$. Moreover, we denote by $G - S$ the graph $G[V \setminus S]$ and by $G - v$ the graph $G[V \setminus \{v\}]$. We write $G - uv$ to denote the graph $(V, E \setminus \{uv\})$.

The *neighborhood* of a vertex x of G is $N_G(x) = \{v \mid xv \in E\}$. The *closed neighborhood* of x is $N_G[x] = N_G(x) \cup \{x\}$. The *degree* of x is $\Delta_G(x) = |N_G(x)|$. If $S \subseteq V$, then $N_G(S) = \bigcup_{x \in S} N_G(x) \setminus S$. We define the *cut* of S to be $\delta^G(S) = \{uv \in E \mid u \in S, v \notin S\}$, and the *cut size* of S to be $d^G(S) = |\delta^G(S)|$. Vertex x is *universal* if $N_G[x] = V$ and *isolated* if $N_G(x) = \emptyset$. We will omit the subscripts and superscripts when there is no ambiguity. A graph is *connected* if there is a path between any pair of vertices. A *connected component* of a disconnected graph is a maximal connected subgraph of it. A *clique* is a set of pairwise adjacent vertices, while an *independent set* is a set of pairwise non-adjacent vertices.

Given a graph $G = (V, E)$, a *layout* L is a one-to-one mapping $L : V \to \{1, \ldots, n\}$. We will also denote a layout L by $< v_1, v_2, \cdots v_n >$ such that $L(v_i) = i$. For an integer i between 1 and n we define the set V_i to be $\{v_1, \cdots, v_i\}$. We say that u *is before* v in L, or $u <_L v$, if $L(u) < L(v)$. The *cut of* G *at the ith gap* in a given layout L is defined as $\delta_L(i) = \delta^G(V_i)$ and $d_L(i) = d^G(V_i)$. The *cutwidth* of a layout L of G is $cw_L(G) = \max_{1 \le i \le n} d^G(V_i)$. The *cutwidth of* G is $cw(G) = \min_L\{cw_L(G)\}$ where L is any possible layout of G. In this paper, an *optimal layout* of G is a layout L such that $cw(G) = cw_L(G)$.

The *bisection width* of G, denoted by $bw(G)$, is the minimum cut size of any set $S \subset V$ on $\lfloor \frac{n}{2} \rfloor$ vertices. Since $\delta(S) = \delta(V \setminus S)$ it follows that $bw(G)$ is is the minimum cut size of S of any set S on $\lfloor \frac{n}{2} \rfloor$ or $\lceil \frac{n}{2} \rceil$ vertices. It should be clear that $bw(G)$ gives a lower bound for $cw(G)$, that is, $bw(G) \le cw(G)$ [10]. We will use the close connection between cutwidth and bisection width actively in some of our proofs. A useful observation is that the cutwidth of a subgraph G cannot exceed the cutwidth of G [10].

A graph is a *split graph* if its vertex set can be partitioned into a clique C and an independent set I, where (C, I) is called a *split partition*. Threshold graphs are a subset of split graphs, and for their original definition we refer to [12,18]. We will use as definition the following characterization: A graph is a *threshold graph* if and only if it has a split partition (C, I) such that vertices of the I (and equivalently the vertices of C) can be ordered by neighborhood inclusion [18].

In fact, if a split partition of G satisfies the above property, then all split partitions of G satisfy it [12,18]. For a graph G with split partition (C, I), if there is a vertex x of C which is not adjacent to any vertex of I then clearly $(C \setminus \{x\},\ I \cup \{x\})$ is also a split partition of G. For our purposes, we will always assume that every vertex of C has a neighbor in I. For a threshold graph G, we refine the sets I and C as follows, and call it a *threshold partition*: $(I_0, I_1, I_2, ..., I_\ell)$ is a partition of I such that I_0 is the set of isolated vertices, and $N(I_1) \subset N(I_2) \subset \ldots \subset N(I_\ell)$, where ℓ is largest possible. Thus all vertices in I_j have the same degree for $0 \le j \le \ell$. This also defines a partition $(C_1, C_2, ..., C_\ell)$ of C, where $C_1 = N(I_1)$ and $C_j = N(I_j) \setminus N(I_{j-1})$ for $2 \le j \le \ell$. Again, all vertices in C_j have the same degree for $1 \leq j \leq \ell$. We say that vertices of C_j and I_j belong to the ith *level* of the clique and of the independent set, respectively. By construction, the sets C_i and I_i are nonempty for every $1 \leq i \leq \ell$. For a vertex v, define $level(v)$ to be the level that v belongs to. Threshold graphs can be recognized, and their threshold partition can be computed, in linear time [12].

Lemma 1 ([13]). *Let G be a threshold graph with threshold partition $((C_1, \ldots, C_\ell), (I_1, \ldots, I_\ell))$. Let uv be an edge such that $u \in I_j$ and $v \in C_j$ for some j. Then $G - uv$ is a threshold graph.*

3 Cutwidth of Split Graphs

Theorem 1. *The cutwidth problem is NP-complete on split graphs.*

Proof. The reduction is from an arbitrary instance of the cutwidth problem. Given an arbitrary graph $G = (V, E)$ with n vertices and m edges, we construct a split graph G' as follows. To start with, G' is a complete graph on V. Let $k = n^2 + 1$. For every edge $uv \in E$ we add k more vertices to G', making each new vertex adjacent to u and v in G'. We say that these vertices of G' *correspond* to the edge uv of G. Observe that G' has $n + km$ vertices where the n vertices of V induce a clique in G'. The remaining km vertices are only adjacent to vertices of this clique. Hence G' is a split graph. Moreover the whole construction can be carried out in polynomial time. We now prove that for any $1 \leq c \leq n^2$ we have $cw(G) \leq c$ if and only if $cw(G') < k(c+1)$. (Note that n^2 is a trivial upper bound on the cutwidth of any graph on n vertices.)

If $cw(G) \leq c$ then consider a layout L for which $cw_L(G) \leq c$. We create a layout L' of G' by ordering the vertices in V in the same order that they have in L. Every vertex x of G' that corresponds to an edge uv of G is placed in an arbitrary position between u and v in L'. Observe that x has degree 2 and is placed between its neighbors, $d_{L'}(L'(x) - 1) = d_{L'}(L'(x))$, and thus, to compute $cw_{L'}(G')$ it is sufficient to consider maximum $d_{L'}(L'(v))$ over all $v \in V$. From the construction of L' it follows that for every vertex v in V, $\delta_{L'}(L'(v))$ contains at most $k \cdot d_L(L(v))$ edges between vertices in V and vertices corresponding to edges of G, and at most n^2 edges between pairs of vertices in V. Thus $d_{L'}(L'(v)) \leq k \cdot d_L(L(v)) + n^2$ and $k \cdot d_L(L(v)) + n^2 < kc + k = k(c+1)$ and $cw(G') \leq cw_{L'}(G') < k(c+1)$ follows.

Let L' be a layout of G' for which $cw_{L'}(G') < k(c+1)$. ¿From L' we construct a layout L of G by ordering the vertices of V in the same order that L' orders them. We prove that $cw_L(G) \leq c$. For a given vertex x we observe that for every edge $uv \in \delta_L(L(x))$ and vertex y of G' corresponding to the edge uv, either the edge yu or the edge yv must be in $\delta_{L'}(L'(x))$. Thus, $k \cdot d_L(L(x)) \leq d_{L'}(L'(x)) < k(c+1)$. By dividing both sides by k we obtain $d_L(L(x)) < c + 1$. Since we chose x arbitrarily, $cw_L(G) \leq c$ and the result follows.

The above proof shows that cutwidth is NP-complete even on split graphs where each vertex of I of the split partition (C, I) has degree 2.

4 Cutwidth of Threshold Graphs

In this section we give an algorithm that computes the cutwidth of threshold graphs in linear time. This algorithm constructs a layout $L =< v_1, v_2, \ldots, v_n >$ by appending at step i the vertex v_i that minimizes the cut $\delta(V_{i-1} \cup \{v_i\})$. For a given graph $G = (V, E)$ and a set $S \subseteq V$, we define the *rank* of a vertex v with respect to S to be $rank_S^G(v) = |N_G(v) \cap (V \setminus S)| - |N_G(v) \cap S|$ (superscript G is omitted when not needed). Observe that if $v \notin S$ then $d(S \cup \{v\}) = d(S) + rank_S(v)$. At step i, we select a vertex of $V \setminus V_{i-1}$ of lowest rank with respect to V_{i-1}. If there is a tie, the algorithm picks a vertex with the highest degree. If there still is a tie, the algorithm picks the vertex with the smallest label. The intuition behind the highest-degree tie-breaking is that when we add v to S, the ranks of all v's neighbors with respect to S decrease by 2, while the ranks of v's non-neighbors remain unchanged. Since we want the rank of the vertices we pick to be as small as possible, it is good to decrease the rank of as many vertices as possible. The details of the algorithm called MinCut are given below.

```
Algorithm: MinCut
Input: A graph $G = (V, E)$.
Output: A layout $L =< v_1, v_2, \cdots, v_n >$ of $G$
$V_0 := \emptyset$;
for i = 1 to $n$ do
    $v_i :=$ the vertex in $V \setminus V_{i-1}$ with smallest label;
    for every vertex $v$ in $V \setminus V_{i-1}$ ordered by increasing label do
        if $rank_{V_{i-1}}(v) < rank_{V_{i-1}}(v_i)$ then $v_i := v$
        else if $rank_{V_{i-1}}(v) = rank_{V_{i-1}}(v_i)$ and $\Delta(v) > \Delta(v_i)$ then $v_i := v$
    $V_i := V_{i-1} \cup \{v_i\}$;
    $L(v_i) := i$;
```

Before reaching the details of why Algorithm MinCut produces optimal layouts when the input is a threshold graph, we need to study how the layouts produced by the algorithm look. Observe first that if G has isolated vertices, then these can be placed in arbitrary positions in any optimal cutwidth layout, and our algorithm places them in the beginning of the output layout. We assume that G has been labeled in a manner such that every vertex in I has smaller label than every vertex in C, for every pair u and v of vertices in I, $level(u) < level(v)$ implies $label(u) < label(v)$ and for every pair u and v of vertices in C, $level(u) < level(v)$ implies $label(u) < label(v)$. This can be achieved through an $O(n)$-time preprocessing step, using the threshold partition of G.

For the statements of the following results in this section, we let $L =< v_1, \ldots, v_n >$ be the layout computed by Algorithm MinCut when run on a threshold graph G with threshold partition $(C, I) = ((C_1, \ldots, C_\ell), (I_1, \ldots, I_\ell))$.

Given two vertices u and v of G such that $u \in I$ and $v \in C$, we define the vertex set $over(u, v)$ to contain all vertices $x \in I$ such that $label(x) < label(u)$

and all vertices $y \in C$ such that $label(y) < label(v)$. The essence of the following lemma is that the algorithm picks vertices at lower levels before proceeding to higher levels, and that it starts with a vertex of I.

Lemma 2. *For any $i \in \{1, \ldots, n\}$ and $k \in \{1, \ldots, \ell\}$:*

(a) If $V_i \cap I_k \neq \emptyset$ then $I_{k'} \subseteq V_i$, for every $1 \leq k' < k$.
(b) If $V_i \cap C_k \neq \emptyset$ then $C_{k'} \subseteq V_i$, for every $1 \leq k' < k$.
(c) If $V_i \cap I_k = \emptyset$ then $V_i \cap C_k = \emptyset$.

As a direct consequence of Lemma 2 (a) and (b), for any i between 1 and n, if $u \in I \cap V_i$ and $v \in C \cap V_i$ then $over(u, v) \subseteq V_i$.

Lemma 3. *Let $u \in I$ and $v \in C$. Then $u <_L v$ if and only if u and v are non-adjacent or $rank_{over(u,v)}(u) < rank_{over(u,v)}(v)$.*

We say that the algorithm *covers* a vertex set S if there is an index i such that $V_i = S$. If the algorithm covers S then we also say that S is covered.

Lemma 4. *Let $u, u' \in I$ with $label(u') = label(u) + 1$, and let $v, v' \in C$ with $label(v') = label(v) + 1$. Then $over(u', v')$ is covered if and only if $u <_L v'$ and $v <_L u'$.*

We are now equipped with most of the tools that are necessary to work with layouts produced by Algorithm MinCut. All that remains before we move on to proving the correctness of the algorithm are a couple of simple observations.

Observation 5. *For each integer $i \leq n - 1$, $rank_{V_i}(v_{i+1}) \geq rank_{V_{i-1}}(v_i) - 2$. Furthermore, if $\Delta(v_{i+1}) > \Delta(v_i)$ then $rank_{V_i}(v_{i+1}) \geq rank_{V_{i-1}}(v_i) - 1$*

Observation 6. *For every level $k \leq \ell$ and every triple of vertices $u, v \in C_k$ and $w \notin C_k$, $u <_L w$ if and only if $v <_L w$.*

4.1 Correctness of Algorithm MinCut

In this subsection, we show that Algorithm MinCut produces optimal layouts when the input is a threshold graph. We assume for contradiction that there is a threshold graph $G = (V, E)$ with threshold partition $((C_1, \ldots, C_\ell), (I_1, \ldots, I_\ell))$ on which Algorithm MinCut outputs layout $L =< v_1, \ldots, v_n >$ such that $cw_L(G) > cw(G)$. We call such a threshold graph a *counterexample*, and we say that a counterexample is *minimal* if it is has the smallest value of $|V| + |E|$ among all counterexamples. A *bad set* of counterexample G is a set $S \subseteq V$ that is covered by the algorithm and for which $d(S) > cw(G)$. A *locally worst* bad set is a bad set $S = V_i$ such that $d(S) \geq d(V_{i+1})$ and $d(S) \geq d(V_{i-1})$. Observe that $rank_{V_{i-1}}(v_i)$ must be non-negative and $rank_{V_i}(v_{i+1})$ must be non-positive for V_i to be locally worst. Observation 5 then implies that $rank_{V_{i-1}}(v_i)$ is 2, 1 or 0. This means that if $v_i \in C_1$ then i is $\lfloor \frac{n}{2} \rfloor$, $\frac{n}{2}$ or $\lceil \frac{n}{2} \rceil$ respectively. Another thing to notice about locally worst bad sets is that if both V_i and V_{i-1} are bad sets with $d(V_{i-1}) \geq d(V_i)$ then some locally worst bad set is a strict subset of V_i.

The main idea of the proof is to show that if there is a counterexample G, then there must be another counterexample G' that either has at most 2 levels, or exactly 3 levels and a very specific structure. We complement this result by showing that the algorithm produces optimal layouts on all graphs with at most 2 levels, and on all graphs with 3 levels and the mentioned structural properties. This yields that $cw_L(G) = cw(G)$ for every threshold graph G.

Lemma 7. *Let G be a threshold graph on exactly 1 level. Then $cw_L(G) = cw(G)$.*

Proof. Let (C, I) be a threshold partition of G. Observe that every vertex of I is adjacent to every vertex of C. The algorithm lays out $\lfloor \frac{|I|}{2} \rfloor$ vertices of I, then all of C, followed by the remaining vertices of I. By inspection, $cw_L(G) = \lfloor \frac{n}{2} \rfloor \cdot \lceil \frac{n}{2} \rceil - \lfloor \frac{|I|}{2} \rfloor \cdot \lceil \frac{|I|}{2} \rceil$. Since all non-edges of G are between vertices in I, $bw(G) = \lfloor \frac{n}{2} \rfloor \cdot \lceil \frac{n}{2} \rceil - \lfloor \frac{|I|}{2} \rfloor \cdot \lceil \frac{|I|}{2} \rceil$. Thus $cw_L(G) = bw(G) \leq cw(G) \leq cw_L(G)$ and $cw_L(G) = cw(G)$ follows.

Already for threshold graphs with 2 levels, the correctness proof for Algorithm MinCut is more complicated. Before we go on to this proof we need more tools to work with locally worst bad sets.

For the statements of all the remaining results and definitions in this section, whenever we mention a counterexample G, we let $((C_1, \ldots, C_\ell), (I_1, \ldots, I_\ell))$ be is its threshold partition. The output of Algorithm MinCut is always denoted by $L =< v_1, \ldots, v_n >$.

Lemma 8. *Every locally worst bad set S of a counterexample G satisfies (i) $C_1 \cap S \neq \emptyset$, (ii) $I_1 \subseteq S$, (iii) $C_\ell \cap S = \emptyset$, and (iv) $I_\ell \setminus S \neq \emptyset$.*

Lemma 9. *Let G be a threshold graph on exactly 2 levels. Then $cw_L(G) = cw(G)$.*

From this it follows that any counterexample has at least 3 levels. Over the next few lemmas, we show how any counterexample can be transformed into a counterexample with exactly 3 levels. The first observation is that in every minimal counterexample all parts of the graph participate in making the graph a counterexample.

Definition 1. *A counterexample has the* extremal property *if it has a bad set S such that $I \setminus I_\ell \subseteq S$ and $S \cap C \subseteq C_1$. In this case, S is called an* extremal bad set.

Lemma 10. *Every minimal counterexample G has the extremal property, and every locally worst bad set of G is an extremal bad set.*

Lemma 11. *If there is a counterexample, then there is a counterexample with the extremal property and at most 3 levels.*

Proof. We start by showing that if there is a counterexample G on at least 4 levels with an extremal bad set S such that the sets $I \cap S$, $I \setminus S$, $C \cap S$ and $C \setminus S$ are non-empty, then there is a counterexample G' with $G \subset G'$, such that (C, I) is a threshold partition of G' and S is an extremal bad set of G'.

Let u and v be the vertices in $I \cap S$ and $C \cap S$ with the highest labels, and let u' and v' be the vertices in $I \setminus S$ and $C \setminus S$ with the lowest labels respectively. Now, $S = \mathit{over}(u', v')$ so by Lemma 4 $u <_L v'$ and $v <_L u'$. We choose x to be the vertex of I_2 with highest label and y to be the vertex if C_3 with the lowest label. We add the edge xy to G to obtain a new threshold graph G'. (C, I) is a threshold partition of G' and $G \subset G'$. Furthermore $\delta^{G'}(S') = \delta(S) \cup \{xy\}$ so $d^{G'}(S) = d(S) + 1 > cw(G) + 1 \geq cw(G \cup \{xy\})$. Also, in G' u' is adjacent to all vertices of C and v is a universal vertex. Let L' be the layout L' produced by Algorithm MinCut when run on G'. To prove that S is an extremal bad set of G' it is sufficient to show that L' covers S. However, $S = \mathit{over}(u', v')$ both in G and G' and none of u, u', v and v' are incident to the new edge xy so $u <_{L'} v'$ and $v <_{L'} u'$. By Lemma 4 L' covers S.

We can now proceed to prove the lemma. Without loss of generality, G is a minimal counterexample. By Lemma 10 G has an extremal locally worst bad set S. By Lemma 8 the sets $I \cap S$, $I \setminus S$, $C \cap S$ and $C \setminus S$ are non-empty. Thus, if G has at most 3 levels we are done, otherwise by the discussion in the previous paragraph, there is a counterexample G' with $G \subset G'$, such that (C, I) is a threshold partition of G' and S is an extremal bad set of G'. If G' has at most 3 levels we are done, otherwise we can again apply the discussion above to G' and S to get yet another counterexample G'' with $G \subset G' \subset G''$, such that (C, I) is a threshold partition of G'' and S is an extremal bad set of G''. Reiterating this argument we can continue producing counterexamples on more and more edges. Since the clique is not a counterexample, this process must stop at some point. The graph at hand at this point is a counterexample with at most 3 levels and with S as an extremal bad set.

Definition 2. *A counterexample has the* super extremal property *if it has an extremal bad set S, such that either $I_\ell \cap S \neq \emptyset$ or $S \cap C \subset C_1$. Then S is called a* super extremal bad set*.*

Lemma 12. *There are no counterexamples with the super extremal property.*

Lemmas 10, 11, and 12 allow us to concentrate on counterexamples on exactly 3 levels with the extremal property, but without the super extremal property.

Definition 3. *We say that a counterexample with* 3 *levels and the extremal property has the* snake property *if (i) $u <_L v$, where u is the vertex of I_2 with highest label and v is the vertex of C_1 with highest label, and (ii) $u' <_L v'$, where u' is the vertex of I_3 with lowest label and v' is the vertex of C_2 with lowest label.*

Lemma 13. *If there is a counterexample with* 3 *levels with the extremal property then there is a counterexample with* 3 *levels with the extremal and the snake properties.*

Lemma 14. *In a counterexample with* 3 *levels with the extremal and snake properties, n is even, $|C|$ and $|I|$ are odd, $|C_1| = |C_2| + |C_3| + 1$, $|I_3| = |I_1| + |I_2| + 1$, and $|I_1| + |I_2| + |C_1| = \frac{n}{2}$.*

At this point all that remains is to analyze how a counterexample on 3 levels and the extremal and snake properties looks, and to show that in such a graph G, $cw_L(G) \leq bw(G) \leq cw(G)$. Now we are ready to show our main result.

Theorem 2. *For any threshold graph G, $cw_L(G) = cw(G)$.*

Proof. Suppose for contradiction that there is a counterexample. Then, by Lemmas 7, 9, 10, 11 and 13 there is a counterexample G on 3 levels with the snake and extremal properties. Since Lemma 12 implies that G does not have the super extremal property, $I_1 \cup I_2 \cup C_1$ is a bad set of G. By Lemma 14, n is even, $|C|$ and $|I|$ are odd, $|C_1| = |C_2| + |C_3| + 1$, $|I_3| = |I_1| + |I_2| + 1$, and $|I_1| + |I_2| + |C_1| = \frac{n}{2}$.

Let S be a vertex set on $\frac{n}{2}$ vertices that minimizes $d(S)$, that is, with $d(S) = bw(G)$. Notice that $\delta(S) = \delta(V \setminus S)$. Thus, without loss of generality $|S \cap I| > \frac{|I|}{2}$. We view the set S as a set of $\frac{n}{2}$ pebbles that have been placed on distinct vertices of G. We can move pebbles from I_3 to C_1 and keep optimality of S, unless one of the following is true: (i) There are no pebbles on vertices of I_3, (ii) All vertices of C_1 have pebbles on them, (iii) Moving a pebble from a vertex in I_3 to a vertex in C_1 increases $d(S)$.

Moving a pebble from a vertex x to a vertex y increases $d(S)$ by $rank_{S\setminus\{x\}}(y) - rank_{S\setminus\{x\}}(x)$. Thus, if there is a pebble on a vertex x in I_3 and a free spot on a vertex y in C_1, moving a pebble from x to y does not increase $d(S)$ if and only if $rank_{S\setminus\{x\}}(y) - rank_{S\setminus\{x\}}(x) = 1 - (|C \setminus S| - |C \cap S|) \leq 0$. Rearranging terms yields that moving a pebble from x to y does not increase $d(S)$ if and only if $|C \cap S| < |C \setminus S|$. In addition, one should notice that if there are no pebbles on vertices of I_3 then $|C \cap S| > |C \setminus S|$ because $|I_3| = |I_1| + |I_2| + 1$. Similarly, if all vertices of C_1 have pebbles on them then $|C \cap S| > |C \setminus S|$ because $|C_1| = |C_2| + |C_3| + 1$.

By our choice of S, before we start moving any pebbles, $|I \cap S| > |I \setminus S|$. Since $|S| = \frac{n}{2}$ this means that $|C \cap S| < |C \setminus S|$. Therefore, by the discussion in the previous paragraph we can move pebbles from I_3 to C_1, preserving minimality of $d(S)$ until the inequality flips from $|C \cap S| < |C \setminus S|$ to $|C \cap S| > |C \setminus S|$. At this point, $|C \cap S| = \lceil \frac{|C|}{2} \rceil = |C_1|$ and $|I \cap S| = |I_1| + |I_2|$. Let α, β and γ be the ranks of a vertex in I_1, I_2 and I_3 with respect to S.

If $\alpha \leq \gamma$ and $\beta \leq \gamma$ we can move pebbles from I_3 to I_1 and I_2 keeping optimality of S. Since exactly $|I_1| + |I_2|$ pebbles are placed on vertices of I, after the move every vertex of $I_1 \cup I_2$ has a pebble on it, and no pebbles are on vertices in I_3. Now we can safely move all pebbles in C_2 and C_3 to C_1, keeping optimality of S. Since exactly $|C_1|$ pebbles are placed on vertices of C, after the move every vertex of C_1 has a pebble on it, and no pebbles are on vertices in $C_2 \cup C_3$. But this means that $S = I_1 \cup I_2 \cup C_1$ and $d(S) \leq bw(G) \leq cw(G)$ contradicting that $I_1 \cup I_2 \cup C_1$ is a bad set of G.

If $\alpha \geq \gamma$ and $\beta \geq \gamma$ we can move all pebbles from I_1 and I_2 to I_3 keeping optimality of S. Since exactly $|I_1| + |I_2|$ pebbles are placed on vertices of I, after this move all but one vertex of I_3 has a pebble on it, and no pebbles are on vertices in $I_1 \cup I_2$. Now we can safely move pebbles in C_1 to C_2 and C_3, keeping optimality of S. Since exactly $|C_1|$ pebbles are placed on vertices of C, after this move exactly one vertex of C_1 has a pebble on it, and all vertices of $C_2 \cup C_3$ have pebbles on them. Let x be the vertex in I_3 without a pebble and y be the vertex in C_1 with a pebble. After the moves, $rank_{S\setminus\{y\}}(x) = rank_{S\setminus\{y\}}(y) = 1$, so we can move the pebble on y to x, keeping optimality of S. However $d(I_1 \cup I_2 \cup C_1) = d(V \setminus S) = d(S) \leq bw(G)$ contradicting that $I_1 \cup I_2 \cup C_1$ is a bad set of G.

If $\alpha \leq \gamma \leq \beta$ we can move pebbles from I_2 and I_3 to I_1 maintaining optimality until each vertex of I_1 has a pebble on it. If any pebbles remain in I_2 we can move these pebbles to I_3. Since there are $|I_1| + |I_2|$ pebbles in I there are exactly $|I_2|$ vertices in I_3 that have pebbles on them. Now, we can move all pebbles in C_2 to C_3 and C_1 maintaining optimality until there are no pebbles left in C_2. If $|I_1| \geq |I_2|$ we can move all pebbles from C_3 to C_1 maintaining optimality. After this move, the set of vertices in C with pebbles on them is exactly C_1. Thus we can move all pebbles in I_3 to I_2 maintaining optimality. In this case, $S = I_1 \cup I_2 \cup C_1$, but $d(S) \leq bw(G)$ contradicting that $I_1 \cup I_2 \cup C_1$ is a bad set of G.

If $|I_1| < |I_2|$ we can move pebbles from C_1 to C_3 until all vertices of C_3 have pebbles on them. After this move there are exactly $|C_1| - |C_3|$ pebbles on vertices in C_1. We consider $d(S)$ and compare it to $d(I_1 \cup I_2 \cup C_1) = |C_1|(|C_2| + |C_3|) + |I_2||C_2| + |I_3||C_1|$. Counting the edges of $d(S)$ we obtain $d(S) = |C_1|(|C_2| + |C_3|) + |I_1||C_3| + |I_2|(|C_1| - |C_3|) + (|I_3| - |I_2|)|C_1| + |I_2|(|C_2| + |C_3|)$. Simplifying yields $d(S) = |C_1|(|C_2| + |C_3|) + |I_1||C_3| + |I_3||C_1| + |I_2||C_2|$. But this means that $d(I_1 \cup I_2 \cup C_1) < d(S) \leq bw(G)$ contradicting that S is a bad set of G.

Finally, suppose $\alpha \geq \gamma \geq \beta$. Since $d(S) = d(V \setminus S)$ we can move all pebbles over to vertices that do not have pebbles and preserve optimality. There are now exactly $|I_3|$ pebbles in I and $|C_1| + |C_2|$ pebbles in $|C|$. Since $|I_3| > |I_1| + |I_2|$ there is a pebble on a vertex x in I_3. Also, since $|C_1| > |C_2| + |C_3|$ there is a vertex y with no pebble in C_1. At this point, $rank_{S\setminus\{x\}}(x) = rank_{S\setminus\{x\}}(y) = 1$, so we can move a pebble from x to y and again obtain a set S with pebbles on $|I_1|+|I_2|$ vertices in I and $|C_1|$ vertices in C. In addition if α', β' and γ' are the ranks of a vertex in I_1, I_2 and I_3 with respect to S, we see that $\alpha' = -\alpha - 2$, $\beta' = -\beta - 2$ and $\gamma' = -\gamma - 2$, so at this point $\alpha' \leq \gamma' \leq \beta'$ and the discussion in the previous paragraphs applies. This concludes the proof.

Theorem 3. *The cutwidth of a threshold graph on n vertices can be computed in $O(n)$ time.*

Proof. We describe an implementation of Algorithm MinCut that runs in $O(n)$ time. Let $(C, I) = ((C_1 \cup C_2 \cdots \cup C_\ell), (I_1 \cup I_2 \cdots \cup I_\ell))$ be the threshold partition of the input graph. By Lemma 2 we know that Algorithm MinCut picks vertices of I by increasing label and the vertices of C by increasing label. Therefore we keep track of the not yet picked vertices $u \in I$ and $v \in C$ with the lowest labels. We also keep track of the ranks of u and v with respect to $over(u, v)$, that is $r_u = rank_{over(u,v)}(u)$ and $r_v = rank_{over(u,v)}(u)$. At each step of the algorithm we pick the one of u and v with the lowest rank (and highest degree if their rank is equal, lowest label if both rank and degree is equal). We now need to update the variables u, v, r_u and r_v. Lemma 2 guarantees that u and v are adjacent, so if we pick u we reduce r_v by 2 and if we pick v we reduce both r_u and r_v by 2. Finally, if we picked u, we need to correct r_u for the fact that the next vertex in I could be in a higher level, and similarly we need to correct r_v. If the algorithm picked u, let u' be the vertex in I with $label(u) + 1 = label(u')$. If $level(u') > level(u)$ we increase r_u by $|C_{level(u')}|$ because Lemma 2 guarantees that no vertices of $C_{level(u')}$ have been picked yet. Similarly, if the algorithm picked v, let v' be the vertex in C with $label(v)+1 = label(v')$. If $level(v') > level(v)$ we increase r_v by $|I_{level(v)}|$ because Lemma 2 guarantees that all vertices of $I_{level(v)}$ have already been picked. For each new vertex to be picked the algorithm does $O(1)$ work so the total time complexity is $O(n)$.

5 Concluding Remarks: Cutwidth of Interval Graphs

A natural open question and a future research direction is resolving the computational complexity of cutwidth on interval graphs. A graph is *interval* if sets of consecutive integers (intervals) can be assigned to its vertices such that two vertices are adjacent if and only if their intervals overlap. Some inherently difficult graph problems, like bandwidth, are polynomially solvable on interval graphs [15], whereas others, like optimal linear arrangement, are NP-complete [7]. Optimal linear arrangement can be seen as sum-bandwidth or sum-cutwidth, equivalently (see [10] for definitions). Simple examples exist to show that Algorithm MinCut can produce a layout with cutwidth that is a factor of $O(n)$ larger than $cw(G)$ when G is an interval graph, or even a proper interval graph. An interval graph is *proper interval* if it has an interval model where no interval properly contains another. Interestingly, for proper interval graphs a trivial $O(n)$-time approach solves the cutwidth problem:

Theorem 4. *Let G be a proper interval graph and let L be an ordering of the vertices of G by increasing right endpoint of their corresponding intervals. Then $cw_L(G) = cw(G)$.*

Note that an increasing right endpoint order is not necessarily an optimal layout for a threshold graph; a star is a simple counterexample. Finally, even the cutwidth of trivially perfect graphs seems to be an interesting open problem.

References

1. Adolphson, D., Hu, T.C.: Optimal linear ordering. SIAM J. Appl. Math. 25, 403–423 (1973)
2. Arora, S., Frieze, A., Kaplan, H.: A new rounding procedure for the assignment problem with applications to dense graphs arrangements. In: Proceedings of FOCS 1996, pp. 21–30. IEEE, Los Alamitos (1996)

3. Blin, G., Fertin, G., Hermelin, D., Vialette, S.: Fixed-parameter algorithms for protein similarity search under RNA structure constraints. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 271–282. Springer, Heidelberg (2005)
4. Botafogo, R.A.: Cluster analysis for hypertext systems. In: Proceedings of SIGIR 1993, pp. 116–125. ACM, New York (1993)
5. Brandstädt, A., Le, V.B., Spinrad, J.: Graph Classes: A Survey. SIAM, Philadelphia (1999)
6. Chung, M.J., Makedon, F., Sudborough, I.H., Turner, J.: Polynomial time algorithms for the min cut problem on degree restricted d trees. In: Proceedings of FOCS 1982, pp. 262–271. IEEE, Los Alamitos (1982)
7. Cohen, J., Fomin, F.V., Heggernes, P., Kratsch, D., Kucherov, G.: Optimal linear arrangement of interval graphs. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 267–279. Springer, Heidelberg (2006)
8. Gavril, F.: Some NP-complete problems on graphs. In: 11th Conference on Information Sciences and Systems, pp. 91–95. John Hopkins University, Baltimore (1977)
9. Diaz, J., Penrose, M., Petit, J., Serna, M.: Approximating layout problems on random geometric graphs. Journal of Algorithms 39, 78–117 (2001)
10. Diaz, J., Petit, J., Serna, M.: A survey of graph layout problems. ACM Computing Surveys 34, 313–356 (2002)
11. Földes, S., Hammer, P.L.: Split graphs. Congressus Numerantium 19, 311–315 (1977)
12. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs, 2nd edn. Annals of Discrete Mathematics, vol. 57. Elsevier, Amsterdam (2004)
13. Heggernes, P., Papadopoulos, C.: Single-edge monotonic sequences of graphs and linear-time algorithms for minimal completions and deletions. In: Lin, G. (ed.) COCOON 2007. LNCS, vol. 4598, pp. 406–416. Springer, Heidelberg (2007)
14. Junguer, M., Reinelt, G., Rinaldi, G.: The traveling salesman problem. In: Handbook on Operations Research and Management Sciences, 7th edn., pp. 225–330. North-Holland, Amsterdam (1995)
15. Kleitman, D.J., Vohra, R.V.: Computing the bandwidth of interval graphs. SIAM J. Disc. Math. 3, 373–375 (1990)
16. Karger, D.R.: A randomized fully polynomial approximation scheme for all terminal network reliability problem. In: Proceedings of STOC 1996, pp. 11–17. ACM, New York (1996)
17. Leighton, F.T., Rao, S.: An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In: Proceedings of FOCS 1988, pp. 422–431. IEEE, Los Alamitos (1988)
18. Mahadev, N., Peled, U.: Threshold graphs and related topics. In: Annals of Discrete Mathematics, vol. 56. North Holland, Amsterdam (1995)
19. Makedon, F., Sudborough, I.H.: Minimizing width in linear layouts. In: Díaz, J. (ed.) ICALP 1983. LNCS, vol. 154, pp. 478–490. Springer, Heidelberg (1983)
20. Monien, B., Sudborough, I.H.: Min cut is NP-complete for edge weighted trees. In: Kott, L. (ed.) ICALP 1986. LNCS, vol. 226, pp. 265–274. Springer, Heidelberg (1986)
21. Mutzel, P.: A polyhedral approach to planar augmentation and related problems. In: Spirakis, P.G. (ed.) ESA 1995. LNCS, vol. 979, pp. 497–507. Springer, Heidelberg (1995)
22. Thilikos, D.M., Serna, M.J., Bodlaender, H.L.: Cutwidth I: A linear time fixed parameter algorithm. Journal of Algorithms 56, 1–24 (2005)
23. Thilikos, D.M., Serna, M.J., Bodlaender, H.L.: Cutwidth II: Algorithms for partial w-trees of bounded degree. Journal of Algorithms 56, 24–49 (2005)
24. Yannakakis, M.: A polynomial algorithm for the min cut linear arrangement of trees. Journal of ACM 32, 950–988 (1985)

The Rank-Width of the Square Grid

Vít Jelínek*

Department of Applied Mathematics, Charles University
Malostranské nám. 25, 118 00 Praha, Czech Republic
jelinek@kam.mff.cuni.cz

Abstract. Rank-width is a graph width parameter introduced by Oum and Seymour. It is known that a class of graphs has bounded rank-width if and only if it has bounded clique-width, and that the rank-width of G is less than or equal to its branch-width.

The $n \times n$ *square grid*, denoted by $G_{n,n}$, is a graph on the vertex set $\{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$, where a vertex (x, y) is connected by an edge to a vertex (x', y') if and only if $|x - x'| + |y - y'| = 1$.

We prove that the rank-width of $G_{n,n}$ is equal to $n - 1$, thus solving an open problem of Oum.

Keywords: rank-width, grid graph.

1 Basic Terminology

A *cut* in a graph $G = (V, E)$ is a partition of its vertex set into two nonempty disjoint parts X, Y. The two sets X and Y will be called *the parts* of the cut. A set of vertices is called *monochromatic* if it is a subset of X or a subset of Y.

The *adjacency matrix* of a cut (X, Y) is the 0-1 matrix whose rows are indexed by X, columns are indexed by Y, and the entry in row i and column j is equal to 1 iff i and j are connected by an edge of G. The *rank* of a cut is the $\mathbb{Z}_2$-rank of its adjacency matrix.

The *rank-decomposition* of a graph G is a ternary tree T whose leaves correspond bijectively to the vertices of G. Every edge in the decomposition defines a cut, whose two parts correspond to the leaves of the two subtrees determined by the edge. The *weight* of the edge is the rank of the associated cut. The weight of the decomposition is the maximum weight of its edges. An optimum decomposition of G is a rank-decomposition with the smallest possible weight. The *rank-width* of G, denoted by $\operatorname{rwd}(G)$, is the weight of its optimum rank-decomposition.

Rank-width has been introduced by Oum and Seymour [6] in 2004 (see also [2]). It is known that a family of graphs has bounded rank-width if and only if it has bounded clique-width, and that the rank-width of a graph G does not exceed its branch-width [5]. A summary of recent results on rank-width, as well as other related graph parameters, can be found in the survey by Hliněný et al. [1].

* Supported by project MSM0021620838 of the Czech Ministry of Education, and by grant 201/05/H014 of the Czech Science Foundation.

Let $G_{m,n}$ denote the grid with m rows and n columns, defined as a graph on the vertex set $\{1,\dots,m\} \times \{1,\dots,n\}$, where a vertex (x,y) is connected by an edge to (x',y') if and only if $|x-x'|+|y-y'|=1$. In this paper, we determine the rank-width of the $n \times n$ grid $G_{n,n}$, thus solving an open problem raised by Oum [4]. Previous results [3] have established the inequalities $\lceil 2n/3 \rceil \leq \mathrm{rwd}(G_{n,n}) \leq n-1$. We close the gap by providing the lower bound $\mathrm{rwd}(G_{n,n}) \geq n-1$, thus proving the following main result.

Theorem 1. *The square grid $G_{n,n}$ has rank-width equal to $n-1$.*

Throughout this paper, we let V_n denote the vertex set of $G_{n,n}$.

2 The Proof

The basis of our approach is to estimate the rank of a cut (X,Y) using the size of a matching with suitable properties. To make this specific, we need more terminology. Let G be a graph, let (X,Y) be a cut. An (X,Y)-*edge* is an edge of G that connects a vertex in X to a vertex in Y. An (X,Y)-*matching* is a set of pairwise disjoint (X,Y)-edges. If the cut (X,Y) is clear from the context, we will use the term 'cut-edge' and 'cut-matching'.

To an (X,Y)-matching M, we associate an *edge-adjacency graph* A_M, which is a directed graph whose vertices correspond to the edges of M, and if $e=\{x,y\}$ and $e'=\{x',y'\}$ are two distinct edges of M such that $x,x' \in X$ and $y,y' \in Y$ then A_M has a directed edge from e to e' if and only if y is adjacent to x' in G. We say that a cut-matching is *acyclic* if its edge-adjacency graph does not contain any directed cycle (not even a directed cycle of length two).

All our lower-bounds on ranks of cuts are based on the following lemma.

Lemma 1. *The rank of a cut (X,Y) in a graph G is greater than or equal to the size of the largest acyclic (X,Y)-matching.*

Proof. Let $M=\{e_1,\dots,e_k\}$ be an acyclic cut-matching. Assume that the edges in M are ordered in such a way that if $i>j$ then there is no directed edge from e_i to e_j. Such an ordering exists since the edge-adjacency graph is acyclic. The $2k$ vertices covered by the matching M then induce in the incidence matrix of (X,Y) a $k \times k$ submatrix which (after an appropriate reordering of rows and columns) has all the diagonal entries equal to 1, and all the entries above the main diagonal equal to 0. This matrix is regular, showing that the rank of the cut is at least k. ☐

To prove our main result, we need to find sufficient conditions that guarantee that a cut has a large acyclic cut-matching. The first step in this direction is the following lemma.

Lemma 2. *Let $G=G_{m,n}$ be a grid with m rows and n columns, and assume that $m<n$. Let (X,Y) be a cut in G with the property that no row of G is monochromatic. Then G has an acyclic (X,Y)-matching of size m.*

We omit the proof of this lemma, due to space constraints.

Corollary 1. *Let (X, Y) be a cut in $G = G_{n,n}$ such that at most one row of G is monochromatic. Then the cut (X, Y) has an acyclic matching of size $n - 1$.*

Proof. By removing a single row from G, we may obtain two rectangular grids (one of them may be empty) that together have $n - 1$ rows, and none of these rows is monochromatic. Applying Lemma 2 to each of these grids, we obtain two acyclic matchings that have together $n - 1$ edges. The union of these two matchings is easily seen to be an acyclic matching in G. $\square$

Of course, the corollary above holds even when 'row' in the statement is replaced with 'column'.

Let T be an optimum rank-decomposition of $G_{n,n}$. Each edge of T determines a cut (X, Y). If the cut has at most one monochromatic row, or at most one monochromatic column, then its rank is at least $n - 1$ by Corollary 1, hence the rank-width of $G_{n,n}$ is at least $n - 1$ as well, and we are done.

Assume now, that for every edge, the corresponding cut (X, Y) has at least two monochromatic rows as well as at least two monochromatic columns. Necessarily, all the monochromatic rows and columns then belong to the same part of the cut. This motivates the following definition.

Definition 1. *Let X be a set of vertices of $G_{n,n}$. We say that X is* large *if X contains the union of two rows and two columns of $G_{n,n}$.*

Clearly, at most one part of a cut can be large. On the other hand, by Corollary 1, a cut with no large part has rank at least $n - 1$.

Let T be again an optimum rank-decomposition of $G_{n,n}$. Assume that every edge of T determines a cut that has a large part. Let (X, Y) be the cut corresponding to an edge η of T. We will turn η into a directed edge pointing towards the component of $T - \eta$ whose leaves form the large set. Each edge of T has a well defined direction. Since T is a tree, it has no directed cycle, hence it must contain a vertex v of outdegree 0. Such a vertex cannot be a leaf, since a singleton set is never large. The three components of $T - v$ determine a partition of V_n into three disjoint sets X, Y, Z, with the property that the union of each two of these sets is a large set. Thus, to prove Theorem 1, it suffices to prove the following proposition, which implies that at least one edge adjacent to v has width at least $n - 1$.

Proposition 1. *Assume that the vertex set of $G_{n,n}$ is partitioned into three nonempty disjoint sets X, Y, Z, and that the union of any two of these sets is a large set. Then at least one of the three cuts $(X, Y \cup Z)$, $(Y, X \cup Z)$ and $(Z, X \cup Y)$ has an acyclic matching of size at least $n - 1$.*

The rest of this paper is devoted to the proof of Proposition 1.

From now on, let us write G instead of $G_{n,n}$. For a set $W \subseteq V_n$, let $G[W]$ denote the subgraph of G induced by W. A matching whose edges all belong to the cut $(W, V_n \setminus W)$ will be called *a matching adjacent to W*.

Our proof of Proposition 1 proceeds by contradiction. Assume that we are given three sets X, Y, Z satisfying the assumptions of Proposition 1, and that none of the three sets has an adjacent acyclic matching of size $n-1$. From now on, the three sets X, Y, Z will be called *parts*, and we will use the term *cut-edge* to refer to any edge whose endpoints belong to two distinct parts.

We will also use the term *component of* G to refer to a connected component of any of the three graphs $G[X], G[Y], G[Z]$. Among all possible counterexamples to Proposition 1, let us choose a counterexample X, Y, Z with the smallest number of components.

Lemma 3. *In a minimal counterexample described above, the three graphs* $G[X \cup Y], G[X \cup Z]$ *and* $G[Y \cup Z]$ *are all connected.*

Proof. Let us prove the lemma for $G[X \cup Y]$. Since $X \cup Y$ is large, $G[X \cup Y]$ must have a (unique) connected component whose vertex set is large. Assume that $G[X \cup Y]$ has another connected component C. Let us remove all the vertices of C from the sets X, Y and add them to Z. Let X_{NEW}, Y_{NEW}, and Z_{NEW} denote the sets obtained from X, Y and Z by this modification.

The three new sets still satisfy the assumptions of Proposition 1. Furthermore, any acyclic matching adjacent to X_{NEW}, Y_{NEW} or Z_{NEW} is also an acyclic matching adjacent to X, Y or Z. Thus $(X_{\mathrm{NEW}}, Y_{\mathrm{NEW}}, Z_{\mathrm{NEW}})$ is a counterexample to Proposition 1, and it has fewer components than (X, Y, Z). □

The four vertices of degree 2 in G will be called *corners*. The components containing the corners will be called *corner components*. Notice that no two corners can belong to the same component, because the complement of every component must be a large set. We will draw the graph G in the plane in the natural way, with the vertex (a, b) drawn as the point with Cartesian coordinates (a, b), and the edges drawn as horizontal or vertical unit segments. Thus, the vertex $(1, 1)$ is the bottom-left corner, and $(n, 1)$ is the bottom-right corner.

Let $v_{\llcorner}$, $v^{\ulcorner}$, $v_{\lrcorner}$ and $v^{\urcorner}$ be the four corner vertices $(1, 1)$, $(1, n)$, $(n, 1)$ and (n, n), and let $C_{\llcorner}$, $C^{\ulcorner}$, $C_{\lrcorner}$ and $C^{\urcorner}$ be the corresponding corner components.

Our main tool is the following technical lemma, whose proof is omitted due to space constraints.

Lemma 4. *Let* X, Y, Z *be a partition of* V_n, *such that the union of any two parts is large and induces a connected subgraph of* G. *Let* P *be a path in* G *that connects the vertex* $v_{\llcorner}$ *to the vertex* $v^{\urcorner}$. *Let* d_1 *and* d_2 *be two vertices of* P *(we will call them "defects"), and assume that the following conditions hold:*

– *All the vertices of* P*, except possibly the two defects, belong to* $X \cup Y$.
– *No vertices of* P*, except possibly the two defects, belong to* $C^{\ulcorner} \cup C_{\lrcorner}$. *In other words, the path* P *connects two diagonally opposite corners, while avoiding (up to the defects) the remaining two corner components.*

Under these assumptions, at least one of the two sets X, Y *has an adjacent acyclic matching of size* $n-1$.

The following simple criterion will be helpful in proving acyclicity of cut-matchings.

Lemma 5. *Let $W \subseteq V_n$ be a set of vertices, and let M be a matching adjacent to W. If the edge-adjacency graph A_M has no directed cycle of length two, and if all the vertical edges in M have a vertex of W below a vertex from $V_n \setminus W$, then M is acyclic.*

Proof. Assume for contradiction that the edge-adjacency graph A_M has a directed cycle C of length $k > 2$, whose vertices are the edges $\{e_1, \dots, e_k\} \subseteq M$, indexed in such a way that the edge-adjacency graph has an arc from e_i to e_{i+1} for every $i \in \{1, \dots, k\}$ (we evaluate the indices modulo k whenever appropriate). Let us write $e_i = \{x_i, y_i\}$ with $x_i \in W$ and $y_i \in V_n \setminus W$.

Let C^* be a cycle in G of length $2k$ formed by the edges $\{x_i, y_i\}$ and $\{y_i, x_{i+1}\}$, for $i \in \{1, 2, \dots, k\}$. Let us direct these edges from x_i to y_i and from y_i to x_{i+1}, so that C^* becomes a directed cycle. Assume without loss of generality that C^* is directed clockwise in our fixed drawing G.

Let c be the rightmost column of G intersected by C^*. Since A_M has no cycle of length two, we may easily see that c must contain at least one vertical edge of $M \cap C^*$. Since C^* is directed clockwise, this edge must be directed towards the bottom, which means that it has a vertex from W at the top, contradicting our assumptions about M. $\square$

For $i \in \{2, \dots, 2n\}$, let D_i be the set $\{(a, b) \in V_n \colon a + b = i\}$. We will call D_i *the i-th decreasing diagonal.* Similarly, for $i \in \{1 - n, \dots, 0, \dots, n - 1\}$ we define the i-th increasing diagonal $I_i = \{(a, b) \in V_n \colon a - b = i\}$.

Definition 2. *For a corner component $C \in \{C_{\llcorner}, C^{\urcorner}\}$ the* height *of C is the number of decreasing diagonals intersected by C. For a corner component $C \in \{C^{\ulcorner}, C_{\lrcorner}\}$, the height of C is defined as the number of increasing diagonals that intersect C.*

The notion of height is motivated by the following Lemma.

Lemma 6. *A corner component C of height k is adjacent to an acyclic matching of size k.*

Proof. Let us prove the lemma for the component $C_{\llcorner}$, the other cases are symmetric. See Fig. 1. Assume that $C_{\llcorner}$ has height k, i.e., it intersects k decreasing diagonals D_2, D_3,..., D_{k+1}. Fix a vertex $(a, b) \in D_{k+1} \cap C_{\llcorner}$. For each $i = 1, \dots, a$, let x_i be the topmost vertex of $C_{\llcorner}$ in column i. Note that x_i does not belong to the topmost row, because the complement of $C_{\llcorner}$ is a large set. Let y_i be the vertex directly above x_i, and let e_i be the cut-edge $\{x_i, y_i\}$.

Similarly, for $j = 1, \dots, b-1$, let x'_j be the rightmost vertex in row j belonging to $C_{\llcorner}$, let y'_j be the vertex to the right of x'_j, and let e'_j be the edge connecting these two vertices. Define $M = \{e_1, \dots, e_a, e'_1, \dots, e'_{b-1}\}$.

Clearly $|M| = a + b - 1 = k$. Let us show that the edges in M are disjoint. Assume that e_i intersects e'_j in a vertex $v = (i, j)$. By construction, none of the

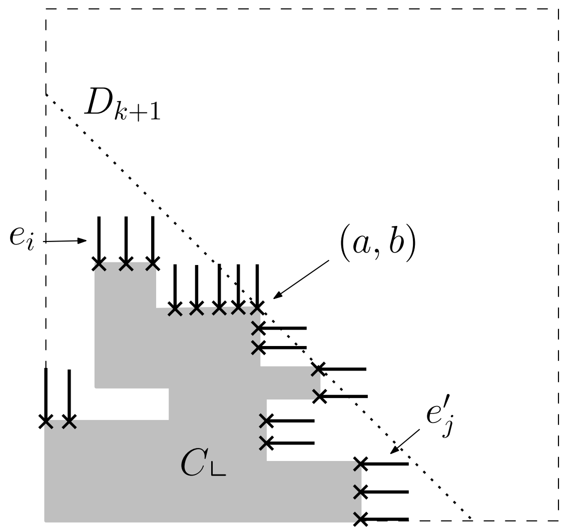


Fig. 1. Illustration of the proof of Lemma 6

vertices above v and none of the vertices to the right of v belong to $C_\llcorner$. Since the vertex (a,b) is to the right and above v, it can not belong to the same component as $v_\llcorner$, contradicting our assumptions. Thus, M is a matching adjacent to $C_\llcorner$. By Lemma 5, M is acyclic. □

Note that if C and C' are two distinct corner components which are both subsets of the same part ($C \cup C' \subseteq X$, say) and if M and M' are two acyclic matchings adjacent to C and C' respectively, then $M \cup M'$ large set, so the shortest path between C and C' has at least three edges. This means that an edge of M cannot intersect an edge of M', and in the edge-adjacency graph of the matching $M \cup M'$, there can be no arc between an edge of M and an edge of M'.

We now have all the necessary ingredients to prove Proposition 1. For a vertex $(a,b) \in V_n$, let $\chi(a,b) \in \{X,Y,Z\}$ be the part containing (a,b). We will now distinguish several cases, depending on the parts containing the four corner vertices. For convenience, we will represent the parts of the four corners by a matrix $T = \begin{pmatrix} \chi(v^\ulcorner) & \chi(v^\urcorner) \\ \chi(v_\llcorner) & \chi(v_\lrcorner) \end{pmatrix}$.

Up to symmetry and renaming of the parts, there are six possibilities for T: $\left(\begin{smallmatrix} X & Z \\ Y & X \end{smallmatrix}\right)$, $\left(\begin{smallmatrix} X & X \\ Y & Z \end{smallmatrix}\right)$, $\left(\begin{smallmatrix} Y & X \\ X & Y \end{smallmatrix}\right)$, $\left(\begin{smallmatrix} Y & Y \\ X & X \end{smallmatrix}\right)$, $\left(\begin{smallmatrix} X & X \\ X & Y \end{smallmatrix}\right)$, and $\left(\begin{smallmatrix} X & X \\ X & X \end{smallmatrix}\right)$. We deal with the first and the second type separately, and then we present an argument that works for the remaining four types.

The case $T = \left(\begin{smallmatrix} X & Z \\ Y & X \end{smallmatrix}\right)$. Since the graph $G[Y \cup Z]$ is connected, it contains a path P from $v_\llcorner$ to $v^\urcorner$, and we may apply Lemma 4.

The case $T = \left(\begin{smallmatrix} X & X \\ Y & Z \end{smallmatrix}\right)$. In this case, the argument is more complicated. The sum of heights of $C^\ulcorner$ and $C^\urcorner$ is at most $n-2$, otherwise we could find an acyclic matching of size $n-1$ adjacent to X. Thus, without loss of generality, we may assume that $C^{\urcorner}$ has height not exceeding $\lfloor (n-2)/2 \rfloor$. In particular, no vertex $(a,b) \in V_n$ with $\min\{a,b\} \leq (n+1)/2$ belongs to $C^{\urcorner}$.

If there is a path P from $v^{\ulcorner}$ to $v_{\lrcorner}$ that avoids $C^{\urcorner}$ and has at most two vertices from Y, we may apply Lemma 4 and we are done. Assume now, that there is no such path. In such case, we will prove that there is a large acyclic matching adjacent to Y. Let R be a path in $G[X \cup Z]$ that connects $v^{\ulcorner}$ with $v^{\urcorner}$, and let R' be a path of $G[X \cup Z]$ that connects $v_{\lrcorner}$ with $v^{\urcorner}$. These paths exist, because $G[X \cup Z]$ is connected. Let us choose the two paths in such a way that they have as few vertices outside of $C^{\urcorner}$ as possible. In particular, as soon as any of the two paths enters into $C^{\urcorner}$, it remains in $C^{\urcorner}$ until it reaches $v^{\urcorner}$.

Note that the two paths cannot intersect outside of $C^{\urcorner}$, otherwise we could find a path from $v^{\ulcorner}$ to $v_{\lrcorner}$ in $(X \cup Z) \setminus C^{\urcorner}$, which we assumed does not exist.

We define, for every $i = 1, \dots, \lfloor (n+1)/2 \rfloor$, an auxiliary path

$$Q_i = (i,n),(i,n-1),\dots,(i,i+1),(i,i),(i+1,i),(i+2,i),\dots,(n-1,i),(n,i).$$

Note that Q_i avoids $C^{\urcorner}$, and that both R and R' intersect Q_i. We define a pair of paths P_i and P'_i: the path P_i starts in $v^{\ulcorner}$ and follows R, until it first reaches Q_i. It then follows Q_i in the direction towards (n,i), until it reaches the first vertex of Y. Similarly, P'_i starts in $v_{\lrcorner}$, follows R', then follows Q_i towards (i,n) until it reaches a vertex of Y. Note that both these paths must eventually reach a vertex of Y, otherwise they would meet, forming path in $G[X \cup Z \setminus C^{\urcorner}]$ from $v^{\ulcorner}$ to $v_{\lrcorner}$. Let e_i be the last edge of P_i and e'_i the last edge of P'_i. These two edges are cut-edges adjacent to Y. Let $M = \{e_i \colon 1 \leq i \leq \lfloor (n+1)/2 \rfloor\}$ and $M' = \{e'_i \colon 1 \leq i \leq \lfloor (n+1)/2 \rfloor\}$.

We claim that $M \cup M'$ is an acyclic matching. To see this, note first that an edge $e_i \in M$ cannot intersect an edge $e'_j \in M'$, otherwise we could combine P_i and P'_j into a single path from $v^{\ulcorner}$ to $v_{\lrcorner}$ with at most one vertex from Y, and then apply Lemma 4. For the same reason, the edge-adjacency graph of the matching $M \cup M'$ has no arc between e_i and e'_j. It thus suffices to check that both M and M' are acyclic matchings, which follows from Lemma 5. Since $|M \cup M'| \geq 2\lfloor (n+1)/2 \rfloor \geq n$, this completes the case $T = \left(\begin{smallmatrix} X & X \\ Y & Z \end{smallmatrix}\right)$.

The case when all corners belong to $X \cup Y$. It remains to deal with the most complicated case, when all the four corners belong to at most two distinct parts X and Y. Let us say that two corner components C and C' are *linked* if $G[X \cup Y]$ has a path that connects C to C' and avoids the remaining two corner components. We say that C and C' are *almost linked* if G has a path between C and C' that avoids the remaining two corner components and has at most one vertex from Z.

If any two diagonally opposite corners are almost linked, then we are done by Lemma 4. Assume now that this is not the case. Since $G[X \cup Y]$ is connected, at least three pairs of corners must be linked. Assume without loss of generality that all the non-diagonal pairs except possibly the bottom pair $(C_{\llcorner}, C_{\lrcorner})$ are linked. We distinguish two subcases, depending on whether $(C_{\llcorner}, C_{\lrcorner})$ is almost linked or not.

First subcase: the pair $(C_{\llcorner}, C_{\lrcorner})$ *is not almost linked.* Let P be a path in $G[X \cup Y]$ from $v_{\llcorner}$ to $v^{\ulcorner}$ that avoids $C_{\lrcorner} \cup C^{\urcorner}$. Such a path exists, since $C_{\llcorner}$ and

$C^{\ulcorner}$ are linked. Choose P in such a way that it has as few vertices outside of $C_{\llcorner} \cup C^{\ulcorner}$ as possible. Let P' be a path that links $C_{\lrcorner}$ to $C^{\urcorner}$, chosen analogously to P. The two paths P and P' are disjoint, and no vertex of P may belong to the same component as a vertex of P', otherwise we could find a link between diagonally opposite corners.

For every row $i = 1, \ldots, n$ let v_i be the leftmost vertex of P that belongs to row i, and let v_i' be the rightmost vertex of P' in row i. Note that v_i is to the left of v_i'. Let $W_0 = \{v_1, \ldots, v_n\}$ and $W_0' = \{v_1', \ldots, v_n'\}$. Assume without loss of generality that at least n vertices in $W_0 \cup W_0'$ belong to X and write $W = W_0 \cap X$ and $W' = W_0' \cap X$. We now create an acyclic matching of size $|W \cup W'|$ adjacent to X. For each vertex $v_i \in W$ let e_i be the leftmost horizontal cut-edge in row i to the right of v_i. Note that such a cut-edge must exist, otherwise v_i and v_i' would belong to the same component.

Symmetrically, for $v_i' \in W'$, let e_j' be the rightmost horizontal cut-edge in row j to the left of v_j'. Let $M = \{e_i \colon v_i \in W\}$ and $M' = \{e_j' \colon v_j' \in W'\}$.

We claim that $M \cup M'$ is an acyclic matching. To see this, it suffices to prove that any two edges $e_i \in M$ and $e_j' \in M'$ are disjoint and non-adjacent in the edge-adjacency graph of $M \cup M'$. If $v_i \in C^{\ulcorner}$ and $v_j' \in C^{\urcorner}$ then the claim follows from the fact that $Y \cup Z$ is large, and hence there are at least two columns separating $C^{\ulcorner}$ from $C^{\urcorner}$. If, on the other hand, $v_i \notin C^{\ulcorner}$, then an adjacency or an intersection of e_i and e_j' implies that $C_{\llcorner}$ is almost linked to either $C_{\lrcorner}$ or $C^{\urcorner}$, which is impossible. If $v_i \in C^{\ulcorner}$ and $v_j' \notin C^{\urcorner}$ we get a contradiction in the same way. Since both M and M' are clearly acyclic matchings, $M \cup M'$ is an acyclic matching of size n adjacent to X. This completes the first subcase.

Second subcase: the pair $(C_{\llcorner}, C_{\lrcorner})$ is almost linked. For a corner component C, let $h(C)$ be the height of C. We claim that either $h(C_{\llcorner}) + h(C^{\urcorner}) < n - 1$ or $h(C^{\ulcorner}) + h(C_{\lrcorner}) < n - 1$. If there are two diagonally opposite corners that belong to the same part, then this claim follows directly from the remark below Lemma 6. If each two diagonally opposite corners belong to distinct parts, then there are two corners belonging to X and two corners belonging to Y. By Lemma 6 and the discussion below, we know that the four corner components have total height at most $2n - 4$. Thus, there must be a pair of diagonally opposite corner components whose combined height is at most $n - 2$.

Without loss of generality, assume that $h(C^{\ulcorner}) + h(C_{\lrcorner}) \leq n - 2$. Assume, furthermore, that every path from $v_{\llcorner}$ to $v^{\urcorner}$ that avoids $C^{\ulcorner} \cup C_{\lrcorner}$ has at least three vertices from Z, otherwise we may apply Lemma 4. Our aim now will be to find a large acyclic matching adjacent to Z. Similarly to the second case, we will do this by constructing a collection $P_1, \ldots, P_{\lfloor (n+1)/2 \rfloor}$ of paths that connect $v_{\llcorner}$ with a vertex of Z, and taking the last edge of each P_i as a matching edge e_i. Then, we construct a similar collection of matching edges e_i' from paths starting in $v^{\urcorner}$ and prove that the whole collection of edges forms an acyclic matching adjacent to Z.

Let us describe the construction in detail. Let R_1 be a path in $G[X \cup Y]$ that connects $v_{\llcorner}$ to $v^{\ulcorner}$ and avoids $C_{\lrcorner} \cup C^{\urcorner}$. Let R_2 be a path in G from $v_{\llcorner}$ to $v_{\lrcorner}$ that avoids $C^{\ulcorner} \cup C^{\urcorner}$ and has at most one vertex in Z (recall that $C_{\llcorner}$ and $C_{\lrcorner}$ are almost linked). Choose R_1 and R_2 to have as few vertices outside the corner components as possible. Let z be the vertex in $Z \cap R_2$; if $Z \cap R_2$ is empty, define z to be any vertex of R_2. Symmetrically, we define R'_1 and R'_2 to be the paths in $G[X \cup Y]$ connecting $v^{\urcorner}$ to $v^{\ulcorner}$ and $v_{\lrcorner}$ respectively, subject to the same minimality assumptions. No two of these four paths may intersect outside of the four corner components, since this would imply that a pair of diagonally opposite corner components is almost linked.

Let us say that an increasing diagonal is *free* if it does not intersect $C^{\ulcorner} \cup C_{\lrcorner}$. Recall that at least $n+1$ increasing diagonals are free. Note that any two adjacent free diagonals induce a path that avoids $C^{\ulcorner} \cup C_{\lrcorner}$, and that every free diagonal is intersected by R_1 or by R_2. Let us match the free diagonals into at least $\lfloor (n+1)/2 \rfloor$ disjoint pairs of adjacent diagonals. If the number of free diagonals is odd, one of them will be unmatched. Let $Q_1, \ldots, Q_{\lfloor (n+1)/2 \rfloor}$ be the paths induced by the matched pairs of diagonals.

For every $i \in \{1, \ldots, \lfloor (n+1)/2 \rfloor\}$, we define a path P_i in the following way: first, we follow either R_1 or R_2, whichever leads to Q_i, until we reach a vertex from Q_i. From this point on, we follow Q_i (in the increasing direction) until we reach the first vertex of Z. We then define the edge e_i to be the last edge of P_i. As a special exception, if the last vertex of P_i is the vertex z defined above, we leave e_i undefined. If e_i is defined, it is a subset of Q_i, hence all the edges we defined in this way are disjoint. Let M be the matching consisting of all the edges e_i defined this way. By Lemma 5, M is acyclic.

Symmetrically, we define the path P'_j, by following either R'_1 or R'_2 until we reach Q_j, and then following Q_j downwards, until we reach a vertex from Z. Let e'_j be the last edge of P'_j, and let M' be the acyclic matching of all the edges e'_j, for $j = 1, \ldots, \lfloor (n+1)/2 \rfloor$.

We will now show that no two edges $e_i \in M$ and $e'_j \in M'$ intersect or induce an arc in the edge-adjacency graph of $M \cup M'$. If this were the case we could combine P_i with P'_j into a path with at most two defects (one of them being the vertex z, the other resulting from the intersection or adjacency of the two edges) and apply Lemma 4.

We conclude that $M \cup M'$ is an acyclic matching adjacent to Z. Since $|M \cup M'| \geq 2\lfloor (n+1)/2 \rfloor - 1 \geq n - 1$, Proposition 1 is proved.

This completes the proof of Theorem 1.

Acknowledgements

I appreciate the valuable discussions with Jiří Fiala, Tomáš Gavenčiak, Eva Jelínková, Jan Kratochvíl, Bernard Lidický, Martin Pergel, Ondřej Suchý and Marek Tesař.

References

1. Hliněný, P., Oum, S., Seese, D., Gottlob, G.: Width parameters beyond tree-width and their applications. Computer Journal (advance access) (2007)
2. Oum, S.: Graphs of Bounded Rank-Width. PhD thesis, Princeton University (2005)

3. Oum, S.: Introduction to rank-width. In: Third workshop on graph classes, optimization, and width parameters, Eugene (2007), `http://math.kaist.ac.kr/~sangil/pdf/2007eugene.pdf`
4. Oum, S.: Recognizing rank-width. In: Oberwolfach Workshop "Graph Theory", `http://math.kaist.ac.kr/~sangil/pdf/oberwolfach2005.pdf`
5. Oum, S.: Rank-width is less than or equal to branch-width. Journal of Graph Theory 57(3), 239–244 (2007)
6. Oum, S., Seymour, P.: Approximating clique-width and branch-width. Journal of Combinatorial Theory, Series B 96(4), 514–528 (2006)

Improved Upper Bounds for Partial Vertex Cover[⋆]

Joachim Kneis, Alexander Langer, and Peter Rossmanith

Dept. of Computer Science, RWTH Aachen University, Germany

Abstract. The Partial Vertex Cover problem is to decide whether a graph contains at most k nodes covering at least t edges. We present deterministic and randomized algorithms with run times of $O^*(1.396^t)$ and $O^*(1.2993^t)$, respectively. For graphs of maximum degree three, we show how to solve this problem in $O^*(1.26^t)$ steps. Finally, we give an $O^*(3^t)$ algorithm for Exact Partial Vertex Cover, which asks for at most k nodes covering exactly t edges.

1 Introduction

The widely known problems Vertex Cover and Dominating Set are among the most important graph-theoretical optimization problems: Find a small set of nodes that cover all edges, or dominate the whole graph, respectively. These NP-complete problems are well studied with respect to approximability [13, 16], exact algorithms [8, 18], and parameterized complexity [6, 7]. Recently, *partial* variants of these and similar problems came into a broader research focus [2, 3, 4, 5, 9, 10, 11, 12, 14, 15]: Instead of covering all edges or dominating all nodes, it is sufficient to cover t edges or dominate t nodes, where t is an additional parameter. Being generalizations of Vertex Cover and Dominating Set, these problems are NP-complete.

In this paper, we study the complexity of Partial Vertex Cover defined as:

Input:	A graph $G = (V, E)$, positive integers k, t
Question:	Is there a $C \subseteq V$, $\lvert C \rvert \leq k$, such that C covers at least t edges?

The best known constant approximation factor with respect to k is 2 and there are several algorithms that achieve an approximation factor of $2 - o(1)$ [2, 4, 9, 11, 12]. Since this coincides with the best result for Vertex Cover (see, e.g., [16]), a significant improvement seems to be unlikely. With respect to t, it is easy to see that a simple greedy algorithm already has an approximation ratio of at least 2.

We can expect that Partial Vertex Cover is a harder problem than Vertex Cover: Many algorithms exploit the fact that if each edge $\{u, v\}$ must be covered, one of u or v must be part of the solution. This simple observation already gives us a 2-approximation and an $O^*(2^k)$ algorithm.[1] In the partial

⋆ Supported by the DFG under grant RO 927/7-1.

1 The O^* notation suppresses polynomial factors.

case, however, one does not know if an edge is being covered at all. Thus, the 2-approximation factor for Partial Vertex Cover is harder to prove, and we cannot expect an $O^*(f(k))$ algorithm for Partial Vertex Cover, as this implies FPT = W[1] [10].

The more interesting question is whether there is an fpt algorithm for the case that t rather than only k is small, i.e., an algorithm with run time bounded by $f(t)poly(n)$. Bläser answered this question positively even for Partial Set Cover [3], which is a generalization of Partial Vertex Cover. His randomized algorithm is based on color coding, achieving a run time of $O^*(5.437^t)$, and can be derandomized into a deterministic algorithm. The base in the exponential function is rather huge, though. A faster and much simpler randomized algorithm developed recently [14] achieves a run time of $O^*(2.09^t)$. While this is a significant improvement, derandomizing it would result in a time complexity that is not exponential in t.

In this paper, we present a deterministic algorithm with run time bounded by $O^*(1.396^t)$, which even beats the best known randomized methods. As the latter are based on the *many witnesses* paradigm, they cannot directly be efficiently derandomized. We overcome this obstacle by a new method that scans the possible witnesses in a special order. This way, either a good witness is found early on or the time spent on false witnesses is small. For graphs of maximum degree three, we devise a special algorithm with run time $O^*(1.26^t)$.

Moreover, we present a randomized algorithm for Partial Vertex Cover with a run time bounded by $O^*(1.2993^t)$ improving all previous results. While the algorithm is very simple — it basically selects either a node of maximum degree or two of its neighbors — the analysis is rather involved.

We also consider the variant of Partial Vertex Cover, where *exactly* t edges must be covered and introduce the new technique of *random orientations*. A randomized algorithm based on this technique solves this variant with a run time bounded by $O^*(3^t)$.

Due to space constraints, some of the proofs are omitted in the paper.

2 Preliminaries

Let $G = (V, E)$ be a graph and $U = \{v_1, \ldots, v_u\} \subseteq V$. For $v \in V$, the set of neighbors of v is denoted by $N(v)$, and $N[v] := N(v) \cup \{v\}$. By $\deg(U)$ we denote the degree sequence $(d_1, \ldots, d_u) = (\deg(v_{i_1}), \deg(v_{i_2}), \ldots, \deg(v_{i_u}))$, where $(i_1, \ldots, i_u)$ is a permutation of $(1, \ldots, u)$, such that $d_1 \geq d_2 \geq \cdots \geq d_u$. By $E(U)$ we denote the set of edges that are incident to some $v \in U$, and $||U|| := |E(U)|$. We call $C \subseteq V$ a (t, k)-vertex cover for G iff $|C| \leq k$ and $||C|| \geq t$. We define the relation $\succ$ on all instances of Partial Vertex Cover as $(G, k, t) \succ (G', k', t')$ iff $t > t'$ or $t = t' \wedge |G| > |G'|$.

A *branching vector* $(x_1, x_2, \cdots, x_l)$ is a short notation for a recursive function $T(n)$ of the form $T(n) = T(n - x_1) + T(n - x_2) + \cdots + T(n - x_l)$ for $n > 1$ and $T(n) = 1$ for $n \leq 1$. The corresponding *branching number* c can be used to bound $T(n)$ by $O(c^n)$ and can easily be computed using characteristic polynomials.

For more information about branching vectors and the corresponding branching numbers, see [17].

Let $s = (s_1, \dots, s_l)$ and $t = (t_1, \dots, t_l)$ be two branching vectors. We say s dominates t (denoted by $s \trianglerighteq t$ or $t \trianglelefteq s$), iff $s_i \geq t_i$ for $1 \leq i \leq l$. If $s \trianglerighteq t$, then the branching number for s is smaller than the branching number for t.

Let $u, v \in V$ be adjacent nodes of degree at least two. If $N[v] \subseteq N[u]$, we say u *dominates* v. We call G *reduced*, if there are no such nodes in G. For Partial Vertex Cover, we can assume G is reduced without loss of generality, otherwise the operation depicted in Figure 1 can be applied.

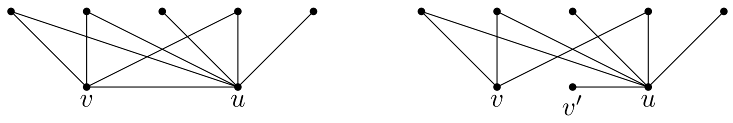


Fig. 1. Domination in graphs can be resolved by small modifications

The following lemmata can easily be deduced from a simple node exchange argument.

Lemma 1. *Let $G = (V, E)$ a graph and v a node of maximum degree d. If $v \notin C$ for* any *(t, k)-vertex cover $C \subseteq V$, then for each (t, k)-vertex cover C holds $i := |C \cap N(v)| > d - d_i + 1$, where* $\deg(C \cap N(v)) = (d_1, \dots, d_i)$.

Lemma 2. *Let G be a graph, v be a node of maximum degree d and $N(v) = \{v_1, \dots, v_d\}$, such that* $\deg(v_1) \geq \cdots \geq \deg(v_d)$. *If there is some i, such that for all (t, k)-vertex cover C we have $C \cap \{v, v_1, \dots, v_i\} = \emptyset$, but* $\deg(v_i) \leq i$, *then G does not contain any (t, k)-vertex cover at all.*

Lemma 3. *Let G be a graph, v be a node of maximum degree d and C a (t, k)-vertex cover for G. Let $N(v) = \{v_1, \dots, v_d\}$ with* $\deg(v_1) \geq \cdots \geq \deg(v_d)$. *If there is no (t, k)-vertex cover containing any node from $v, v_1, \dots, v_{d-2}$, then there is a (t, k)-vertex cover C' for G containing both v_{d-1} and v_d and we have* $\deg(v_{d-1}) + \deg(v_d) > d$.

3 A Fast Algorithm on Graphs of Maximum Degree Three

In this section, we present a new deterministic algorithm for Partial Vertex Cover on graphs of maximum degree three with a run time bounded by $O^*(1.26^t)$. The algorithm always branches on a node of degree three and some of its neighbors, thereby avoiding any node that has three neighbors of degree three if possible. Since we are only forced to select such a node in a three-regular graph, this can be avoided in any but the first step.

The branching itself depends on the degree of the neighbors and the edges between them, leading to a large case distinction. In order to increase the readability, we do not present the algorithm explicitly, but describe its behavior in the upcoming lemmata. Lemma 4 shows the possible branching operations if there is some triangle containing a node of degree three. Lemma 5 establishes the branching operations needed in triangle-free graphs of maximum degree three.

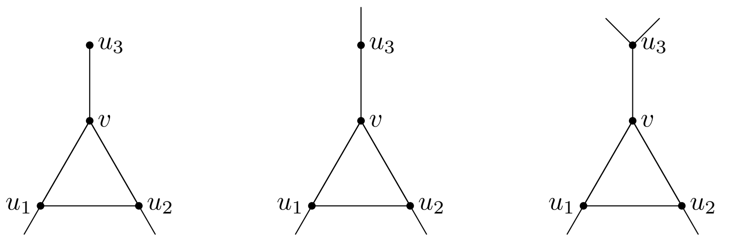


Fig. 2. Possible neighborhoods of a node v of maximum degree three being part of a triangle

Lemma 4. *Let G be a reduced graph of maximum degree three and v be a node of maximum degree that is part of a triangle (v, u_1, u_2). Then there is a branching with a branching vector of at least $(3, 3)$.*

Proof. Let u_3 be the remaining third neighbor of v. Since G is reduced, every node is part of at most one triangle, and each triangle does not contain any node of degree two. Therefore, the neighborhood of v is one of the three cases depicted in Figure 2.

Since v is of maximum degree, either v or some of its neighbors belong to some (t, k)-vertex cover, if such a cover exists. If v does not belong to any (t, k)-vertex cover, each cover C covers at least four edges with nodes from $N(v)$, because otherwise we could replace $C \cap N(v)$ with $\{v\}$. Thus, at least two nodes from $N(v)$ must be part of a (t, k)-vertex cover, if v is not.

- Let $\deg(u_3) = 1$ and C be a (t, k)-vertex cover. Without loss of generality, $u_3 \notin C$. If $\{u_1, u_2\} \subseteq C$ but $v \notin C$, $C \cup \{v\} \setminus \{u_2\}$ is a valid (t, k)-vertex cover. Thus, we can safely add v to C and no branching is necessary.
- Let $\deg(u_3) = 2$ and C be a (t, k)-vertex cover containing u_1 and u_3 but neither v nor u_2. Then $C \cup \{v\} \setminus \{u_3\}$ covers t edges with k nodes. If C contains only $\{u_1, u_2\}$, we can replace u_2 by v. Therefore, either v is part of some (t, k)-vertex cover for G, or all nodes in $\{u_1, u_2, u_3\}$ belong to such a cover, which results in a branching vector of $(3, 7)$.
- Let $\deg(u_3) = 3$. Similar to previous case, a cover containing only $\{u_1, u_2\}$ can be replaced by a cover containing $\{u_1, v\}$. Hence, either v or u_3 is part of an optimal solution, which yields the branching vector $(3, 3)$.

Lemma 5. *Let G be a graph of maximum degree three that is not three-regular and v a node of maximum degree that has a neighbor of degree two. Then there is a branching on nodes from $N[v]$ which yields a branching vector of at least $(3, 3)$.*

Proof. Recall that if no (t,k)-vertex cover contains v, at least two nodes from $N(v)$ are part of any (t,k)-vertex cover, since v is of maximum degree. Let furthermore $\Delta_i := |\{\, u \in N(v) \mid \deg(u) = i \,\}|$.

- If $\Delta_1 \geq 1$ and $\Delta_3 \leq 1$, v must be part of some (t,k)-vertex cover. Since $|N(v) \cap C| \geq 2$, at least one neighbor of degree at most two must be part of some (t,k)-vertex cover. Thus, Lemma 1 implies that there exists a (t,k)-vertex cover containing all neighbors of v, and since at least one u_i is of degree one, we can replace it with v.
- Let $\Delta_1 = 1$ and $\Delta_3 = 2$. Then both neighbors of degree three are contained in some (t,k)-vertex cover, because otherwise the node with degree one would be part of some (t,k)-vertex cover. But then, this node could be replaced by v. This implies a branching vector of $(3,6)$.
- In the following we can assume that no node in $N(v)$ is of degree one. Let $\Delta_2 = 3$. Since v is not part of any (t,k)-vertex cover, Lemma 1 implies that $N(v)$ belongs to some (t,k)-vertex cover. Again, we obtain the branching vector $(3,6)$.
- Now let $1 \leq \Delta_2 \leq 2$ and $\Delta_3 = 3 - \Delta_2$. Let $u_1 \in N(v)$ be a node of degree three. Either we have directly u_1 is part of some (t,k)-vertex cover or both other neighbors. But since one of these is of degree two, Lemma 1 implies that u_1 is part of some (t,k)-vertex cover too. Therefore, the corresponding branching vector is $(3,3)$.

Combining these lemmata, we obtain a run time bound of $O^*(1.26^t)$.

Theorem 1. Partial Vertex Cover *on graphs of maximum degree three can be solved in* $O^*(1.26^t)$.

Proof. Let us first consider the case that G is a connected graph of maximum degree three. If G is not three-regular, it is easy to see that applying the branching operations or the reduction rule does not lead to a connected three-regular graph. Even more, if the branching operation splits the graph into several components, each of these components is not three-regular as well. Hence, the three-regular case can only occur at the beginning. A simple branch for each $v \in V$, where membership in the (t,k)-vertex cover is tested, increases the run time by a factor n, but leaves us with a graph that is not three-regular.

The algorithm always chooses a node of degree three with at least one neighbor of degree two or less. Thus, by Lemma 4 and Lemma 5, its branching vector is at least $(3,3)$ in t, i.e., its running time is in $O^*(1.26^t)$.

If G is not connected, let $G_0, \ldots, G_s$ be its components. For each component G_j and each $k' \leq k$ we compute the maximum number $t_{j,k'}$ of edges that can be covered in G_j with k' nodes. Each component G_j is connected, hence t calls of the branching algorithm with parameter $0 \leq t' \leq t$ are sufficient per component. We can then use dynamic programming to compute the maximum number of edges $t_{1\cdots j,k'}$, $2 \leq j \leq n$ that can be covered with k' nodes, if only nodes from components $G_1, \ldots, G_j$ are allowed: For each $2 \leq j \leq s$ and each $0 \leq k' \leq k$, we have

$$t_{1\cdots j,k'} := \min\{\, t_{1\cdots(j-1),p} + t_{j,q} \mid p + q = k' \,\}.$$

The branching algorithm takes time $1.26^t \cdot poly(n)$, and is called $s \cdot t \cdot k$ times. Dynamic programming takes $O(s \cdot k^2)$ steps, hence we obtain an overall run time bound of $O^*(1.26^t)$ for arbitrary graphs of maximum degree three.

If we replace the actual branching with a randomized selection of the respective branching node(s), we obtain a simple randomized version of above algorithm.

Corollary 1. *There is a randomized algorithm* RPVC$_3$ *deciding* Partial Vertex Cover *on graphs of maximum degree three with success probability of at least* $(1/1.26)^t$.

4 A Deterministic Algorithm

In this section, we introduce a deterministic algorithm for arbitrary graphs. This algorithm, shown in Table 1, basically behaves as follows. A node of maximum degree is tested for membership in the (t, k)-vertex cover. If this test fails, one of its neighbors must be part of the solution, and the algorithm tests them in the decreasing order of their degrees.

Table 1. A deterministic algorithm for Partial Vertex Cover

```
Algorithm PVC($G, k, t$):
Input: Graph $G$, $k$, $t$
select a node $v$ of maximum degree $d$
let $N(v) = \{v_1, \dots, v_d\}$ and $\deg(v_1) \geq \dots \geq \deg(v_d)$
**if** $\deg(v) \leq 3$ **then** apply branching rules for graphs of maximum degree three
**else for** $i = 1, \dots, d-1$ **do**
    **if** $i \geq \deg(v_i)$ **then return** "no"
    **else if** $(i < d-1)$ **and** PVC($G - \{v\}, k-1, t-d$) **then return** "yes"
    **else return** PVC($G - \{v_{d-1}, v_d\}, k-2, t - \deg(v_{d-1}) - \deg(v_d)$)
```

Theorem 2. Partial Vertex Cover *can be solved in at most* $O^*(1.396^t)$ *steps by Algorithm* PVC.

Proof. Let G be a graph of maximum degree d. By Lemmata 2 and 3, Algorithm 1 solves Partial Vertex Cover. As for the run time, we first note that each recursive call only takes polynomial time. Now, we bound the number of recursive calls by a function of t. To do so, we measure how t decreases in each branch and evaluate the corresponding branching vectors. If $d \leq 3$, Partial Vertex Cover can be solved in $O^*(1.26^t)$ by Theorem 1. The corresponding branching vector is $(3, 3)$. If $d > 3$, either the ith recursive call in the loop returns "yes" and we obtain the branching vector

$$\big(d, \deg(v_1), \dots, \deg(v_{i-1})\big) \trianglerighteq \big(d, i, \dots, i\big) \trianglerighteq \big(i+1, i, \dots, i\big).$$

Otherwise, each of these calls returns "no", so $i = d$ and we obtain the branching vector

$$\big(d, \deg(v_1), \dots, \deg(v_{d-2}), \deg(v_{d-1}) + \deg(v_d)\big) \trianglerighteq \big(i, i, \dots, i, i+1\big).$$

For $i \geq 5$, we may estimate the branching with the simpler branching vector

$$(\underbrace{i, \dots, i}_{i \text{ times}}) \trianglelefteq (i+1, \underbrace{i, \dots, i}_{i-1 \text{ times}}).$$

The characteristic polynomial of this vector is $z^i - i$ with largest positive real root $i^{1/i} \leq 5^{1/5} \leq 1.38$. For $i < 5$, we obtain the branching numbers 1.325 for the vector $(5, 4, 4, 4)$ and 1.396 for the vector $(4, 3, 3)$ by a short computation. Thus, the number of recursive calls in Algorithm 1 is bounded by 1.396^t.

5 A Randomized Algorithm

In this section, we present a randomized algorithm for Partial Vertex Cover. Again, a node v of maximum degree is chosen deterministically, but either v or two of its neighbors are added to the (t, k)-vertex cover with certain probabilities. This technique leads to a polynomial-time algorithm with success probability of at least $1/1.2993^t$.

Fix $\alpha = (\frac{\sqrt{41}-1}{20})^{1/5} > 1/1.2993$ and let $p_d = \alpha^d$ for each $d \in \mathbf{N}$. The algorithm, see Table 2, is straight-forward and handles a only small number of border cases. We begin with an estimation that will be required later.

Lemma 6. *For $d = 4$ and $3 \leq i \leq d$, and for each $5 \leq d \in \mathbf{N}$ and each $i \in \mathbf{N}$, such that $2 \leq i \leq d$, we have*

$$(1 - \alpha^d)\binom{i}{2} \geq \alpha^{2d-2i+4}\binom{d}{2}.$$

Proof. It is sufficient to show that

$$1 \leq g(d, i) := (1 - \alpha^d)\alpha^{2i-2d-4}\frac{i(i-1)}{d(d-1)}$$

for all relevant d, i. For $d = 4$ and $i \in \{3, 4\}$ this is easily confirmed. Hence, fix $5 \leq d \in \mathbf{N}$. We first consider the cases $g(d, 2)$ and $g(d, d)$. The function

$$h_1(z) := \frac{1}{z(z-1)}\alpha^{-2z}$$

is strictly increasing on $[5, \infty)$, since

$$\frac{d}{dz}\ln\Bigl(\frac{1}{z(z-1)}\alpha^{-2z}\Bigr) = -\frac{1}{z} - \frac{1}{z-1} - 2\ln\alpha > 0$$

for $z \geq 5$. This implies

$$h_2(z) := g(z, 2) = (1 - \alpha^z)\frac{2}{z(z-1)}\alpha^{-2z} = 2(1 - \alpha^z)h_1(z)$$

Table 2. A randomized algorithm for (t, k)-vertex cover

```
Algorithm RPVC($G, k, t$):
Input: Graph $G$, $k$, $t$
if $k < 0$ then return "no"
if $t \leq 0$ then return "yes"
if $G$ is not connected then
    Compute optimal solutions for all $t' \leq t$ for every component of $G$.
    Combine the solutions using dynamic programming.
    Return whether there is a global solution for $G$.
if $G$ has maximum degree three then return RPVC$_3(G, k, t)$
if $G$ is four-regular then
    choose arbitrary $v \in V$
    if "yes" $\in \{ \text{RPVC}(G - u, k - 1, t - 4) \mid u \in N[v] \}$ then return "yes"
    else return "no".
else choose $v \in V$ of maximum degree $d$, so that $\deg(N(v)) \neq (4, 4, 4, 4)$
$X := \binom{N(v)}{2}$
if $\deg(N(v)) = (4, 4, 4, 3)$ then
    $X := \{ x \in X \mid \deg(x) = (4, 4) \}$
else if $\deg(N(v)) = (4, 4, 3, 3)$ then
    $X := \{ x \in X \mid \deg(x) \neq (3, 3) \}$
Uniformly choose $C \in X$.
Return $\begin{cases} \text{RPVC}(G - v, k - 1, t - d) & \text{with probability } p_d \\ \text{RPVC}(G - C, k - 2, t - ||C||) & \text{with probability } 1 - p_d \end{cases}$
```

is strictly increasing on $[5, \infty)$. Furthermore, $h_2(5) \geq 1$ — with equality for $d = 5$ — and therefore $g(d, 2) \geq 1$ for each $5 \leq d \in \mathbf{N}$. Similarly, let

$$h_3(z) := g(z, z) = (1 - \alpha^z)\alpha^{-4}.$$

Here, $\alpha^{-4} \approx 2.849$, and $\alpha^z \leq \frac{1}{2}$ for $z \geq 5$. Hence $h_3(z) = g(z, z) > 1$ for $z \geq 5$.

What remains to show is that $g(d, i) \geq 1$ for $i \in (2, d)$. We consider

$$f_d(z) := \ln g(d, z) = \ln\Big((1 - \alpha^d)\alpha^{2z-2d-4}\frac{z(z-1)}{d(d-1)}\Big).$$

Of course $g(d, i) \geq 1$ iff $f_d(i) \geq 0$. However, $f_d(z)$ is convex on $[2, d]$, because

$$f_d''(z) = -\frac{1}{z^2} - \frac{1}{(z-1)^2} < 0.$$

With $f_d(2) = \ln g(d, 2) \geq 0$ and $f_d(d) = \ln g(d, d) \geq 0$, we conclude $f_d(z) \geq 0$ on $[2, d]$ and hence $g(d, i) \geq 1$ for all $i \in \mathbf{N}$ with $2 \leq i \leq d$.

The correctness of RPVC and its success probability is due to the following lemma.

Lemma 7. *Let $G = (V, E)$ a graph.* RPVC(G, k, t) *runs in polynomial time. If there is no (t, k)-vertex cover for G, then* RPVC(G, k, t) *answers "no". Otherwise* RPVC(G, k, t) *answers "yes" with probability at least α^t.*

Proof. If there is no (t, k)-vertex cover for G, then RPVC(G, k, t) clearly answers "no". Otherwise, we use induction over the order $\succ$ on instances.

For $t = 0$, RPVC(G, k, t) answers "yes" with probability $1 = \alpha^0$. If $t > 0$ and G is not connected, let $G_0, \dots, G_s$ be its components, and let $(k_0, t_0), \dots, (k_s, t_s)$, such that $k_0 + \dots + k_s = k$, $t_0 + \dots + t_s = t$, and for each $0 \le i \le s$ there is a (t_i, k_i)-vertex cover for G_i. For all $0 \le i \le s$, we have $(G, k, t) \succ (G_i, k_i, t_i)$. Hence, by hypothesis RPVC(G_i, k_i, t_i) answers "yes" with probability at least α^{t_i}. Therefore, with probability at least

$$\alpha^{t_0}\alpha^{t_1}\cdots\alpha^{t_s} = \alpha^t$$

the dynamic programming approach is successful and RPVC(G, k, t) answers "yes". For correctness and run time of the dynamic programming approach, see the proof of Theorem 1.

From now, we assume G is connected. If G has maximum degree three, by Corollary 1 RPVC$_3(G, k, t)$ answers "yes" with probability at least $(1/1.26)^t > \alpha^t$.

If G is four-regular, let v be the node chosen by the algorithm. We know that there is a solution containing at least one $u \in N[v]$. Calling RPVC$(G - u, k - 1, t - 4)$ for each $u \in N[v]$ adds factor of five to the run time. Similar to the three-regular case, it is easy to see that the respective G will be four-regular at most once on any path in the recursive call tree.

If otherwise G is not four-regular, but contains a node of degree four or larger, let v be the node of maximum degree $d > 3$ that was chosen by the algorithm. Note that it is always possible to choose v with $\deg(N(v)) \neq (4, 4, 4, 4)$, since G is not four-regular. Let C be a (t, k)-vertex cover with $v \in C$, then there is a $(k - 1, t - d)$-vertex cover for $G - v$. With probability α^d the algorithm chooses v and calls RPVC$(G - v, k - 1, t - d)$. Hence, by induction hypothesis, the algorithm answers "yes" with probability at least $\alpha^d\alpha^{t-d} = \alpha^t$.

If otherwise there is no (t, k)-vertex cover containing v, we know that $|C \cap N(v)| \ge 2$ for each (t, k)-vertex cover C. Fix such a (t, k)-vertex cover C for G and let $D := C \cap N(v)$, $i := |D|$, and $\deg(D) = (d_1, \dots, d_i)$. By Lemma 1, we know $i \ge d - d_i + 2$, or $d_i \ge d - i + 2$. We begin with the two special cases that are distinguished by the algorithm:

1. If $d = 4$ and $\deg(N(v)) = (4, 4, 4, 3)$, then

$$\deg(D) \in \{\,(4, 4), (4, 4, 3), (4, 4, 4), (4, 4, 4, 3)\}.$$

With probability $(1 - \alpha^4)$ we do not choose v, and with probability at least $1/3$ we find the correct nodes $v_1, v_2 \in N(v)$. These nodes cover at least seven edges, and hence the probability to answer "yes" is, by induction, at least

$$(1 - \alpha^4)\frac{1}{3}\alpha^{t-7} = \frac{\alpha^{t-7} - \alpha^{t-4}}{3} > \alpha^t.$$

2. If $d = 4$ and $\deg(N(v)) = (4, 4, 3, 3)$, then

$$\deg(D) \in \{\,(4, 4), (4, 3, 3), (4, 4, 3, 3)\},$$

i.e., we know at least one node of degree four is in any (t,k)-vertex cover. If $i=2$, then $\deg(D)=(4,4)$, and we furthermore know that there is no edge between these neighbors (otherwise we can construct a (t,k)-vertex cover containing v, a contradiction). Hence, at least eight edges are being covered, and the probability to answer "yes" is, by induction, at least

$$(1-\alpha^4)\frac{1}{5}\alpha^{t-8}=\frac{\alpha^{t-8}-\alpha^{t-4}}{5}>\alpha^t.$$

If otherwise $3\le i\le 4$, we can only guarantee that six edges are being covered, but we gain an improved probability to pick two neighbors in D. We obtain a probability to answer "yes" of at least

$$(1-\alpha^4)\frac{1}{5}\binom{i}{2}\alpha^{t-6}=\frac{\alpha^{t-6}-\alpha^{t-2}}{5}\binom{i}{2}>\alpha^t.$$

The remaining cases are $d\ge 5$, or $d=4$ and

$$\deg(N(v))\notin\{\,(4,4,3,3),(4,4,4,3),(4,4,4,4)\,\}.$$

The latter enforces $i=|D|\ge 3$ due to the minimum degree d_i in D. With probability $(1-\alpha^d)$, the algorithm does not choose v. With probability $\binom{i}{2}/\binom{d}{2}$ the algorithm chooses two correct nodes $v_1,v_2\in D\subseteq N(v)$. Furthermore, v_1 and v_2 cover at least $2(d-i+2)$ edges: This is clear if v_1 and v_2 are not connected or neither v_1 nor v_2 are of degree $d-i+2$. However, if at least one node, say v_1, is of degree $d-i+2$, and v_1 and v_2 are connected, then a (t,k)-vertex cover containing v can be constructed from C by replacing v_1 with v—a contradiction.

By induction, the probability that RPVC$(G-\{v_1,v_2\},k-2,t-2(d-i+2))$ returns "yes" is at least $\alpha^{t-2(d-i+2)}$. Therefore, the success probability of RPVC(G,k,t) is at least

$$(1-\alpha^d)\frac{i(i-1)}{d(d-1)}\alpha^{t-2(d-i+2)}\ge\alpha^{2(d-i+2)}\alpha^{t-2(d-i+2)}=\alpha^t,$$

using the estimation from Lemma 6.

6 Exact Partial Vertex Cover

We define the parameterized problem EXACT PARTIAL VERTEX COVER as follows:

Input: A graph $G=(V,E)$, positive integers k, t
Question: Is there a $C\subseteq V$, $|C|\le k$, such that C covers *exactly* t edges?

The algorithms from the previous sections cannot be adapted to solve this problem, since there are solutions that do contain neither a node of maximum degree nor any of its neighbors.

While the first algorithm for Partial Vertex Cover by Bläser [3] might be modified to solve Exact Partial Vertex Cover as well, the random separation method by Cai, Chan, and Chan [5] solves this problem more efficiently in $O^*(2^{k+t}) = O^*(4^t)$ steps.

We use a similar and—to our best knowledge—new technique that we call *random orientation*: First randomly choose an orientation for each edge $\{v, u\}$: $v \to u$ (to u), $v \leftarrow u$ (to v), or $v - u$ (undirected). An *inner* node v is a node such that all edges incident to v are either undirected or point to v. An *inner* component U is a minimal, nonempty set U of inner nodes, such that $u \leftarrow v$ for each edge $\{u, v\}$ with $u \in U$ and $v \notin U$.

Theorem 3. *Let $G = (V, E)$ be a graph and C a solution of the* Exact Partial Vertex Cover *instance (G, k, t). Then C is a union of some inner components for a random orientation of E with probability at least 3^{-t}.*

After randomly choosing an orientation for a graph, compute all inner components. Note that no edge is incident to more than one inner component, thus we can use dynamic programming to test, whether some of these components contain together at most k nodes and cover exactly t edges. The overall run time is polynomial in t, k, and G. Obviously, the success probability is at least 3^{-t}. We easily obtain the following result:

Theorem 4. Exact Partial Vertex Cover *can be solved by a randomized algorithm in $O^*(3^t)$ with constant error probability.*

Note that this algorithm can be derandomized by using t-independent hash functions [1] yielding a run time of $O^*(c^t)$ for some constant c.

The method of random orientation can easily be used for other variants of Partial Vertex Cover, including several weighted problems. However, note that some variants are $W[1]$ hard, especially if we look for a (t, k)-vertex cover whose weight is exactly a given number (by a reduction from Subset Sum).

References

1. Alon, N., Yuster, R., Zwick, U.: Color-coding. J. ACM 42(4), 844–856 (1995)
2. Bar-Yehuda, R.: Using homogenous weights for approximating the partial cover problem. In: Proc. of 10th SODA, pp. 71–75 (1999)
3. Bläser, M.: Computing small partial coverings. Inf. Proc. Letters 85, 327–331 (2003)
4. Bshouty, N.H., Burroughs, L.: Massaging a linear programming solution to give a 2-approximation for a generalization of the vertex cover problem. In: Meinel, C., Morvan, M. (eds.) STACS 1998. LNCS, vol. 1373, pp. 298–308. Springer, Heidelberg (1998)
5. Cai, L., Chan, S.M., Chan, S.O.: Random separation: A new method for solving fixed-cardinality optimization problems. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 239–250. Springer, Heidelberg (2006)
6. Chen, J., Kanj, I.A., Xia, G.: Simplicity is beauty: Improved upper bounds for vertex cover. Technical Report TR05-008, School of CTI, DePaul University (2005)

7. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness. Congressus Numerantium 87, 873–921 (1992)
8. Fomin, F., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. Technical Report 359, Department of Informatics, University of Bergen (July 2007)
9. Gandhi, R., Khuller, S., Srinivasan, A.: Approximation algorithms for partial covering problems. Journal of Algorithms 53, 55–84 (2004)
10. Guo, J., Niedermeier, R., Wernicke, S.: Parameterized complexity of generalized vertex cover problems. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 36–48. Springer, Heidelberg (2005)
11. Halperin, E., Srinivasan, R.: Improved approximation algorithms for the partial vertex cover problem. In: Jansen, K., Leonardi, S., Vazirani, V.V. (eds.) APPROX 2002. LNCS, vol. 2462, pp. 185–199. Springer, Heidelberg (2002)
12. Hochbaum, D.S.: The t-vertex cover problem: Extending the half integrality framework with budget constraints. In: Jansen, K., Rolim, J.D.P. (eds.) APPROX 1998. LNCS, vol. 1444, pp. 111–122. Springer, Heidelberg (1998)
13. Johnson, D.S.: Approximation algorithms for combinatorial problems. J. Comput. Syst. Sci. 9, 256–278 (1974)
14. Kneis, J., Mölle, D., Richter, S., Rossmanith, P.: Intuitive algorithms and t-vertex cover. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 598–607. Springer, Heidelberg (2006)
15. Kneis, J., Mölle, D., Rossmanith, P.: Partial vs. complete domination: t-dominating set. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 367–376. Springer, Heidelberg (2007)
16. Monien, B., Speckenmeyer, E.: Ramsey numbers and an approximation algorithm for the vertex cover problem. Acta Informatica 22, 115–123 (1985)
17. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
18. Robson, J.M.: Finding a maximum independent set in time $O(2^{n/4})$. Technical Report 1251-01, Université Bordeaux I, LaBRI (2001)

On the Expressive Power of CNF Formulas of Bounded Tree- and Clique-Width

Pascal Koiran[1] and Klaus Meer[2,⋆]

[1] Laboratoire de l'Informatique du Parallélisme
ENS Lyon, France
koiran@ens-lyon.fr
[2] Lehrstuhl Theoretische Informatik
BTU Cottbus, Germany
meer@informatik.tu-cottbus.de

Abstract. Starting point of our work is a previous paper by Flarup, Koiran, and Lyaudet [5]. There the expressive power of certain families of polynomials is investigated. Among other things it is shown that polynomials arising as permanents of bounded tree-width matrices have the same expressiveness as polynomials given via arithmetic formulas. A natural question is how expressive such restricted permanent polynomials are with respect to other graph-theoretic concepts for representing polynomials over a field $\mathbb{K}$. One such is representing polynomials by formulas in conjunctive normal form. Here, a monomial occurs according to whether the exponent vector satisfies a given CNF formula or not. We can in a canonical way assign a graph to such a CNF formula and speak about the tree-width of the related CNF polynomial.

In this paper we show that the expressiveness of CNF polynomials of bounded tree-width again gives precisely arithmetic formulas. We then study how far the approach of evaluating subclasses of permanents efficiently using a reduction to CNF formulas of bounded tree-width leads. We show that there does not exist a family of CNF polynomials of bounded tree-width which can express general permanent polynomials. The statement is unconditional. An analoguous result for CNF polynomials of bounded clique-width is given, this time under the assumption that $\#P \not\subseteq FP/poly$.

The paper contributes to the comparison between classical Boolean complexity and algebraic approaches like Valiant's one.

1 Introduction

An active field of research in computational complexity is devoted to the design of efficient algorithms for subclasses of problems which in full generality likely are hard to solve. It is common in this area to define such subclasses via bounding

⋆ Support of the following institutions is gratefully acknowledged: ENS Lyon; the French embassy in Denmark, Service de Coopération et d'Action Culturelle, Ref.:39/2007-CSU 8.2.1; the Danish Agency for Science, Technology and Innovation FNU.

some significant problem parameters. Typical such parameters are the tree- and clique-width if a graph structure is involved in the problem's description.

In the present paper we consider a graph-theoretic approach in order to deal with problems that are related to families of polynomials. These families are given in a particular manner through certain Boolean formulas in conjunctive normal form, shortly CNF formulas. More precisely, we consider functions of the form

$$f(x) = \sum_{e \in \{0,1\}^n} \varphi(e) x^e, x \in \{0,1\}^n \text{ , for some } n \in \mathbb{N}, \qquad (*)$$

where φ is a CNF formula in n Boolean variables. We are interested in the question how expressive such a representation of polynomials is and under which additional conditions $f(x)$ can be evaluated efficiently. Fischer, Makowsky, and Ravve [4], extending earlier results from [2], have shown that the counting SAT problem, i.e. computing $\sum\limits_{e \in \{0,1\}^n} \varphi(e)$ for a CNF formula φ can be solved in time $O(n \cdot 4^k)$ if a certain bipartite graph G_φ canonically attached to φ is of bounded tree-width k.

Our first main result precisely characterizes the expressive power of functions of form $(*)$ when G_φ is of bounded tree-width. It is shown that the class of these polynomials equals both the class of polynomials representable by arithmetic formulas of polynomial size and the class of functions obtained as permanents of matrices of bounded tree-width and polynomially bounded dimension. Here, equality of the latter two concepts was known before due to a result of Flarup, Koiran, and Lyaudet [5].

Recall that in Valiant's algebraic model of computation for families of polynomials the permanent is VNP complete and thus likely not efficiently computable. Though an unconditional proof of this conjecture seems extremely difficult we next show that at least trying to obtain an efficient algorithm for computing permanents through formulas of type $(*)$ with G_φ of bounded tree-width must fail. This result is unconditioned in that it does not rely on any open conjecture in complexity theory.

Next, we pose the corresponding question for CNF formulas of bounded clique-width. Using another result from [4] we show that expressing the permanent of an arbitrary matrix by formulas of type $(*)$, this time with G_φ of bounded clique-width would imply $\#P \subseteq FP/poly$ and thus is unlikely.

The paper is organized as follows. In Section 2 we recall basic definitions as well as the needed results from [4] and [5]. Section 3 first shows how permanents of matrices of bounded tree-width can be expressed via polynomials of form $\sum\limits_{e \in \{0,1\}^n} \varphi(e) x^e$ with G_φ of bounded tree-width. Then, we extend a result from [4] to link such polynomials to arithmetic formulas. The results in [5] now imply equivalence of all three notions. In Section 4 the above mentioned negative results concerning expressiveness of (general) permanents by CNF formulas of bounded tree- or clique-width are proven.

Our results contribute to the comparison of Boolean and algebraic complexity. In particular, we consider it to be interesting to find more results like Theorem 8 below which states that certain properties **cannot** be expressed via (certain) graphs of bounded tree-width.

2 Basic Definitions

We start briefly collecting basic definitions and results that are needed below.

2.1 Arithmetic Circuits

Definition 1. *a) An arithmetic circuit is a finite, acyclic, directed graph. Vertices have indegree 0 or 2, where those with indegree 0 are referred to as* inputs*. A single vertex must have outdegree 0, and is referred to as* output*. Each vertex of indegree 2 must be labeled by either + or ×, thus representing computation. Vertices are commonly referred to as* gates*. By choosing as input nodes either some variables x or constants from a field $\mathbb{K}$ a circuit in a natural way represents a multivariate polynomial over $\mathbb{K}$.*

b) An arithmetic formula *is a circuit for which all gates except the output have outdegree 1 (therefore, reuse of partial results is not allowed in arithmetic formulas).*

c) The size *of a circuit is the total number of* gates *in the circuit.*

2.2 Tree- and Clique-Width

Treewidth for undirected graphs is defined as follows:

Definition 2. *Let $G = \langle V, E\rangle$ be a graph. A k-tree-decomposition of G is a tree $T = \langle V_T, E_T\rangle$ such that:*

(i) Each $t \in V_T$ is labelled by a subset $X_t \subseteq V$ of size at most $k + 1$.
(ii) For each edge $(u, v) \in E$ there is a $t \in V_T$ such that $\{u, v\} \subseteq X_t$.
(iii) For each vertex $v \in V$ the set $\{t \in V_T | v \in X_T\}$ forms a (connected) subtree of T.

The tree-width of G is then the smallest k such that there exists a k-tree-decomposition for G.

Next we recall the clique-width notion.

Definition 3. *A graph G has* clique-width *at most k iff there exists a set of k labels $\mathcal{S}$ such that G can be constructed using a finite number of the following operations:*

i) $vert_a, a \in \mathcal{S}$ (create a single vertex with label a);
ii) $\phi_{a\to b}(H), a, b \in \mathcal{S}$ (rename all vertices having label a to have label b);

iii) $\eta_{a,b}(H), a, b \in \mathcal{S}, a \neq b$ *(add edges between all vertices having label a and all vertices having label b);*

iv) $H_1 \oplus H_2$ *(disjoint union of graphs).*

To each graph of clique-width k we can attach a (rooted) parse-tree whose leaves correspond to singleton graphs and whose vertices represent one of the operations above. The graph G then is represented at the root.

2.3 Permanent Polynomials

Definition 4. *The permanent of an* (n, n)*-matrix* $M = (m_{i,j})$ *is defined as* $perm(M) := \sum_{\sigma \in S_n} \prod_{i=1}^{n} m_{i,\sigma(i)}$ *, where* S_n *is the symmetric group.*

We are interested in representing polynomials via permanents. If M above has as entries either variables or constants from some field $\mathbb{K}$, then $f = perm(M)$ is a polynomial with coefficients in $\mathbb{K}$ (in Valiant's terms f is a projection of the permanent polynomial). One main result in [5] characterizes arithmetic formulas of polynomial size by certain such polynomials. The tree-width of a matrix $M = [m_{ij}]$ is defined to be the tree-width of the graph we get by including an edge (i, j) iff $m_{ij} \neq 0$.

Theorem 1. *([5]) Let* $(f_n)_{n \in \mathbb{N}}$ *be a family of polynomials with coefficients in a field* $\mathbb{K}$*. The following properties are equivalent:*

(i) $(f_n)_{n \in \mathbb{N}}$ *can be represented by a family of polynomial size arithmetic formulas.*

(ii) There exists a family $(M_n)_{n \in \mathbb{N}}$ *of polynomial size, bounded tree-width matrices such that the entries of* M_n *are constants from* $\mathbb{K}$ *or variables of* f_n*, and* $f_n = perm(M_n)$*.*

2.4 Clause Graphs

One of our goals is to relate Theorem 1 to yet another concept, namely CNF formulas of bounded tree-width. The latter will be defined in this subsection. Our presentation follows closely [4].

Definition 5. *Let* φ *be a Boolean formula in conjunctive normal form with clauses* $C_1, \ldots, C_m$ *and Boolean variables* $x_1, \ldots, x_n$*.*

a) The signed clause graph $SI(\varphi)$ *is a bipartite graph with the* x_i *and the* C_j *as nodes. Edges connect a variable* x_i *and a clause* C_j *iff* x_i *occurs in* C_j*. An edge is signed* $+$ *or* $-$ *if* x_i *occurs positively or negated in* C_j*.*

b) The incidence graph $I(\varphi)$ *of* φ *arises from* $SI(\varphi)$ *by omiting the signs* $+, -$*.*

c) The primal graph $P(\varphi)$ *of* φ *has only the* x_i*'s as its nodes. An edge connects* x_i *and* x_j *iff both occur commonly in one of the clauses.*

d) The tree- or clique-width of a CNF formula φ *is defined to be the tree- or clique-width of* $I(\varphi)$*, respectively.*
If below we want to speak about the tree-width of $P(\varphi)$ *we mention this explicitly.*

Theorem 2. *([4]) a) Given φ and a tree-decomposition of $I(\varphi)$ of width k one can compute the number of satisfying assignments $\sum_{x\in\{0,1\}^n} \varphi(x)$ of φ in $4^k n$ arithmetic operations.*

b) Given a CNF formula φ and a parse-tree for the signed clause graph $SI(\varphi)$ of clique-width $\leq k$ the number $\sum_x \varphi(x)$ of satisfying assignments of φ can be computed in $O(n2^{ck})$ many arithmetic operations.

Below, we extend the algorithm proving Theorem 2 a) in order to relate CNF formulas to arithmetic formulas and Theorem 1. Note that similar results to those of part a) of Theorem 2 have independently been obtained in [9].

2.5 Non-deterministic OBDDs

The final notion we need to introduce is that of deterministic and non-deterministic Ordered Binary Decision Diagrams OBDDs. For a more extensive presentation of OBDDs see [10].

Definition 6. *a) A* binary decision diagram *or BDD is a rooted directed acyclic graph having two kinds of nodes. Output nodes are nodes with no outgoing edge and are labeled with a Boolean constant from* $\{0, 1\}$. *Inner nodes are labeled with an element from some variable set* $\{x_1, \ldots, x_n\}$. *They have two outgoing edges one of which is labeled by* 0 *and the other by* 1. *The* size *of a BDD is the number of nodes of the underlying graph.*

b) A BDD is ordered (denoted OBDD) if there is an ordering of the variables such that they occur along each path from the root to an output node according to the ordering.

c) Each OBDD computes a Boolean function $f : \{0, 1\}^n \to \{0, 1\}$ in the following way. Given an assignment for the Boolean variables $x_1, \ldots, x_n$ one follows starting from the root at each node labeled x_i that edge which is labeled with the value of x_i. The result obtained is the label of the output node reached.

d) An OBDD is non-deterministic *if it contains an additional type of nodes labeled as guess nodes. Such a node has two outgoing edges that are unlabeled. Each such edge can be followed by an input. The non-deterministic OBDD computes the result 1 for input x iff there is at least one path leading to a leaf labeled by* 1 *that can be followed for input x.*

A non-deterministic OBDD $\mathcal{O}$ with n variables can be seen as a deterministic OBDD $\tilde{\mathcal{O}}$ working on m additional inputs Θ. Then $\mathcal{O}(x) = 1 \quad \Leftrightarrow \quad \exists\ \Theta \in \{0, 1\}^m$ s.t. $\tilde{\mathcal{O}}(x, \Theta) = 1$.

3 Expressiveness of CNF Polynomials of Bounded Tree-Width

In this section we prove our first main result. We study how expressive polynomials p_n are which are given via CNF formulas φ_n of bounded tree-width.

It turns out that permanents of bounded tree-width matrices are captured by such CNF polynomials, whereas the latter in turn are captured by arithmetic formulas. Given the equivalence stated in Theorem 1 all three concepts have the same expressive power.

3.1 From Permanents to Clause Graphs

Theorem 3. *Let $M = [m_{ij}]$ be an $n \times n$ matrix such that the corresponding directed weighted graph $G_M = (V_M, E_M)$ is of tree-width k. Then there is a CNF formula φ of tree-width $O(k^2)$ and of size polynomially bounded in n such that*

$$perm(M) \;=\; \sum_{e,\theta} \varphi(e,\theta) \cdot m^e.$$

Here, $e = \{e_{i,j}\}$ denotes variables representing the edges of $G_M, m = \{m_{i,j}\}$ denotes the entries of M and $m^e := \prod_{i,j} m_{i,j}^{e_{i,j}}$, where $m_{i,j}^{e_{i,j}} = \begin{cases} m_{i,j} & \text{if } e_{i,j} = 1 \\ 1 & \text{if } e_{i,j} = 0 \end{cases}$.

For every e there exists θ such that $\varphi(e,\theta) = 1$ if and only if e is a cycle cover of G_M; in this case, the corresponding θ is unique.

Moreover, the number of additional variables θ is of order $O(n)$. Finally, a tree decomposition of $I(\varphi)$ of width $O(k^2)$ can be obtained from a decomposition of G_M in time $O(n)$.

Remark 1. In the above CNF polynomial $\sum_{e,\theta} \varphi(e,\theta) \cdot m^e$ there are no monomials corresponding to θ. Formally one could introduce another block y of variables and add to each monomial m^e another factor y^θ. Then $perm(M)$ is obtained as a projection (in Valiant's sense) of a CNF-polynomial $\sum_{e,\theta} \varphi(e,\theta) \cdot m^e \cdot y^\theta$ by plugging in for each y-variable the value 1.

Proof. Let $(T, \{X_t\}_t)$ be a tree decomposition of width k for G_M. Without loss of generality T is a binary tree. The CNF formula φ to be constructed contains two blocks of variable vertices, one being the edge-variables $e_{i,j}$ of G_M and another block θ of auxiliary variables to be explained below. The tree decomposition $(T, \{X'_t\}_t)$ that we shall construct for φ uses the same underlying tree T as the tree decomposition of G_M, but the boxes X'_t will be different from the boxes X_t in the initial decomposition.

A straightforward set of clauses to describe cycle covers in G_M is the following collection:

(i) for each vertex $i \in V_M$ clauses Out_i and In_i containing as its literals all outgoing edges from and all incoming edges into i, respectively;
(ii) for each $i \in V_M$ and each pair of outgoing edges $e_{i,j}, e_{i,l}$ a clause $\neg e_{i,j} \vee \neg e_{i,l}$; similarly for incoming edges to i.

A tree decomposition of the resulting formula then is obtained from T by taking the same tree and joining in a box X'_t for every $i \in X_t$ all vertices resulting from (i) and (ii). However, due to the conditions under (ii) this might not result in a decomposition of bounded width.

To resolve this problem for each box $t \in T$ and each $i \in V_M$ we add additional variables $check^t_{i+}, check^t_{i-}$. Fix t and the subtree T_t of T that has t as its root. For any assignment of the $e_{i,j}$ indicating which edges in G_M have been chosen for a potential cycle cover a condition $check^t_{i+} = 1$ indicates that an edge starting in i has already been chosen with respect to those vertices of G_M occuring in the subtree T_t.

Further clauses are introduced to guarantee that each i finally is covered exactly once for a satisfying assignment of $\varphi(e, \theta)$, where θ is the collection of all check variables. More precisely, we proceed bottom up. Let t be a leaf of T. For every $i \in X_t$ in addition to the variable vertices $check^t_{i+}, check^t_{i-}$ introduce clause variables representing the following clauses:

(1) $\bigvee\limits_{j \in X_t} e_{i,j} \vee \neg check^t_{i+}$;
Interpretation: if none of the $e_{i,j}$'s where chosen yet, then $check^t_{i+} = 0$.
(2) $\neg e_{i,j} \vee \neg e_{i,l}$ for all $j, l \in X_t$;
Interpretation: at most one outgoing edge covers i.
(3) $\neg e_{i,j} \vee \neg check^t_{i+}$ for all $j \in X_t$;
Interpretation: if an $e_{i,j}$ was chosen (i.e. $e_{i,j} = 1$), then $check^t_{i+} = 1$.

Analogue clause variables are added for $check^t_{i-}$.

For the box X'_t in the decomposition of $I(\varphi)$ that corresponds to box X_t of T all variable vertices $e_{i,j}, check^t_{i+}, check^t_{i-}, i, j, \in X_t$ as well as the clause variables resulting from (1)-(3) above are included. These are $O(k^2)$ many elements in X'_t. Now T' is constructed bottom up. The check variables propagate bottom up the information whether a partial assignment for those $e_{i,j}$ that already occured in a subtree can still be extended to a cycle cover of G_M. At the same time, the width of the new boxes of T' constructed will not increase too much. Suppose in T there are boxes t, t_1, t_2 such that t_1 is the left and t_2 the right child of t. Let $i \in X_t \cap X_{t_1} \cap X_{t_2}$. The case where i only occurs in two or one of the boxes is treated similarly. Assuming X'_{t_1}, X'_{t_2} already been constructed the following clauses are included in X_t :

(1′) $\bigvee\limits_{j \in X_t \setminus \{X_{t_1} \cup X_{t_2}\}} e_{i,j} \vee check^{t_1}_{i+} \vee check^{t_2}_{i+} \vee \neg check^t_{i+}$;
Interpretation: if all new $e_{i,j}$'s and the previous check variables are 0, then the new check variable $check^t_{i+}$ is 0 as well;
(2′) $\neg x \vee \neg y$ for all $x, y \in \{e_{i,j} : j \in X_t \setminus \{X_{t_1} \cup X_{t_2}\}\} \cup \{check^{t_1}_{i+}, check^{t_2}_{i+}\}; x \neq y$
Interpretation: at most one among the old check variables and the new edge variables gets the value 1;
(3′) $\neg x \vee \neg check^t_{i+}$ for all $x \in X_t \setminus \{X_{t_1} \cup X_{t_2}\} \cup \{check^{t_1}_{i+}, check^{t_2}_{i+}\}$;
Interpretation: if one among the values $e_{i,j}$ or $check^{t_1}_{i+}, check^{t_2}_{i+}$ is 1, then $check^t_{i+} = 1$.

Again, analogue clauses are added for the ingoing edges to i. Box X'_t contains all related edge vertices $e_{i,j}$ for the new $j \in X_t \setminus \{X_{t_1} \cup X_{t_2}\}$, the six check vertices and the $O(k^2)$ many clause vertices resulting from (1')-(3').

This way $(T, \{X'_t\}_t)$ is obtained. Finally, for each $i \in T$ two new clauses containing the single literals $check^r_{i+}$ and $check^r_{i-}$, respectively, are included in that box X_r which represents the root r of the subtree of T generated by all boxes that contain i. This is to guarantee that i is covered in both directions.

Clearly, $(T, \{X'_t\}_t)$ is a binary tree with each X'_t containing at most $O(k^2)$ many vertices. Let θ denote the vector of all check variables. It is obvious from the construction that

$$\exists\theta \ \varphi(e,\theta) \Leftrightarrow e \text{ represents a cycle cover}$$

(via those $e_{i,j}$ that have value 1). Moreover, for each assignment of e^* giving a cycle cover there is precisely one assignment θ^* such that $\varphi(e^*,\theta^*)$ because e^* uniquely determines which check variables have to be assigned the value 1. Therefore

$$perm(M) \ = \ \sum_{e,\theta} \varphi(e,\theta) \cdot m^e.$$

Finally, it remains to show that $(T', \{X'_t\}_t)$ actually is a tree decomposition of the graph $I(\varphi)$. Vertices resulting from check variables at most occur in two consecutive boxes of T' and thus trivially satisfy the connectivity condition. Clause vertices related to one of the construction rules (1), (3), $(1') - (3')$ for a fixed $t \in T$ only occur in the single box X'_t. Finally, an edge variable $e_{i,j}$ occurs in a box X'_t iff both i and j occur in X_t. Thus, the fact that $(T, \{X_t\}_t)$ is a tree decomposition implies that the connectivity condition also holds for these vertices and $(T', \{X'_t\}_t)$. □

3.2 From Clause Graphs to Arithmetic Formulas

In the next step we link CNF polynomials to arithmetic formulas. More precisely, the next theorem shows the latter concept to be strong enough to capture the former.

Theorem 4. *Let $\mathbb{K}$ be a field. Let $\{\varphi_n\}_n$ be a family of CNF formulas of bounded tree-width k and with n variables, $SI(\varphi_n)$ the related signed clause graphs and $(T_n, \{X_t\}_t)$ a tree decomposition of $I(\varphi_n)$. Then there is a family $\{f_n\}_n$ of polynomials with coefficients in $\mathbb{K}$ such that $\{f_n\}_n$ can be represented by a family of polynomial size arithmetic formulas and for all $x \in \mathbb{K}^n$,*

$$f_n(x) = \sum_{z \in \{0,1\}^n} \varphi_n(z) \cdot x^z$$

The proof is based on an extension of results in [4] and shall be given in the full paper. Theorems 1, 3 and 4 imply.

Theorem 5. *Let $(f_n)_{n\in\mathbb{N}}$ be a family of polynomials with coefficients in a field $\mathbb{K}$. The following properties are equivalent:*

(i) *$(f_n)_{n\in\mathbb{N}}$ can be represented by a family of polynomial size arithmetic formulas.*
(ii) *There exists a family $(M_n)_{n\in\mathbb{N}}$ of polynomial size, bounded tree-width matrices such that the entries of M_n are constants from $\mathbb{K}$ or variables of f_n, and $f_n = perm(M_n)$.*
(iii) *There exists a family $(\varphi_n)_{n\in\mathbb{N}}$ of CNF formulas having polynomial size in n and of bounded tree-width such that $f_n(x)$ can be expressed as the projection: $f_n(x) = \sum\limits_{\tilde{e}} \varphi_n(\tilde{e}) \cdot z^{\tilde{e}}$. Here, projection means that the z_i's can be taken either as constants from $\mathbb{K}$ or as variables among the x_j's.* □

4 Lower Bounds

Given Theorem 3 together with the efficient algorithm resulting from Theorem 4 the following question arises: How far does the approach of reducing permanent computations to computations of the form $\sum\limits_{e,\theta} \varphi(e,\theta) \cdot m^e$ lead when φ comes from a clause graph of bounded tree-width?

More precisely, we ask whether there exist polynomial size CNF formulas $\varphi_n(e,\theta)$ of bounded tree-width such that $\varphi_n(e,\theta) = 1$ iff $e \in \{0,1\}^{n\times n}$ is a permutation matrix and for each permutation matrix e there is exactly one θ such that $\varphi(e,\theta) = 1$.

In this section we prove that such formulas do not exist. A negative answer could be expected since Theorem 4 would otherwise imply that the permanent can be represented by polynomial size arithmetic formulas. The point is that Theorem 8 below is unconditional (and does not even need the uniqueness assumption on θ). A second (conditional) result shows that when replacing tree- by clique-width a formula with the above properties does not exist unless $\#P \subseteq FP/poly$.

The unconditional statement concerning tree-width relies on two results by Ferrara et al. [3] on the one side and by Krause et al. [8] on the other. The former relates clause graphs to OBDDs (ordered binary decision diagrams), whereas the latter gives lower bounds on the size of non-deterministic OBDDs deciding the property of being a permutation matrix.

Let us first recall these results.

Theorem 6. *([3]) Let φ be a CNF formula with n variables and $P(\varphi)$ the corresponding primal graph of φ. If $P(\varphi)$ has tree-width k, then there is an OBDD representation of φ which has size polynomial in n and exponential in k.*

Due to the additional block θ of variables introduced in the transformation constructed in the proof of Theorem 3, an application of the above theorem leads to non-deterministic OBDDs.

Theorem 7. *([8,6]) For $n \in \mathbb{N}$ define $PERM_n : \{0,1\}^{n^2} \to \{0,1\}$ to be the characteristic function for permutation matrices, i.e. for $e \in \{0,1\}^{n\times n}$ we have*

$PERM_n(e) = 1$ iff e is a permutation matrix. Then $2^{\Omega(n)}$ is a lower size bound for any non-deterministic OBDD computing $PERM_n$.

The technical result below allows to apply Theorem 6 the way we need it. Note that the theorem assumes the primal graph $P(\varphi)$ to be of bounded tree-width, whereas we want to deal with the incidence graph $I(\varphi)$.

Proposition 1. *Let $\varphi = C_1 \wedge \ldots \wedge C_m$ be a CNF formula with n variables $x_1, \ldots, x_n$ such that its incidence clause graph $I(\varphi)$ has tree-width k. Then there is a CNF formula $\tilde{\varphi}(x, y)$ such that the following conditions are satisfied:*

– *each clause of $\tilde{\varphi}$ has at most $O(k)$ many literals;*
– *the primal graph $P(\tilde{\varphi})$ has tree-width $O(k)$. A tree-decomposition can be constructed in linear time from one of $I(\varphi)$;*
– *the number of variables and clauses in $\tilde{\varphi}$ is of order $O(n \cdot m)$;*
– *for all $x^* \in \{0,1\}^n$ $\varphi(x^*)$ holds true iff there exists y^* such that $\tilde{\varphi}(x^*, y^*)$. Moreover, such a y^* is unique.*

Proof. Let $(T, \{X_t\}_t)$ be a (binary) tree-decomposition of $I(\varphi)$. The construction below combines the use of check variables in the proof of Theorem 3 with the usual way of reducing a general CNF formula instance to one with bounded number of literals in each clause. Let C be a clause of φ and T_C the subtree of T induced by C. We replace C bottom up in T_C by introducing $O(n)$ many new variables and clauses. More precisely, start with a leaf box X_t of T_C. Suppose it contains k variables that occur in literals of C, without loss of generality say $x_1 \vee \ldots \vee x_k$. The case where additional clause variables occur in X_t is treated similarly. Introduce a new variable y_t together with $O(k)$ many clauses expressing the equivalence $y_t \Leftrightarrow x_1 \vee \ldots \vee x_k$. Each of the new clauses has at most $k+1$ many literals. Next, consider an inner node t of T_C having two childs t_1, t_2. Suppose $x'_1, \ldots, x'_k$ to be those variables in X_t that occur as literals in C, again without loss of generality in the form $x'_1 \vee \ldots \vee x'_k$. If y_{t_1}, y_{t_2} denote the new variables related to C that have been introduced for X_{t_1}, X_{t_2}, for X_t define a new variable y_t together with clauses expressing $y_t \Leftrightarrow y_{t_1} \vee y_{t_2} \vee x'_1 \vee \ldots \vee x'_k$. Again, there are at most $O(k)$ new clauses containing $O(k)$ literals each. Finally, if t is the root of T_C we define y_t as before and add a clause saying $y_t = 1$.

Do the same for all clauses of φ. This results in a CNF formula $\tilde{\varphi}$ which depends on $O(m \cdot n)$ additional variables y and contains $O(m \cdot n \cdot k)$ many clauses. The construction guarantees that $\varphi(x)$ iff there exists a y such that $\tilde{\varphi}(x, y)$ and in that case y is unique.

A tree-decomposition of the primal graph $P(\tilde{\varphi})$ is obtained as follows. For each occurence of a clause C in X_t of T replace the clause vertex by the newly introduced y variables related to the clause and the box X_t. In addition, for boxes X_t, X_{t_1}, X_{t_2} such that t_1, t_2 are sons of t include the variables y_{t_1}, y_{t_2} also in the upper box X_t. The x_i variables that previously occured are maintained. Since for a single box X_t at most three y_j are included for each clause, and since there are at most $k+1$ clause vertices in an original box, the tree-width of $P(\tilde{\varphi})$ is $\leq 4(k+1)$. The decomposition satisfies the requirements of a tree-decomposition since we did not change occurences of the x_i's and the only y_t-variables that occur in several boxes occur in two consecutive ones. $\square$

As consequence we get

Theorem 8. *There does not exist a family $\{\varphi_n\}_n$ of CNF formulas $\varphi_n(e,\theta)$ such that $I(\varphi_n)$ is of bounded tree-width, the size of φ_n is polynomially bounded in n, and $\exists\theta\ \varphi_n(e,\theta)$ iff $e \in \{0,1\}^{n\times n}$ is a permutation matrix.*

Proof. Suppose to the contrary that such a family exists. Then by Proposition 1 we can assume without loss of generality that the primal graphs $P(\varphi_n)$ are of bounded tree-width as well. Moreover, the number of new variables and clauses introduced by applying the proposition remians polynomially bounded; they can formally be added to the θ variables of the statement.

Now Theorem 6 implies the existence of an OBDD representation of $\varphi_n(e,\theta)$ of polynomial size in n. Taking into account the role the θ variables are playing this OBDD is a non-deterministic polynomial-size OBDD for computing the function $PERM_n$. However, the existence of such an OBDD contradicts Theorem 7. $\square$

The question answered negatively by Theorem 8 for tree-width can be posed as well in relation to the clique-width parameter. That is: Can the permanent function be described via CNF formulas of bounded clique-width and polynomial size? Next we relate this question to Theorem 2 b) and show that such a representation is only possible if the conjecture $\#P \not\subseteq FP/poly$ fails to be true.

Theorem 9. *Suppose there is a family $\{\varphi_n\}_n$ of CNF formulas of polynomial size such that all $I(\varphi_n)$ are of clique-width at most k for some fixed k and for each $Y \in \{0,1\}^{n^2}$ we have that $\varphi_n(Y)$ holds iff Y is a permutation matrix. Then $\#P \subseteq FP/poly$.*

Proof. Suppose $\{\varphi_n\}$ is given as in the assumption. For a matrix $X \in \{0,1\}^{n^2}$ we shall construct from φ_n and a parse-tree of it (given as non-uniform advice) another CNF formula $\psi_n^X(Y)$ of bounded clique-width together with a parse-tree for ψ_n^X such that

$$Perm(X) = \sum_{Y\in\{0,1\}^{n^2}} \psi_n^X(Y).$$

Theorem 2 b) then implies that the latter can be computed in polynomial time. Given $\#P$-completeness of the permanent function on 0-1-matrices the claim follows. The construction of ψ_n^X works as follows. We have $Perm(X) = \sum\limits_{Y\in\{0,1\}^{n^2}} \varphi_n(Y)\cdot X^Y$, where $X^Y = \prod\limits_{i,j} x_{i,j}^{y_{i,j}}$ and $x_{i,j}^{y_{i,j}} = \begin{cases} x_{i,j} & \text{if } y_{i,j}=1 \\ 1 & \text{if } y_{i,j}=0 \end{cases}$. We replace the monomial X^Y by the conjunctions $\bigwedge\limits_{i,j}(x_{i,j} \vee \neg y_{i,j})$. The clause graph $I(\psi_n)$ of the CNF formula

$$\psi_n(X,Y) \equiv \varphi_n(Y) \wedge \bigwedge_{i,j}(x_{i,j} \vee \neg y_{i,j})$$

can easily be seen to have clique-width $\leq k+2$. Each time when in the clique-width construction of $I(\varphi_n)$ along the parse-tree a node $y_{i,j}$ is created, in the corresponding construction for $I(\psi_n)$ two new nodes for $x_{i,j}$ and the clause $D_{i,j} := x_{i,j} \vee \neg y_{i,j}$ are created with an own label each. Then $D_{i,j}$ is connected to both $x_{i,j}$ and $y_{i,j}$ (respecting the necessary signs of the edges). Finally the labels for $D_{i,j}$ and $x_{i,j}$ are removed again.

Now for a fixed given matrix X we plug the values of the $x_{i,j}$ into the CNF formula $\psi_n(X, Y)$. Clauses that are satisfied by the assignment are removed. In clauses that are not satisfied by the assignment all occurences of $x_{i,j}$ literals are removed. That way a new CNF formula ψ_n^X is obtained. The clause graph $I(\psi_n^X)$ results from $I(\psi_n)$ by removing certain nodes, namely the $x_{i,j}$ as well as some clause nodes. This operation clearly does not increase the clique-width. □

The result holds as well if we allow additional variables in $\varphi_n(Y)$ as in the statement of Theorem 8. It remains an open question whether Theorem 9 can be strengthened to hold unconditionally:

*Conjecture:*There is no family $\{\varphi_n\}_n$ of CNF formulas of polynomial size with all $I(\varphi_n)$ of bounded clique-width such that $\varphi_n(Y) \Leftrightarrow Y$ is a permutation matrix.

We will show in the full version of this paper that Theorem 8 holds even without the polynomial size hypothesis on φ_n.

References

1. Bodlaender, H.L.: NC-algorithms for graphs with small tree-width. In: van Leeuwen, J. (ed.) WG 1988. LNCS, vol. 344, pp. 1–10. Springer, Heidelberg (1989)
2. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear Time Solvable Optimization Problems on Graphs of Bounded Clique Width. Theory of Computing Systems 33(2), 125–150 (2000)
3. Ferrara, A., Pan, G., Vardi, M.Y.: Treewidth in Verification: Local vs. Global. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS, vol. 3835, pp. 489–503. Springer, Heidelberg (2005)
4. Fischer, E., Makowsky, J., Ravve, E.V.: Counting Truth Assignments of Formulas of Bounded Tree-Width or Clique-Width. Disc. Appl. Mathematics 154(4), 511–529 (2008)
5. Flarup, U., Koiran, P., Lyaudet, L.: On the expressive power of planar perfect matching and permanents of bounded tree-width matrices. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 124–136. Springer, Heidelberg (2007)
6. Jukna, S.: The effect of null-chains on the complexity of contact schemes. In: Csirik, J.A., Demetrovics, J., Gecseg, F. (eds.) FCT 1989. LNCS, vol. 380, pp. 246–256. Springer, Heidelberg (1989)
7. Karpinski, M.: On the computation power of randomized branching programs, Electronic Colloquium on Computational Complexity, Report Nr. 38 (1998)
8. Krause, M., Meinel, C., Waack, S.: Separating the eraser Turing machine classes $L_e, NL_e, co-NL_e$ and P_e. Theoretical Computer Science 86, 267–275 (1991)
9. Samer, M., Szeider, S.: Algorithms for Propositional Model Counting. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS, vol. 4790, pp. 484–498. Springer, Heidelberg (2007)
10. Wegener, I.: Branching Programs and Binary Decision Diagrams: Theory and Applications. SIAM Monographs on Discrete Mathematics and Applications (2000)

Planar Feedback Vertex Set and Face Cover: Combinatorial Bounds and Subexponential Algorithms

Athanassios Koutsonas[⋆] and Dimitrios M. Thilikos[⋆⋆]

Department of Mathematics, National and Kapodistrian University of Athens
Panepistimioupolis, GR-15784 Athens, Greece
{akoutson,sedthilk}@math.uoa.gr

Abstract. The Planar Feedback Vertex Set problem asks, whether an n-vertex planar graph contains at most k vertices meeting all its cycles. The Face Cover problem asks, whether all vertices of a plane graph G lie on the boundary of at most k faces of G. Standard techniques from parameterized algorithm design indicate, that both problems can be solved by sub-exponential parameterized algorithms (where k is the parameter). In this paper, we improve the algorithmic analysis of both problems by proving a series of combinatorial results, relating the branchwidth of planar graphs with their face cover. Combining this fact with duality properties of branchwidth, allows us to derive analogous results on feedback vertex set. As a consequence, it follows that Planar Feedback Vertex Set and Face Cover can be solved in $O(2^{15.11\cdot\sqrt{k}} + n^{O(1)})$ and $O(2^{10.1\cdot\sqrt{k}} + n^{O(1)})$ steps, respectively.

1 Introduction

In this paper, we offer improved algorithmic analysis for two widely studied combinatorial problems on planar graphs. The first is the planar version of Feedback Vertex Set that asks, given a graph and a non-negative integer k, whether all cycles of G can be blocked by a set of at most k vertices. The second is the Face Cover that asks, given a plane graph G and a non-negative integer k whether the boundary of at most k faces contains all the vertices of G. Our aim is to show that both problems are closely related and to use this fact to improve the analysis of algorithms for both problems.

The Feedback Vertex Set, as well as its directed version, are one of the most studied *NP*-complete problems, mainly due to their numerous applications (see [12]). A wide range of algorithmic results on Feedback Vertex Set have been proposed including approximation algorithms [5,18,17], exact algorithms [14] and heuristics [21].

[⋆] The research of the first author is part of his M.Sc. Thesis for the Graduate Program in Logic, Algorithms, and Computation ($\mu\prod\lambda\forall$).

[⋆⋆] Supported by the Project "Kapodistrias" (Α*Π* 736/24.3.2006) of the National and Kapodistrian University of Athens (project code: 70/4/8757).

In our study, we focus our attention on the parameterized complexity of both Planar Feedback Vertex Set and Face Cover. In a parameterized problem the input is being seen as a pair (I, k), where I is the main part of the problem and k is a parameter. A *fixed parameter* algorithm (or simply FPT-*algorithm*) is one that solves the problem in $O(f(k) \cdot |I|^{O(1)})$ steps where f is a function depending exclusively on the parameter k (for more on parameterized complexity and algorithms, see [13,22]). For both Feedback Vertex Set and Face Cover, we consider their parameterized versions, where k is the parameter. Many FPT-algorithms were proposed for Feedback Vertex Set. The best current results in this direction are the $O(4^k kn)$ step probabilistic algorithm in [1] and the $O(5^k kn^2)$ step algorithm in [4] (throughout the paper and for both problems, we denote by n the size of the input graph).

When restricted to planar graphs, both Planar Feedback Vertex Set and Face Cover are solvable by subexponential FPT-algorithms, i.e. algorithms running in $O(2^{o(k)} \cdot n^{O(1)})$ steps. The first results of this kind were given by Kloks et al. in [20] for both problems. Furthermore, Fernau and Juedes proved that Face Cover can be solved in $O(2^{24.551\sqrt{k}} \cdot n)$ steps [11]. All previous results can be improved using the win/win meta-algorithmic framework emerging from the bidimensionality theory in [6]. In this paper, we proceed to a "taylor made" analysis of the complexity of both Planar Feedback Vertex Set and Face Cover. In fact, we unify the analysis of both problems, by exploiting a duality relation between them. As a consequence, we prove that Planar Feedback Vertex Set and Face Cover can be solved in $O(2^{15.11 \cdot \sqrt{k}} \cdot n + n^3)$ and $O(2^{10.1 \cdot \sqrt{k}} \cdot n + n^3)$ steps, respectively. Our analysis resides in a thorough analysis of the structure of face covers in planar graphs, which leads to combinatorial bounds of independent interest.

The presentation of the paper is divided in two parts. First, in Section 2, we present the main algorithmic techniques, as well as our approach for proving the claimed complexity bounds. Next, in Section 3, we prove the combinatorial bounds supporting the results of Section 2. Finally, in Section 4, we conclude with some open problems.

2 The Algorithms

Definitions and preliminaries. We consider graphs that may have loops or multiple edges. If a graph has no multiple edges or loops we call it *simple*. Given a graph G, we denote as $V(G)$ its vertex set and as $E(G)$ its edge multiset. For any set $S \subseteq V(G)$, we denote as $G[S]$ the subgraph of G induced by the vertices in S. We also denote as $G \setminus S$ the graph $G[V(G) - S]$ and if $v \in V(G)$ we also write $G \setminus v$ instead of $G \setminus \{v\}$. Finally, if $e \in E(G)$, we write $G \setminus e$ instead of $(V(G), E(G) - \{e\})$.

We use the term *plane graph* for a planar graph along with an embedding of it in the sphere $\mathbb{S}_0$ without crossings. To simplify notations, we do not distinguish between a vertex of the graph and the point of $\mathbb{S}_0$ used in the drawing to represent the vertex or between an edge and the open line segment representing it. We denote as $F(G)$, the set of faces of this embedding, i.e. the connected components of $\mathbb{S}_0 \setminus G$, (that are open sets of $\mathbb{S}_0$). We also use the notation G^* to denote an embedding of the dual graph of G.

Given a plane graph G with at least one edge, we define its *radial graph* R_G as the plane graph whose vertex set is $V(G) \cup V(G^*)$ and whose edges are defined as follows: Let $\mathcal{C} = \{C_1, \ldots, C_r\}$ be the connected components of $\mathbb{S}_0 \setminus (G \cup G^*)$ and observe that for $i = 1, \ldots, r$ C_i is an open set whose boundary contains one vertex, say v_i, from $V(G)$ and one vertex, say u_i, from $V(G^*)$. The edge set of R_G is the set $E(R_G) = \{\{v_i, u_i\}, i = 1, \ldots, r\}$ where edge $\{v_i, u_i\}$ has multiplicity 1 if both v_i and u_i have degree at least 2 in G and G^* respectively, otherwise its multiplicity is 2 (clearly, $\{v_i, u_i\}$ can be seen as a subset of the open set C_i). Notice that $R_G = (V(G) \cup F(G), E(R_G))$ is a bipartite graph, whose parts are the vertex and face sets of G, respectively.

A vertex set $S \subseteq V(G)$ is a *feedback vertex set* of a graph G, if the graph $G \setminus S$ is acyclic. The *feedback vertex set number* of a graph G, denoted as $\mathbf{fvs}(G)$, is the minimum size of a feedback vertex set of G. A *Face Cover* of a plane graph G is a set $R \subseteq F(G)$ of faces, such that all vertices of G are lying on the boundary of some face in R. We define the *face cover number* of a plane graph G, as the minimum size of a face cover of G and we denote it as $\mathbf{fc}(G)$. We consider the following two parameterized problems.

Planar Feedback Vertex Set
Instance: A planar graph G and a non-negative integer k
Parameter: k
Question: $\mathbf{fvs}(G) \leqslant k$?
Face Cover
Instance: A plane graph G and a non-negative integer k
Parameter: k
Question: $\mathbf{fc}(G) \leqslant k$?

Branch decompositions. Let G be a graph on n vertices. A *branch decomposition* (T, μ) of a graph G consists of an unrooted ternary tree T (i.e. all internal vertices are of degree three) and a bijection $\mu : L \to E(G)$ from the set L of leaves of T to the edge set of G. We define for every edge e of T the *middle set* $\omega(e) \subseteq V(G)$, as follows: Let T_1 and T_2 be the two connected components of $T \setminus e$. Then, let G_i be the graph induced by the edge set $\{\mu(f) : f \in L \cap V(T_i)\}$ for $i \in \{1, 2\}$. The *middle set* is the intersection of the vertex sets of G_1 and G_2, i.e. $\omega(e) = V(G_1) \cap V(G_2)$. The *width* of (T, μ) is the maximum order of the middle sets over all edges of T (in case T has no edges, then the width of (T, μ) is equal to 0). An optimal branch decomposition of G is defined by the tree T and the bijection μ which give the minimum width, the *branchwidth*, denoted by $\mathbf{bw}(G)$. Given two graphs H and G, we write $H \leqslant G$, when H can occur from a subgraph of G after a series of edge contractions. It is known from [23], that if $H \leqslant G$, then $\mathbf{bw}(H) \leqslant \mathbf{bw}(G)$. The following theorem follows from Theorem (7.2) of [24] and will be useful for our analysis.

Theorem 1. *Let G be a non-ayclic planar graph and G^* be a dual of G. Then, the branch-width of G is equal to the branch-width of G^*.*

In Section 3, we prove the following theorem on the relation between the branch-width of a plane graph and the branchwidth of its radial.

Theorem 2. *For any plane graph G containing a vertex of degree at least 2, it holds that $\mathbf{bw}(R_G) \leqslant 2 \cdot \mathbf{bw}(G)$.*

The Win/Win approach. The standard technique for the design of subexponential parameterized algorithms for graph parameters on planar graphs, relies on two conditions: the existence of a sublinear combinatorial bound for the branch-width in terms of the parameter and dynamic programming on branch decompositions. In particular, we refer to any graph parameter $\mathbf{p}$, for which there exist two positive real numbers α and β, such that:

(**a**) For any planar graph G, $\mathbf{bw}(G) \leq \alpha \cdot \sqrt{\mathbf{p}(G)}$.
(**b**) For every planar graph G and given an optimal branch decomposition (T, μ) of G, $\mathbf{p}(G)$ can be computed in $O(2^{\beta \cdot \mathbf{bw}(G)} \cdot n)$ steps.

Theorem 3 ([10, Theorem 1]). *If conditions* (**a**) *and* (**b**) *above are satisfied for some parameter* $\mathbf{p}$ *and some α and β, then one can construct an an algorithm that checks whether $\mathbf{p}(G) = k$ in $O(2^{\alpha \cdot \beta \cdot \sqrt{k}} n)$ steps.*

As proved in [24] and [19], there is an algorithm computing an optimal branch decomposition of planar graphs in $O(n^3)$ steps. Thus whenever we apply Theorem 3 without assuming the existence of an optimal branch decomposition, we should add an additive overhead of $O(n^3)$ steps.

According to the results in [6], conditions (a) and (b) are satisfied for both **fvs** and **fc**. Therefore Theorem 3 can be applied for these parameters. We define $\alpha_{\mathbf{fvs}}$ ($\alpha_{\mathbf{fc}}$) and $\beta_{\mathbf{fvs}}$ ($\beta_{\mathbf{fc}}$) as the minimum values for α and β, for which Condition (a) and (b), respectively, holds for **fvs** (**fc**). In what follows, we provide bounds to these constants towards improving the time analysis of the algorithm in Theorem 3.

Estimating $\beta_{\mathbf{fvs}}$ and $\beta_{\mathbf{fc}}$. Regarding Condition (b), and in case of **fvs**, it is known that given an optimal branch decomposition of a n-vertex planar graph G, there is a dynamic programming algorithm that computes $\mathbf{fvs}(G)$ in $O(2^{3.56\mathbf{bw}(G)} \cdot n)$ steps [8]. We conclude, that condition (b) holds for $\beta_{\mathbf{fvs}} \leqslant 3.56$.

To estimate $\beta_{\mathbf{fc}}$, we use the well known reduction of the Face Cover problem to the following problem:

Planar Blue-Red Dominating Set
Instance: A planar bipartite graph G with parts B and R and a non-negative integer k.
Parameter: k

Question: Is there a vertex set $D \subseteq R$ where $|D| \leqslant k$ and such that every vertex in B has a neighbour in D?

Observe that (G, k) is a yes-instance of Face Cover, if and only if (R_G, k) is a yes-instance of Planar Blue-Red Dominating Set (set $B = V(G)$ and $R = F(G)$) (see also [11]). From [9, Theorem 2.3.2], Planar Blue-Red Dominating Set can be solved in $O(2^{1.19 \cdot \mathbf{bw}(G)} \cdot |V(G)|)$ steps, provided that an optimal branch decomposition is given. Combining this fact with Theorem 2, we can derive that Condition (b) is satisfied for $\beta_{\mathbf{fc}} \leqslant 2.38$.

Easy bounds for $\alpha_{\mathbf{fvs}}$ and $\alpha_{\mathbf{fc}}$. Condition (a) follows directly from the theory of bidimensionality introduced in [6]. Applying the meta-algorithmic result of [6] (Theorem 4.14) for both parameters **fvs** and **fc**, condition (a) holds for $\alpha_{\mathbf{fvs}}, \alpha_{\mathbf{fc}} \leqslant 8$. This implies the existence of an $O(2^{28.48 \cdot \sqrt{k}} \cdot n + n^3)$ step algorithm for the Planar Feedback Vertex Set problem (to our knowledge, no other exact bound for this problem exists) and the existence of an $O(2^{19.04 \cdot \sqrt{k}} \cdot n + n^3)$ step algorithm for the Face Cover problem (improving the constants of [11] for the same problem).

The above estimations for $\alpha_{\mathbf{fvs}}$ and $\alpha_{\mathbf{fc}}$ can be easily further improved using known results. Kloks et al. [20] proved that for any planar graph G, there is a planar graph H containing G as a subgraph such that $\mathbf{ds}(H) \leqslant \mathbf{fvs}(G)$ (here by $\mathbf{ds}(H)$ we denote the minimum size of a dominating set of H). Moreover it holds that for any planar graph H, $\mathbf{bw}(H) \leqslant 6.364\sqrt{\mathbf{ds}(H)}$ [15]. As $\mathbf{bw}(G) \leqslant \mathbf{bw}(H)$, we obtain that $\mathbf{bw}(G) \leqslant 6.364\sqrt{\mathbf{fvs}(G)}$ and this yields Condition (a) for $\alpha_{\mathbf{fvs}} \leqslant 6.364$. For $\alpha_{\mathbf{fc}}$, we need to make the following observation: Suppose that a plane graph G has a face cover $U \subseteq F(G)$ of size $\leqslant k$. Let H be the graph obtained from G, if for each $f \in U$ we draw a vertex v_f inside f and connect it with the vertices incident to f. Notice that the new vertices constitute a dominating set of H, of size at most k. Again, from the result of [15], we conclude that $\mathbf{bw}(G) \leqslant \mathbf{bw}(H) \leqslant 6.364 \cdot \sqrt{k}$, thus $\alpha_{\mathbf{fc}} \leqslant 6.364$.

According to the above, there is an $O(2^{22.66 \cdot \sqrt{k}} \cdot n + n^3)$ step algorithm for the Planar Feedback Vertex Set problem and an $O(2^{15.15 \cdot \sqrt{k}} \cdot n + n^3)$ step algorithm for the Face Cover problem. To our knowledge, these are the fastest, so far, algorithms for these problems.

Better bounds for $\alpha_{\mathbf{fvs}}$ and $\alpha_{\mathbf{fc}}$. In order to find better bounds for $\alpha_{\mathbf{fvs}}$ and $\alpha_{\mathbf{fc}}$, we should focus our attention to the structure of the corresponding parameters. In fact, face cover and planar feedback vertex set are closely related in dual graphs. Informally speaking, the "dual" version of the face cover number is upper bounded by the vertex feedback set number. In particular we observe the following:

Lemma 1. *Let G and G^* be dual plane graphs that are not forests. Then, $\mathbf{fc}(G^*) \leqslant \mathbf{fvs}(G)$.*

Proof. Let $S \subseteq V(G)$ be a feedback vertex set in G, with $|S| \leqslant k$. As the boundary of any face $f \in F(G)$ contains a cycle of G, it also contains a vertex

$v \in S$. This implies that any vertex $f^* \in V(G^*)$ of G^* is in the boundary of some face v^* of S^*, where $S^* \subseteq F(G^*)$ is the set of the duals of the vertices in S. Therefore S^* is a face cover of G^*.

The above lemma, combined with the duality of branchwidth (Theorem 1), yields that $\mathbf{bw}(G) = \mathbf{bw}(G^*) \leqslant \alpha_{\mathbf{fc}}\sqrt{\mathbf{fc}(G^*)} \leqslant \alpha_{\mathbf{fc}}\sqrt{\mathbf{fvs}(G)} \Rightarrow \alpha_{\mathbf{fvs}} \leqslant \alpha_{\mathbf{fc}}$ (we examine the non-trivial case where G is not a forest). Therefore, any improvement in the estimation of $\alpha_{\mathbf{fc}}$ reflects to $\alpha_{\mathbf{fvs}}$ as well. Section 3 will be devoted to the improvement of the bound for $\alpha_{\mathbf{fc}}$. In particular (in Section 4) we prove the following Theorem.

Theorem 4. *For any planar graph G, $\mathbf{bw}(G) \leq 2 \cdot \sqrt{4.5} \cdot \sqrt{\mathbf{fc}(G)}$.*

Theorem 4, implies that $\alpha_{\mathbf{fvs}} \leqslant \alpha_{\mathbf{fc}} \leqslant 4.243$. This fact leads to the main result of this paper:

Theorem 5. Planar Feedback Vertex Set *and* Face Cover *can be solved in $O(2^{15.11\cdot\sqrt{k}} \cdot n + n^3)$ and $O(2^{10.1\cdot\sqrt{k}} \cdot n + n^3)$ steps, respectively.*

We stress, that according to Bodlaender [2] (see also [3]) there exists a polynomial algorithm producing a $O(k^3)$ size kernel for the Feedback Vertex Set problem, when parameterized by k (i.e. an equivalent instance of the problem where the input graph has at most $O(k^3)$ vertices). Combining this fact with Theorem 5, we derive the existence of an $O(2^{15.11\cdot\sqrt{k}}) + n^{O(1)}$ algorithm for Planar Feedback Vertex Set. For the Face Cover, a $O(k^2)$ kernel has been reported in [20]. Therefore, Face Cover can be solved in $O(2^{10.1\cdot\sqrt{k}}) + n^{O(1)}$ steps.

Planar cycle packing. Our combinatorial bounds can be useful for computing other parameters that can be bounded by the face cover or the feedback vertex set. An example of such a parameter is the cycle packing number, denoted as $\mathbf{cp}(G)$, that is the maximum number of disjoint cycles in a graph G. According to the results of [9,8], computing $\mathbf{cp}(G)$ on planar graphs can be done in $O(2^{2.78\cdot\mathbf{bw}(G)} \cdot n)$ steps. Therefore, Condition (b) holds for $\mathbf{cp}$ when $\beta \leqslant 2.78$. Recall that in Section 2 we proved that for any planar G, $\mathbf{bw}(G) \leqslant 2\cdot\sqrt{4.5 \cdot \mathbf{fvs}(G)}$. According to Kloks et al. in [20], for any planar graph G, it holds that $\mathbf{fvs}(G) \leqslant 5 \cdot \mathbf{cp}(G)$. We conclude that for any planar G, $\mathbf{bw}(G) \leqslant 2 \cdot \sqrt{4.5 \cdot 5 \cdot \mathbf{cp}(G)}$ and thus Condition (a) holds for $\mathbf{cp}$ for $\alpha \leqslant 9.49$. By Theorem 3, the Planar Cycle Packing can be solved in $O(2^{26.347\cdot\sqrt{k}} \cdot n)$ steps.

3 Bounding Branchwidth

Given a sphere $\mathbb{S}_0$ and a subset $\Delta \subseteq \mathbb{S}_0$, the *closure* of Δ is denoted by $\overline{\Delta}$, and the boundary of Δ is $\widehat{\Delta} = \overline{\Delta} \cap \overline{\mathbb{S}_0 - \Delta}$. Given a graph $G = (V, E)$ drawn in $\mathbb{S}_0$, we call *noose*, a Jordan curve in $\mathbb{S}_0$ that meets the drawing only in vertices of G. For a noose N passing through vertices $v_1, v_2, \ldots, v_n$ we will use the same notation we use for a cycle of a graph, i.e. $N = v_1 v_2 \ldots v_n v_1$. The *length* $|N|$ of a noose N is the number of vertices it meets.

Plane graphs and hypergraphs. Hypergraphs will be an important ingredient of the proofs of this section. Our first step is to extend some of the basic concepts previously defined, for the case of hypergraphs. Let us, before that, denote the number of the vertices of a hyperedge as the *arity* of the hyperedge. We insist in still calling edges, hyperedges of arity equal to two (i.e. those that have only two endpoints). Notice that the definition of branch decomposition and branchwidth applies directly for hypergraphs. Therefore, we use the notation $\mathbf{bw}(H)$, also when H is a hypergraph. We call *plane hypergraph*, any hypergraph H whose vertices are those of a plane graph G, and whose hyperedges are some of the edges of G, plus some new pairwise distinct hyperedges, each containing the boundary vertices of some of the faces of G. By construction, H has an embedding in $\mathbb{S}_0$ that copies the one of G and where hyperedges are drawn inside the corresponding faces of G.In this case, we say that the plane graph G *generates* the plane hypergraph H. The proof of the following lemma is easy.

Lemma 2. *Let G be a plane graph and let H be a hypergraph generated by G. Then $\mathbf{bw}(G) \leqslant \mathbf{bw}(H)$.*

Given a plane hypergraph H, we define its dual H^* as the hypergraph whose vertices are the faces of H and where each hyperedge e of H corresponds to a hyperedge e^* of H^* whose endpoints are the faces of H that are incident to e. By drawing each vertex of H^* inside the corresponding face of H, one can see that H^* is also a plane hypergraph.

In the rest of this section, we will consider only plane hypergraphs generated by simple 2-connected planar graphs. This permits us to consider the hyperedges of a plane hypergraph as closed disks whose boundary vertices are their endpoints. To simplify notations, while working with plane hypergraphs we will not distinguish between a vertex of the graph and the point of the sphere $\mathbb{S}_0$ used in the drawing to represent the vertex, or between a edge (hyperedge) and the closed line segment (closed disk) representing it in the embedding. Using this convention, we can define the set of faces of a hypergraph H as the set of connected components of $\mathbb{S}_0 - H$. It is now clear that the notion of a face cover naturally extends for plane hypergraphs.

Let G be a 2-connected plane graph and let R_G be its radial graph. Notice that, as G is 2-connected, all faces of R_G are squares (i.e. their boundaries are cycles of length 4). We define $\tilde{R}_G$ as the plane hypergraph generated by R_G, if we remove all the edges of R_G and add a hyperedge for each face of R_G. The proof of the following Lemma is omitted.

Lemma 3. *For any 2-connected plane graph G, $\mathbf{bw}(\tilde{R}_G) \leqslant 2 \cdot \mathbf{bw}(G)$.*

Notice that Lemmata 2 and 3 imply Theorem 2 claimed in Section 2.

Normalization. Given a plane graph G and a face cover S_G of it, we will refer to the faces in $S_G \subseteq F(G)$ as *$\mathcal{FC}$-faces.* We say that two $\mathcal{FC}$-faces f_1 and f_2 *touch* if , $\widehat{f_1} \cap \widehat{f_2} \neq \emptyset$. Two vertices will be called a *pair*, if they are adjacent and lie on the same $\mathcal{FC}$-face. We call a face of G *triangle* if its boundary is a cycle of length 3. We call an edge of G *bridge* if there are $\mathcal{FC}$-faces f_1 and f_2 such that e it is the unique edge whose endpoints belong in the boundaries of f_1 and f_2.

Let f_1, f_2 be two $\mathcal{FC}$-faces and let x_1, x_2, y_1, y_2 be four vertices, such that $x_i, y_i \in \hat{f}_i, i = 1, 2)$; a noose of the form $x_1 y_1 x_2 y_2 x_1$, will be called a *4-noose.* As a Jordan curve, a 4-noose N bounds two closed discs. If one of them contains exactly one hyperedge, whose endpoints are the vertices of N, then we refer to such a 4-noose as *trivial.* We proceed to the first lemma on the structure of the graph (the proof is omitted).

Lemma 4. *Let G be a simple 3-connected plane graph, such that $\mathbf{fc}(G) \leqslant k$. Then, there exists a (plane) graph G' and a face cover $S_{G'}$ of G', such that: (a) $\mathbf{bw}(G) \leqslant \mathbf{bw}(G')$, (b) $|S_{G'}| \leqslant k$, (c) G' is simple and 3-connected, (d) No two different $\mathcal{FC}$-faces touch, (e) G' does not contain any bridge, and (f) A face of G' is either a $\mathcal{FC}$-face, or a square whose boundary contains two pairs of two different $\mathcal{FC}$-faces, or a triangle incident to three different vertices, that in turn are incident to three different $\mathcal{FC}$-faces.*

We call a face of a plane hypergraph H *degenerate*, if it is bounded by exactly two hyperedges of H.

Lemma 5. *Let G be a 3-connected simple graph such that $\mathbf{fc}(G) \leqslant k$. Then, there exists a hypergraph H and a face cover S_H of H with size at most k, such that: (a) $\mathbf{bw}(G) \leqslant \mathbf{bw}(H)$, (b) No two different $\mathcal{FC}$-faces touch. (c) H contains no edges and each hyperedge of H has arity 4 and contains two disjoint pairs that are incident to two different $\mathcal{FC}$-faces. (d) A face of H is either a non-degenerate $\mathcal{FC}$-face or a degenerate face or a triangle incident to three different vertices that in turn are incident to three different $\mathcal{FC}$-faces.*

Proof. Let G' be a planar hypergraph and $S_{G'}$ a face cover of G', as in Lemma 4. Let also H be the plane hypergraph generated by G', if we remove all edges and add a hyperedge for each square of G'. It is not hard to verify that H is the required plane hypergraph. □

A plane hypergraph H with a face cover S_H, satisfying properties (b) - (d) of Lemma 5, will be characterized, from now on, as *normalized.* The proof of the following lemma is in the Appendix.

Lemma 6. *Let H be a normalized hypergraph with face cover S_H and let N a non-trivial 4-noose bounding the closed discs Δ_1, Δ_2. Let also H_i, $(i = 1, 2)$ be the subgraph of H containing all vertices and edges included in Δ_i, plus the edge $\tilde{e}$ with endpoints the four vertices the 4-noose passes through. Then, H_i (for $i = 1, 2$) is a normalized graph with $\mathbf{fc}(H_i) \leqslant \mathbf{fc}(H)$ and less vertices than H.*

Prime hypergraphs. A normalized hypergraph H will be denoted as *prime*, if every 4-noose is trivial. Let H be a prime hypergraph and S_H a face cover of H with $|S_H| \geqslant 3$. We define its *reduced* graph $\mathbf{red}(H)$ as follows: There is a bijection $\phi_v : S_H \to V(\mathbf{red}(H))$ and a bijection $\phi_e : E(H) \to E(\mathbf{red}(H))$, such that two vertices in $x, y \in V(\mathbf{red}(H))$ are joined by an edge in $E(\mathbf{red}(H))$, if and only if there is a hyperedge with vertices lying on the faces $\phi_v^{-1}(x)$ and $\phi_v^{-1}(y)$. The proof of the next lemma is omitted.

Lemma 7. *Let H be a prime hypergraph with $\mathbf{fc}(H) \geqslant 3$. Then, the graphs H^* and $\tilde{R}_{\mathbf{red}(H)}$ are isomorphic.*

Corollary 1. *If H is a prime hypergraph, then $\mathbf{bw}(H) \leqslant 2 \cdot \sqrt{4.5 \cdot \mathbf{fc}(H)}$.*

Proof. If $\mathbf{fc}(H) = 2$, then H is the graph of 6 vertices - three on each disk - with 3 hyperedges of arity of four between these vertices. It is $\mathbf{bw}(H) = 4 \leqslant 2 \cdot \sqrt{4.5 \cdot 2}$. Suppose now, that S_H is a face cover of H where $3 \leqslant |S_H| = \mathbf{fc}(H)$ and notice that $\mathbf{red}(H)$ contains $|S_H|$ vertices. From the main result in [16], any n-vertex plane graph has branchwidth bounded by $\sqrt{4.5 \cdot n}$. Applying this result on $\mathbf{red}(H)$ we have that $\mathbf{bw}(\mathbf{red}(H)) \leqslant \sqrt{4.5 \cdot \mathbf{fc}(H)}$. Also, applying [24, Theorem (7.2)] on H and H^* it follows that $\mathbf{bw}(H) = \mathbf{bw}(H^*)$. From Lemmata 3 and 7, we obtain that $\mathbf{bw}(H) = \mathbf{bw}(H^*) = \mathbf{bw}(\tilde{R}_{\mathbf{red}(H)}) \leqslant 2 \cdot \mathbf{bw}(\mathbf{red}(H)) \leqslant 2 \cdot \sqrt{4.5 \cdot \mathbf{fc}(H)}$. □

Applying inductively Lemma 6 and using Corollary 1 along with [15, Lemma 3.1] we can prove the following (the full proof is omitted).

Lemma 8. *Let H be a normalized graph. Then $\mathbf{bw}(H) \leqslant 2 \cdot \sqrt{4.5 \cdot \mathbf{fc}(H)}$.*

We are now ready to prove the main combinatorial result of this paper.

Proof (of Theorem 4). We can assume that $\mathbf{fc}(G) \geqslant 2$, as otherwise G is either a forest or an outerplanar graph, implicating that $\mathbf{bw}(G) \leqslant 2$ yielding trivially the result. Also, we can assume that G is simple as the removal of loops or multiples edges may reduce the branchwidth of a graph by at most 2 and this only in the case where the resulting graph is a forest. We will use induction on $|V(G)|$. For the smallest graph with $\mathbf{fc}(G)$ at least two, namely the K_4, the upper bound is true. We assume the same for any graph with less than $n > 4$ vertices and we show that it holds also for any n-vertex graph. If the graph G is 3-connected, then by Lemmata 4 and 5, there is a hypergraph H where $\mathbf{fc}(H) \leqslant \mathbf{fc}(G)$ and $\mathbf{bw}(G) \leqslant \mathbf{bw}(H)$ and the result follows from Lemma 8. Let us assume that G is not 3-connected. Then it has a separator of at most two vertices. We will describe the case where the minimum separator has two vertices x and y as, otherwise, the result follows by applying Lemma [15, Lemma 3.1] to the (bi-)connected components of G. Let C be some of the connected components of $G[V(G) - \{x, y\}]$. We set $G_1 = G[V(C) \cup \{x, y\}]$ and $G_2 = G[V(G) - V(C)]$ and we add in both G_1 and G_2 the edge $e = \{x, y\}$ (if its does not already exists). Notice that $G_i \leqslant G$, therefore $\mathbf{fc}(G_i) \leqslant \mathbf{fc}(G)$. By the induction hypothesis, $\mathbf{bw}(G_i) \leqslant 2 \cdot \sqrt{4.5 \cdot \mathbf{fc}(G_i)}$ and the result follows applying Lemma [15, Lemma 3.1] for G_1 and G_2. □

4 Conclusion

According to the Win/win approach, the algorithmic analysis of all problems examined in this paper is reduced to the problem of bounding the decomposability of a planar graph (i.e. the branchwidth) by a sublinear function of the parameter. While such general (but not optimal) upper bounds are provided by bidimensionality theory [6] better constants (and thus faster algorithms) have been achieved by a "tailor made" analysis of the parameter in the cases of *vertex cover*, *edge dominating set*, and *dominating set* (see [7,15]). Our results for *feedback vertex set*, *face cover*, and *cycle packing* offer to the same line of research. Furthermore, specific combinatorial similarities between our proofs in Section 3 and the proofs in [7,15], make us believe, that a generic technique for wider families of problems may exist.

References

1. Becker, A., Bar-Yehuda, R., Geiger, D.: Randomized algorithms for the loop cutset problem. J. Artificial Intelligence Res. 12, 219–234 (2000) (electronic)
2. Bodlaender, H.L.: A cubic kernel for feedback vertex set. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 320–331. Springer, Heidelberg (2007)
3. Burrage, K., Estivill-Castro, V., Fellows, M., Langston, M., Mac, S., Rosamond, F.: The undirected feedback vertex set problem has a Poly(k) kernel. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 192–202. Springer, Heidelberg (2006)
4. Chen, J., Fomin, F.V., Liu, Y., Lu, S., Villanger, Y.: Improved algorithms for the feedback vertex set problems. Technical Report 348, Department of Informatics, University of Bergen, Bergen, Norway (2007)
5. Chudak, F.A., Goemans, M.X., Hochbaum, D.S., Williamson, D.P.: A primal-dual interpretation of two 2-approximation algorithms for the feedback vertex set problem in undirected graphs. Oper. Res. Lett. 22(4-5), 111–118 (1998)
6. Demaine, E.D., Fomin, F.V., Hajiaghayi, M., Thilikos, D.M.: Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. J. ACM 52(6), 866–893 (2005) (electronic)
7. Demaine, E.D., Hajiaghayi, M., Thilikos, D.M.: Exponential speedup of fixed-parameter algorithms for classes of graphs excluding single-crossing graphs as minors. Algorithmica 41(4), 245–267 (2005)
8. Dorn, F.: Dynamic programming and fast matrix multiplication. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 280–291. Springer, Heidelberg (2006)
9. Dorn, F.: Designing Subexponential Algorithms: Problems, Techniques & Structures. PhD thesis, Department of Informatics, University of Bergen (2007)
10. Dorn, F., Fomin, F.V., Thilikos, D.M.: Subexponential parameterized algorithms. Comp, Sc. Rev. 2(1), 29–39 (2008)
11. Fernau, H., Juedes, D.: A geometric approach to parameterized algorithms for domination problems on planar graphs. In: Fiala, J., Koubek, V., Kratochvíl, J. (eds.) MFCS 2004. LNCS, vol. 3153, pp. 488–499. Springer, Heidelberg (2004)
12. Festa, P., Pardalos, P.M., Resende, M.G.C.: Feedback set problems. In: Handbook of combinatorial optimization, vol. A (suppl.), pp. 209–258. Kluwer Acad. Publ., Dordrecht (1999)

13. Flum, J., Grohe, M.: Parameterized complexity theory. Theoretical Computer Science. EATCS Series. Springer, Berlin (2006)
14. Fomin, F.V., Gaspers, S., Pyatkin, A.V., Razgon, I.: On the minimum feedback vertex set problem: Exact and enumeration algorithms. Algorithmica (to appear, 2008)
15. Fomin, F.V., Thilikos, D.M.: Dominating sets in planar graphs: branch-width and exponential speed-up. SIAM J. Comput. 36(2), 281–309 (2006) (electronic)
16. Fomin, F.V., Thilikos, D.M.: New upper bounds on the decomposability of planar graphs. Journal of Graph Theory 51(1), 53–81 (2006)
17. Goemans, M.X., Williamson, D.P.: Primal-dual approximation algorithms for feedback problems in planar graphs. In: Cunningham, W.H., Queyranne, M., McCormick, S.T. (eds.) IPCO 1996. LNCS, vol. 1084, pp. 147–161. Springer, Heidelberg (1996)
18. Goemans, M.X., Williamson, D.P.: Primal-dual approximation algorithms for feedback problems in planar graphs. Combinatorica 18(1), 37–59 (1998)
19. Gu, Q.-P., Tamaki, H.: Optimal branch-decomposition of planar graphs in $O(n^3)$ time. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 373–384. Springer, Heidelberg (2005)
20. Kloks, T., Lee, C.M., Liu, J.: New algorithms for k-face cover, k-feedback vertex set, and k-disjoint set on plane and planar graphs. In: Kučera, L. (ed.) WG 2002. LNCS, vol. 2573, pp. 282–295. Springer, Heidelberg (2002)
21. Lin, H.-M., Jou, J.-Y.: On computing the minimum feedback vertex set of a directed graph by contraction operations. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 19(3), 295–307 (2000)
22. Niedermeier, R.: Invitation to fixed-parameter algorithms. Oxford Lecture Series in Mathematics and its Applications, vol. 31. Oxford University Press, Oxford (2006)
23. Robertsonand, N., Seymour, P.D.: Seymour. Graph minors. X. Obstructions to tree-decomposition. J. Combin. Theory Ser. B 52(2), 153–190 (1991)
24. Seymour, P.D., Thomas, R.: Call routing and the ratcatcher. Combinatorica 14(2), 217–241 (1994)

What Is between Chordal and Weakly Chordal Graphs?

Elad Cohen[1], Martin Charles Golumbic[1], Marina Lipshteyn[1], and Michal Stern[1,2]

[1] Caesarea Rothschild Institute, University of Haifa, Israel
[2] The Academic College of Tel-Aviv - Yaffo, Israel

Abstract. An (h, s, t)-representation of a graph G consists of a collection of subtrees $\{S_v | v \in V(G)\}$ of a tree T, such that (i) the maximum degree of T is at most h, (ii) every subtree has maximum degree at most s, and (iii) there is an edge between two vertices in the graph if and only if the corresponding subtrees in T have at least t vertices in common. For example, chordal graphs correspond to $[\infty, \infty, 1] = [3, 3, 1] = [3, 3, 2]$ graphs (notation of ∞ here means that no restriction is imposed).

We investigate the complete bipartite graph $K_{2,n}$ and prove new theorems characterizing those $K_{2,n}$ graphs that have an $(h, s, 2)$-representation and those that have an $(h, s, 3)$-representation.

We characterize $[3, 2, 4]$ graphs as equivalent to the 4-flower-free $[2, 4, 4]$ graphs and give a recognition algorithm for $[2, 3, 4]$ graphs.

Based on these characterizations, we present new results that confirm that weakly chordal graphs, as opposed to chordal graphs, can not be characterized within the $[h, s, t]$ framework. Furthermore, we show a hierarchy of families of graphs between chordal and weakly chordal within the $[h, s, t]$ framework.

1 Introduction and Motivation

A graph is chordal if it contains no chordless cycle of size greater than 3. The chordality of a graph plays a fundamental role in graph theory. The class of chordal graphs is widely investigated. One of the reasons is that the class has a natural intersection model as the intersection graph of subtrees of a tree [2], [4], [13] and hence a concise tree representation. A tree representation can be constructed in linear time, called a clique tree, where each node of the tree corresponds to a maximal clique of the chordal graph.

In many real world applications, the intersection representation of a graph is more important than the graph itself. In [11], [12], the intersection representation of a graph on a tree is generally defined as follows. An (h, s, t)-representation of G consists of a collection of subtrees $\{S_v | v \in V(G)\}$ of a tree T, such that (i) the maximum degree of T is at most h, (ii) every subtree has maximum degree at most s, and (iii) there is an edge between two vertices in G if and only if the corresponding subtrees in T have at least t vertices in common. The class of graphs that have an (h, s, t)-representation is denoted by $[h, s, t]$. We say

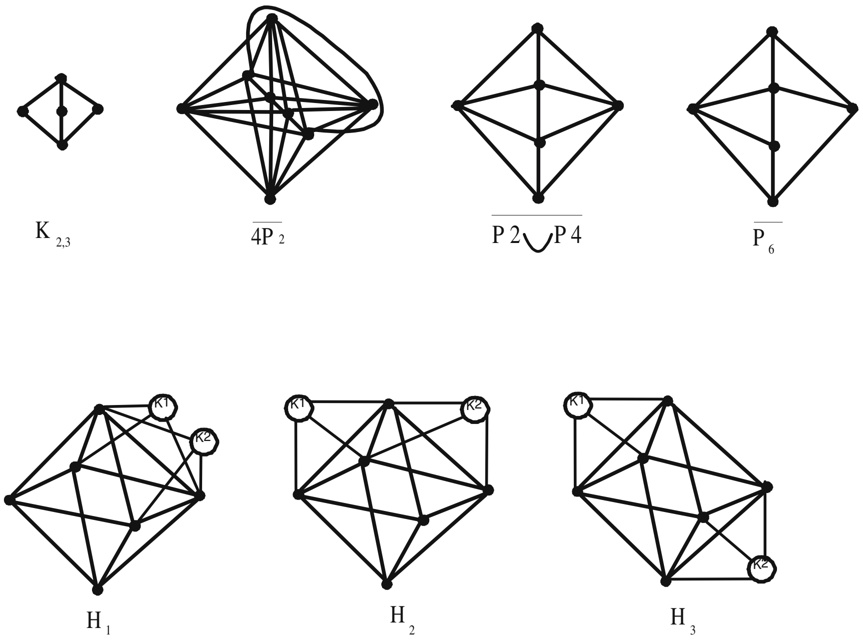


Fig. 1. The forbidden subgraphs

that G is *sharply contained* in $[h, s, t]$ if G is an $[h, s, t]$-graph and G is not an $[h', s', t]$-graph for any $h' < h$ or $s' < s$, which we denote by $G \# \in [h, s, t]$.

Thus, chordal graphs correspond to $[\infty, \infty, 1]$, and was strengthened in [10] and [11], respectively, to be equivalent to $[3, 3, 1]$ and $[3, 3, 2]$. The notation ∞ here means that no restriction is imposed. Interval graphs, by definition, are the $[2, 2, 1]$ graphs. There are other papers that study $[h, s, t]$ graphs, for specific values of h, s and t, although without using this notion. For example, the edge intersection graphs of paths in a tree [6] (EPT graphs) are the $[\infty, 2, 2]$ graphs, and the vertex intersection graphs of paths in a tree [5], [9] (VPT graphs or path graphs) are the $[\infty, 2, 1]$ graphs.

A graph is weakly chordal if neither the graph nor its complement contains a chordless cycle of size greater than 4. The class of weakly chordal graphs is also well studied and has a number of known applications. Our main motivation in this paper was to determine if there is an $[h, s, t]$ class of graphs that corresponds to weakly chordal graphs. By studying the complete bipartite graph, which are a subfamily of weakly chordal graphs, we are able to confirm that weakly chordal graphs can not be characterized within $[h, s, t]$ framework.

In particular, in this paper, we investigate the complete bipartite graph $K_{2,n}$, where one part of the bi-partition has two vertices and the other part has n vertices. In [11], [12], a function $f(h, s, t)$ is given, such that for any $n > f(h, s, t)$, the $K_{2,n}$ graph has no (h, s, t)-representation. We strengthen their results and prove new theorems characterizing those $K_{2,n}$ graphs that have an $(h,s,2)$-representation and those that have an $(h,s,3)$-representations. In [7], the following theorem characterizing $[4,4,2]$ graphs was given.

Theorem 1 ([7]). *A graph G is a weakly chordal ($K_{2,3}$, $\overline{4P_2}$, $\overline{P_2 \cup P_4}$, $\overline{P_6}$, H_1, H_2, H_3)-free graph (see Fig. 1) if and only if the graph G has a $(4,4,2)$-representation.*

In this paper, we characterize the family of $[4,3,2]$ graphs and prove that $[4,3,2]$ graphs are equivalent to 4-flower-free $[4,4,2]$ graphs. We also provide a polynomial time recognition algorithms for the class $[4,3,2]$.

Based on the characterizations of $K_{2,n}$ and $[4,3,2]$ graphs, we present new results that confirm that weakly chordal graphs, as opposed to chordal graphs, can not be characterized within the $[h,s,t]$ framework. Furthermore, we show a hierarchy of families of graphs between chordal and weakly chordal within the $[h,s,t]$ framework. We also show a hierarchy of graphs between chordal and weakly chordal graphs.

The paper is organized as follows. We investigate the (h,s,t)-representations of the $K_{2,n}$ graphs in Sect. 2. In Sect. 3 we give a recognition algorithm for $[4,3,2]$ graphs. In Sect. 4 we deal with the hierarchy of chordal, weakly chordal and $[h,s,t]$ graphs. We also prove that the weakly chordal graphs can not be characterized within $[h,s,t]$ framework.

2 The (h,s,t)-Representations of $K_{2,n}$

Jamison and Mulder [11], [12] have investigated the intersection graph of subtrees of a tree by studying the representations of the complete bipartite graph $K_{2,n}$. They found an upper bound for the size of n as a function of s and t, but as they have mentioned this bound is far from being optimal. We summarize their results as follows:

Let $R(s,t)$ denote the complete balanced rooted tree whose root has s children, internal nodes have $s-1$ children and all leaves are at distance $t-1$ from the root. Let $\gamma(s,t)$ be the number of subtrees $R(s,t)$ which have exactly t nodes and which contain the root.

Jamison and Mulder prove [11] the following:

Theorem 2 ([11]). *Let h,s and t be integers with $h \geq s$, and let n be an integer with $n > \gamma(s,t)(t+1)$, then $K_{2,n}$ is not an $[h,s,t]$ graph.*

Our work started by looking at specific values of the parameter t. In this section, we will improve the bound of Jamison and Mulder for $t=2$ and $t=3$. These results will also be used in section 4 for particular separation examples in the hierarchy shown in Fig. 4.

Theorem 3. $K_{2,s}\# \in [2s,s,2]$.

Corollary 4. *$K_{2,n}$ is an $[h,s,2]$-graph if and only if $h \geq 2s$ and $s \geq n$.*

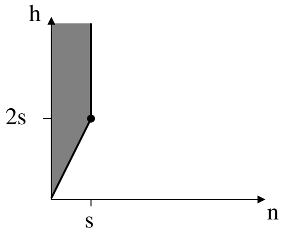


Fig. 2. *The* $K_{2,n}$ representability diagram for $t = 2$

Figure 2 shows the (shaded) area in which $K_{2,n}$ is representable for a fixed s and $t = 2$.

Theorem 5. *$K_{2,f(s)} \# \in [2s-1, s, 3]$, where $f(s) = 2(s-1)^2 + 2(s-1)$.*

Corollary 6. *The graph $K_{2,n}$ is an $[h, s, 3]$ graph if and only if $h \geq 2s-1$ and $s \geq 2(n-1)^2 + 2(n-1)$.*

3 Recognition of [4, 3, 2] Graphs

In this section, we give a characterization and a polynomial time recognition algorithm for the class of $[4, 3, 2]$ graphs. These results are based in part on the previous results of [7] for the class $[4, 4, 2]$.

We start with the following definition of the family of 4-flower graphs:

Definition 7. *Let G_Q be a graph with the vertex set $Q = \{q_{ij} | 1 \leq i < j \leq 4\}$ and edges as follows. We denote Q_i=$\{q_{jk} | j = i$ or $k = i$, $1 \leq j < k \leq 4\}$. Each vertex set Q_i is a clique in G_Q.*

A 4-flower graph consists of:

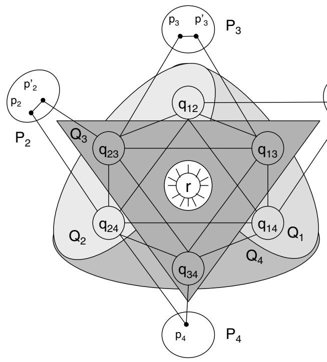
1. *A vertex r.*
2. *An induced subgraph $G_{Q'}$ of G_Q, such that $|Q'_i| \geq 2$, for every $1 \leq i \leq 4$, where Q'_i=$Q_i \cap Q'$. The vertex r is adjacent to all vertices in Q'.*
3. *Four distinct sets of vertices $P_1, \ldots, P_4$, where P_i is either a single vertex that is adjacent to at least two elements of Q'_i or an edge that is adjacent to at least two elements of Q'_i. The vertex r is adjacent to at least one vertex in each P_i.*

Figures 3(a) and 3(b) show examples of 4-flowers. A 4-flower is an example of a graph which is $[4, 4, 2]$, but is not $[4, 3, 2]$ as we will show in Lemma 8.

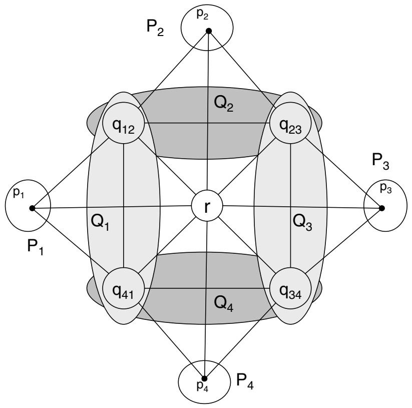
Lemma 8. *4-flower $\# \in [4, 4, 2]$.*

Corollary 9. *The family of $[4, 3, 2]$ graphs is strictly contained in the family of $[4, 4, 2]$ graphs.*

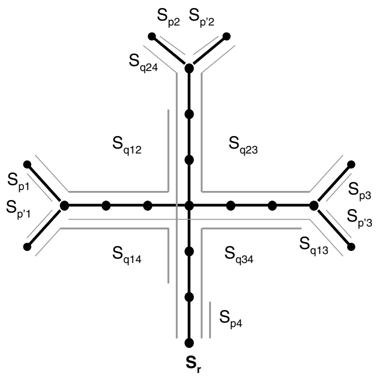
We now present a new recognition algorithm **Recognize** $(4, 3, 2)$**-representation**. The algorithm is robust since it either answers that the input graph is not a $[4, 3, 2]$ graph or finds a certificate: a $(4, 3, 2)$-representation of the input graph.



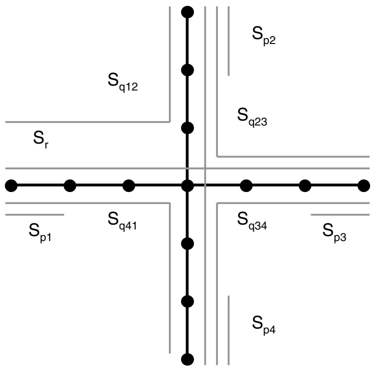
(a) An example of a 4-flower graph.



(b) The minimal 4-flower graph.



(c) The (4,4,2)-representation of a 4-flower graph shown in Figure 3(c).



(d) The (4,4,2)-representation of a minimal 4-flower graph shown in Figure 3(d).

Fig. 3. Examples of a 4-flower graph and their (4,4,2)-representations

Definition 10. *Let $\langle \mathcal{S}, T\rangle$ be an (h, s, t)-representation of G. Let G_U be the induced subgraph of G such that every vertex in G_U corresponds to a subtree in $\langle \mathcal{S}, T\rangle$ that contains the vertex $u \in T$. The collection of subtrees $\mathcal{S}_U$ corresponds to the vertex set of G_U. Every subtree in $\mathcal{S}_U$ is called a core subtree, and the other subtrees in $\langle \mathcal{S}, T\rangle$ are called non-core subtrees. A core edge in T is one that is contained only in core subtrees, and the other edges in T are called non-core edges.*

A two-pair in a graph G is a pair of vertices $\{x, y\}$, such that every chordless path between x and y contains exactly two edges. Clearly, the common neighborhood of a two-pair $\{x, y\}$ is a minimal (x, y)-separator, which we denote by $Sep(x, y)$.

```
Recognize (4,3,2)-representation
input: a graph $G$
output: 'no' (if $G$ is not a $[4,3,2]$ graph) or a $(4,3,2)$-representation $\langle \mathcal{S},T\rangle$ of $G$
1  if G is not weakly chordal (verify using one of the known algorithms, for example [1]) then
       return 'no' ;
2  if G has an induced subgraph isomorphic to one of $\{K_{2,3}, \overline{4P_2}, \overline{P_2 \cup P_4}, \overline{P_6}, H_1, H_2, H_3$, 4-flower$\}$ then
       return 'no';
3  $\langle \mathcal{S},T\rangle \leftarrow$ Construct $(4,4,2)$-representation($G$) (given in [7]);
4  while $\exists$ a vertex $u \in T$, such that $\exists$ subtree $S \in \langle \mathcal{S},T\rangle$ with degree 4 at $u$ do
       $\langle \mathcal{S},T\rangle \leftarrow$ Subtree-degree-reduce4($\langle \mathcal{S},T\rangle$,$u$);

Subtree-degree-reduce4 procedure
input: a $(4,4,2)$-representation $\langle \mathcal{S},T\rangle$ of $G$ and a vertex $u \in T$ at which there is a subtree of degree 4
output: a $(4,4,2)$-representation $\langle \mathcal{S}',T'\rangle$ of $G$ with fewer vertices with a subtree of degree 4, such that $u$ has no subtree with degree 4
5  $\langle \mathcal{S},T\rangle \leftarrow$ PreprocessingA($\langle \mathcal{S},T\rangle$, $u$);
6  find the induced subgraph $G_U$;
7  if $G_U$ is a clique then
       $\langle \mathcal{S}',T'\rangle \leftarrow$ TransformationB($\langle \mathcal{S},T\rangle$, $u$);
       end procedure;
   //$G_U$ is not a clique
8  find a two-pair $\{x,y\}$ in $G_U$ and $Sep(x,y)$;
9  if $Sep(x,y)$ is a clique then
       $\langle \mathcal{S}',T'\rangle \leftarrow$ TransformationC-subtree($\langle \mathcal{S},T\rangle$, $u$, $S_x$);
       end procedure;
   //$Sep(x,y)$ is not a clique
10 find $\{\mathcal{C}_{ij}\}$, $1 \le i < j \le 4$ and $\mathcal{C}_5$ in $G_U$ by Definition 13;
11 MultiColoringD $\leftarrow$MultiColoringD-subtree($\langle \mathcal{S},T\rangle$, $u$, $\{\mathcal{C}_{ij}\}$, $\mathcal{C}_5$);
12 $\langle \mathcal{S}',T'\rangle \leftarrow$ TransformationD-subtree($\langle \mathcal{S},T\rangle$, $u$, MultiColoringD);
```

Theorem 11. [3] *A graph G is weakly chordal if and only if every induced subgraph of G either has a two-pair or is a clique.*

We now present the algorithm. After verifying that G is weakly chordal (Step 1) and contains none of the forbidden subgraphs (Step 2), the algorithm then initializes a $(4,4,2)$-representation (Step 3) using the algorithm of [7]. The heart of our algorithm is the iterative loop (Step 4), refining the representation at each stage by reducing the degree of the subtrees at a vertex u.

The following Lemma 12 is needed to prove the correctness of the algorithm and the characterization Theorem 17.

Lemma 12. *Let $\langle \mathcal{S},T\rangle$ be a $(4,4,2)$-representation of a 4-flower-free graph G. If $\langle \mathcal{S},T\rangle$ is an input to **Subtree-degree-reduce4**($\langle \mathcal{S},T\rangle$, u), then the output is a $(4,4,2)$-representation of G with fewer vertices with a subtree of degree 4,*

```
**PreprocessingA procedure**
**input:** $\langle\mathcal{S},T\rangle$, $u$; **output:** $\langle\mathcal{S},T\rangle$
//let $v_1,\ldots,v_4$ be the neighbors of $u$ in $T$;
1 **foreach** *star edge* $(v_i,u)$ **do**
      find $\mathcal{S}(v_i,u)$, which is the collection of subtrees in $\mathcal{S}$ that contains only the edge
      $(v_i,u)$ among $(v_1,u),\ldots,(v_4,u)$;
      **if** $\mathcal{S}(v_i,u)\neq\emptyset$ **then** split the edge $(v_i,u)$ into two edges, by adding a dummy
      vertex $w$ such that:
      **foreach** *subtree* $S\in\mathcal{S}(v_i,u)$ **do**
          replace $(v_i,u)$ by the edge $(v_i,w)$ in $S$ (thus making $w$ the endpoint of $S$);
      **foreach** *subtree* $S\notin\mathcal{S}(v_i,u)$ *and* $(v_i,u)$ *is contained in* $\mathcal{S}$ **do**
          replace $(v_i,u)$ by the two edges $(v_i,w)$ and $(w,u)$ in $S$.
2 find $\mathcal{S}'\subset\mathcal{S}$ the collection of subtrees such that each contains exactly three edges
  among $(v_1,u),\ldots,(v_4,u)$;
3 **if** $\mathcal{S}'\neq\emptyset$ **then**
      **foreach** *subtree* $S_v\in\mathcal{S}'$ **do**
          add the non-existing edge $(v_i,u)$ to $S_v$;

**TransformationB procedure**
**input:** $\langle\mathcal{S},T\rangle$, $u$; **output:** $\langle\mathcal{S}',T'\rangle$
split the edge $(v_1,u)$ into two edges by adding a dummy vertex $w$;
remove the edge $(v_2,u)$;
add the edge $(v_2,w)$ such that:
**foreach** *subtree* $S\in\mathcal{S}_U$ *that contains the edge* $(v_i,u), i=1,2,$ **do**
    replace $(v_i,u)$ by the two edges $(v_i,w)$ and $(w,u)$ in $S$;

**TransformationC-Subtree procedure**
**input:** $\langle\mathcal{S},T\rangle$, $u$, $S_x$; **output:** $<\mathcal{S}',T'>$
//Suppose $S_x$ contains the star edges $(v_1,u)$ and $(v_2,u)$ ;
split the edge $(v_1,u)$ into two edges by adding a dummy vertex $w$;
replace $(v_1,u)$ by $(v_1,w),(w,u)$ in every subtree containing it;
replace $(v_2,u)$ by $(v_2,w),(w,u)$ in every subtree containing it;

**MultiColoringD-Subtree procedure**
**input:** $\langle\mathcal{S},T\rangle$, $u$, $\{\mathcal{C}_{ij}\}$ for $1\leq i<j\leq 4$, $\mathcal{C}_5$
**output:** *MultiColoringD* ($C(e)$ $\forall e\in T$ and $C(S)$ $\forall$ non-core subtree $S$)
define seven colors $\{c_\lambda\}$, $\lambda=\{12,13,14,23,24,34,5\}$, such that each $c_\lambda$ corresponds
to $\mathcal{C}_\lambda$;
multicolor each edge $e\in T$ with all colors $c_\lambda$, such that $e$ is contained in a core
subtree in $\mathcal{C}_\lambda$;
**repeat**
    **if** *an edge* $e\in T$ *is contained in a non-core subtree* $S$ *with an edge colored* $c_\lambda$
    **then** color the edge $e$ with color $c_\lambda$;
**until** *no further coloring is possible*;
for all edges $e\in T$, $C(e)\leftarrow$ the set of colors of $e$;
for all non-core subtrees $S$, $C(S)\leftarrow\cup\{C(e)|e$ is contained in $S\}$;

**TransformationD-subtree procedure**
**input:** $\langle\mathcal{S},T\rangle$, $u$, *MultiColoringD*; **output:**$\langle\mathcal{S}',T'\rangle$
4 find $1\leq pivot\leq 4$, such that $\mathcal{S}_{v_{pivot}}$ is empty;
5 $\langle\mathcal{S},T\rangle\leftarrow$ **MovePivot**($\langle\mathcal{S},T\rangle$, $u$, *MultiColoringD*, $T_{v_{pivot}}$);
6 **foreach** *subtree* $S_v$, $v\in\mathcal{C}_5$ **do**
      Remove the edge $(u,v_{pivot})$ and its edges in $T_{v_{pivot}}$ from $S_v$;
```

```
MovePivot procedure
input: $\langle \mathcal{S},T \rangle$, $u$, $MultiColoringD$, $T_v$; output: $\langle \mathcal{S},T \rangle$
foreach core edge $e = (v,a) \in T_v$, $c_5 \in C(e)$ do
    $\langle \mathcal{S},T \rangle \leftarrow$ MovePivot($\langle \mathcal{S},T \rangle$, $MultiColoringD$, $T_a$);
while $\exists$ a non-core edge $e = (v,a) \in T_v$, $c_5 \in C(e)$ do
    $\mathcal{E} \leftarrow \{e\} \cup \{e' = (v,a') \in T_v |\ e'$ is a non-core edge and $C(e') = C(e)\}$;
    find star edge $e'' \neq (u, v_{pivot})$ and $C(e'') \supseteq C(e)$;
    split the star edge $e''$ into two edges, by adding a dummy vertex $w$;
    replace the star edge $e''$ by the two edges in every core subtree containing it;
    add a dummy edge $(w', w)$;
    foreach edge $(v,v') \in \mathcal{E}$ do
        replace $(v,v')$ by $(w,w'),(w',v')$ in every core subtree containing it;
        replace $(v,v')$ by $(w',v')$ in every non-core subtree containing it;
        delete the edge $(v,v')$ from $T$;
```

and such that u has no subtree of degree 4. Moreover, no subtree in $\langle \mathcal{S},T \rangle$ has increased its maximal degree in the output representation.

Proof. By Theorem 1, G is a weakly chordal ($K_{2,3}$, $\overline{4P_2}$, $\overline{P_2 \cup P_4}$, $\overline{P_6}$, H_1, H_2, H_3)-free graph. We justify each Step of the procedure with a Claim.

Step 1. The procedure **PreprocessingA**($\langle \mathcal{S},T \rangle$, u) finds a $(4,4,2)$-representation of G, such that every subtree in $\mathcal{S}$ either uses no edges with the endpoint u or uses two edges or four edges with the endpoint u according to Claim 3(i). Moreover, the number of vertices in T contained in a subtree with degree 4 remains the same.

Step 2. We find the induced subgraph G_U, where each vertex in G_U corresponds to a subtree that contains the vertex u in T. By the hereditary property, the induced subgraph G_U is also a weakly chordal graph.

Step 3. According to Theorem 11, the subgraph G_U is either a clique or has a two-pair. If G_U is a clique, then at Step 3 we call **TransformationB($\langle \mathcal{S},T \rangle$, u)** and **Subtree-degree-reduce4** ends. The output is a $(4,4,2)$-representation of G with fewer vertices with subtree of degree 4, such that u has no subtree with degree 4 according to Claim 3(ii). Moreover, no subtree has increased its maximal degree in the output representation.

Otherwise, at Steps 4-8 we assume that G_U is not a clique and therefore has a two-pair.

Step 4. We find a two-pair $\{x,y\}$ and the set $Sep(x,y)$.

Step 5. If $Sep(x,y)$ is a clique, then we perform **TransformationC-subtree** ($\langle \mathcal{S},T \rangle$**,** u**,** S_x), and

Subtree-degree-reduce4 ends. By Claim 3(iii), this finds a $(4,4,2)$-representation of G with fewer vertices with subtree of degree 4, such that u has no subtree of degree 4. Moreover, the maximal degree of each subtree does not increase in the output representation.

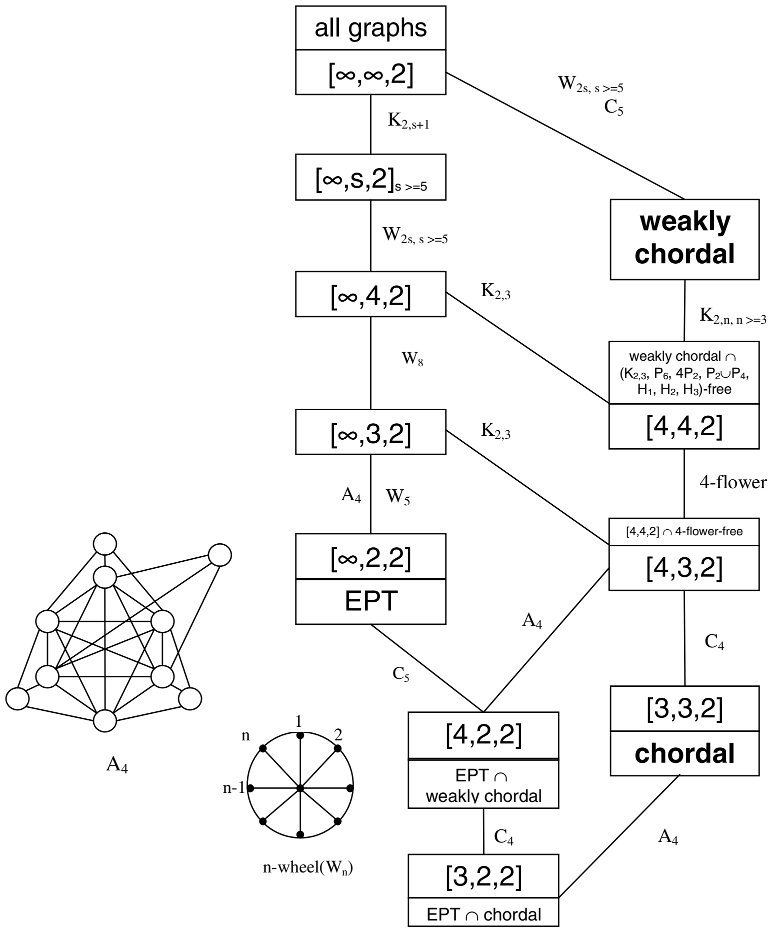


Fig. 4. The hierarchy

Otherwise, at Step 6-8 we assume that $Sep(x, y)$ is not a clique.

Step 6. We find the sets $\mathcal{C}_{ij}$, $1 \leq i < j \leq 4$, and $\mathcal{C}_5$ in G_U by Definition 13.

Step 7. We call **MultiColoringD-subtree**($\langle \mathcal{S},T \rangle$, u, $\mathcal{C}_{ij}$ for $1 \leq i < j \leq 4$, $\mathcal{C}_5$) to obtain *MultiColoringD*, which has the properties proved in Claim 3.

Step 8. We call **TransformationD-subtree**($\langle \mathcal{S},T \rangle$, u, *MultiColoringD*) and according to Claim 3 obtain a $(4, 4, 2)$-representation with fewer vertices with subtree of degree 4, such that u has no subtree with degree 4. Moreover, the maximal degree of each subtree does not increase in the output representation.

□

Claim. Let $\langle \mathcal{S},T \rangle$ be a $(4, 4, 2)$-representation of G and let u be a vertex with neighbors $v_1, \ldots, v_4$ in T:

(i) By performing **PreprocessingA** we obtain a $(4,4,2)$-representation of G, such that every subtree in $\mathcal{S}$ either uses no edges with the endpoint u or uses two or four edges with the endpoint u. Moreover, the number of vertices in T contained in a subtree with degree 4 remains the same.
(ii) If G_U is a clique and there exists a subtree of degree 4 at u, then the output of **TransformationB** is a $(4,4,2)$-representation of G with fewer vertices with subtree of degree 4, such that u has no subtree with degree 4. Moreover, the maximal degree of each subtree does not increase.
(iii) If $Sep(x,y)$ is a clique, then the output of **TransformationC-subtree** is a $(4,4,2)$-representation of G with fewer vertices with subtree of degree 4, such that u has no subtree of degree 4. Moreover, the maximal degree of each subtree does not increase in the output representation.

Definition 13. *Let $v_1,\ldots,v_4$ be the neighbors of $u \in T$. For $1 \leq i < j \leq 4$, we define the cliques $\mathcal{C}_{ij}$=$\{v \in G_U | S_v$ contains exactly two of the star edges (v_i,u) and $(v_j,u)\}$. In addition, $\mathcal{C}_5$=$\{v | v \in G_U - \{\mathcal{C}_{ij}\}, 1 \leq i < j \leq 4\}$. Note that these seven cliques partition $\mathcal{S}_U$ and, by Claim 3, each subtree S_v, $v \in \mathcal{C}_5$, contains $(v_1,u),\ldots,(v_4,u)$.*

Definition 14. *In MultiColoringD, a non-core subtree S touches a color $c \in C(S)$ if S shares an edge with at least one core subtree corresponding to color c. Let T_{v_i} be the subtree rooted at $v_i \in T$, obtained by removing the vertex u from T. Let $\mathcal{S}_{v_i}$, $1 \leq i \leq 4$, be the collection of non-core subtrees S that are contained in T_{v_i}, such that $c_5 \in C(S)$ and $|C(S)| > 2$. For any vertex $a \in T_{v_i}$, we further define the subtree T_a of T_{v_i} to be rooted at a.*

Claim. In the output of **MultiColoringD-subtree** the following hold:

I. Suppose there exists i, such that $\mathcal{S}_{v_i} \neq \emptyset$.
  (i) $\exists S \in \mathcal{S}_{v_i}$ that touches at least two colors.
  (ii) At least one of the following holds:
    (a) $\exists S \in \mathcal{S}_{v_i}$ that touches c_5 and touches at least two other colors.
    (b) $\exists S, S' \in \mathcal{S}_{v_i}$ such that S and S' share an edge in T, and $S \cup S'$ touches c_5 and at least two other colors.
II. $\exists i$, such that $\mathcal{S}_{v_i} = \emptyset$.
III. Let e be a non-core edge, such that $c_5 \in C(e)$ and e is contained in a subtree T_{v_i}, where $\mathcal{S}_{v_i} = \emptyset$. Then there exists a star edge (u,v_j), such that $C(e) \subseteq C((u,v_j))$ where $i \neq j$.

Consider **TransformationD-subtree procedure**. In Step 1, a pivot is chosen according to Claim 3(II). In Step 2, the recursive procedure **MovePivot** is used to modify the representation by identifying and moving a rooted subtree of $T_{v_{pivot}}$ to some other T_{v_j}. In Step 3, we remove edges of $T_{v_{pivot}}$ from each core subtree corresponding to $\mathcal{C}_5$.

Claim. In the output representation $\langle \mathcal{S}, T \rangle$ of **MovePivot** the following hold:
(i) There is no non-core edge e, $\mathcal{C}_5 \in C(e)$ in $T_{v_{pivot}}$.

(ii) Every dummy edge $(w, w') \in T$ is contained in core subtrees with edges colored with c_5 and at most one color of $\{c_{ij}\}$.
(iii) The output representation is a $(4, 4, 2)$-representation of G. Moreover, the maximal degree of each subtree does not increase in the output representation.

Theorem 15. *The hierarchy represented in Fig. 4 is complete.*

Claim. The output of **TransformationD-subtree** is a $(4, 4, 2)$-representation of G with fewer vertices with subtree of degree 4, such that u has no subtree of degree 4. Moreover, the maximal degree of each subtree does not increase in the output representation.

Corollary 16. *Let G be a 4-flower-free* $[4, 4, 2]$ *graph, then* ***Recognize*** $(4, 3, 2)$***-representation*** *algorithm finds a* $(4, 3, 2)$*-representation of G.*

Theorem 17. *G is* $[4, 3, 2]$ *if and only if G is 4-flower-free* $[4, 4, 2]$ *graph.*

Proof. If G is a $[4, 3, 2]$ graph, then it is certainly a $[4, 4, 2]$ graph. Moreover, G does not contain a 4-flower graph by Lemma 8. Conversely, if G is a 4-flower-free $[4, 4, 2]$ graph, then by Corollary 16, G is a $[4, 3, 2]$ graph. □

4 The Hierarchy

In this section, we investigate the relationship between various $[h, s, t]$-graphs and the well-known families of chordal and weakly chordal graphs. Specifically, we demonstrate the results illustrated in the complete hierarchy shown in Fig. 4. We also prove that the class of weakly chordal graphs is incomparable with $[h, s, t]$ graphs for $t > 2$ or $t = 2$ and $h \geq 5$.

We say that a hierarchy is *complete*, when all containment relationships are given. That is, (1) classes that appear in the same box are equivalent, (2) a downward edge from class A to class B indicates that class A contains class B, (3) an example appearing along the edge between two classes is a separating example for those classes, (4) the lack of a hierarchical (containment) relation indicates that the classes are incomparable.

We now state the main result in this section:

Theorem 18. *The class of* $[h, s, t]$ *graphs for any fixed* h, t *is incomparable with the class of weakly chordal graphs if and only if either* $t > 2$ *or* $t = 2$ *and* $h \geq 5$*.*

Proof. ($\Leftarrow$) Let $[h, s, t]$ be a class of graphs with fixed $t \geq 2$. Then by Theorem 2, the graph $K_{2,n}$, $n > \gamma(s, t)(t + 1)$, is not an $[h, s, t]$ graph but is weakly chordal. The graph C_5 is not weakly chordal, but is an $[\infty, 2, 2]$ graph and by containment is an $[h, s, t]$ graph. Therefore, $[h, s, t]$ is incomparable with weakly chordal graphs.

($\Rightarrow$) Let $[h, s, t]$ be a class of graphs which is incomparable with the weakly chordal graphs. Then $t \geq 2$, since the $[h, s, 1]$ graphs are chordal by [4], and therefore are contained in the weakly chordal graphs. Suppose $t = 2$. We consider all possibilities for $h < 5$. The graph classes $[4, 4, 2]$, $[4, 3, 2]$, $[4, 2, 2]$, $[3, 3, 2]$, $[3, 2, 2]$ are weakly chordal by Theorem 15. The graph class $[2, 2, 2]$ are the interval graphs which are weakly chordal. Therefore, $h \geq 5$. □

References

1. Berry, A., Bordat, J., Heggernes, P.: Recognizing weakly triangulated graphs by edge separability. Nordic Journal of Computing 7(3), 164 (Fall, 2000)
2. Buneman, P.: A characterization of rigid circuit graphs. Discrete Math. 9, 205–212 (1974)
3. Hayward, R.B., Hoàng, C.T., Maffray, F.: Optimizing weakly triangulated graphs. Graphs and Combinatorics 5, 339–349 (1989)
4. Gavril, F.: The intersection graphs of subtrees in trees are exactly the chordal graphs. J. Comb. Theory Ser. B, 47–56 (1974)
5. Gavril, F.: A recognition algorithm for the intersection graphs of paths in trees. Discrete Math. 23, 211–227 (1978)
6. Golumbic, M.C., Jamison, R.E.: The edge intersection graphs of paths in a tree. Journal of Combinatorial Theory, Series B 38, 8–22 (1985)
7. Golumbic, M., Lipshteyn, M., Stern, M.: Finding intersection models of weakly chordal graphs. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 241–255. Springer, Heidelberg (2006)
8. Golumbic, M., Lipshteyn, M., Stern, M.: Equivalences and the complete hierarchy of intersection graphs of paths in a tree. Discrete Applied Mathematics (to appear)
9. Golumbic, M., Trenk, A.: Tolerance Graphs, Ch. 11. Cambridge Univ. Press, Cambridge (2004)
10. McMorris, F.R., Scheinerman, E.: Connectivity threshold for random chordal graphs. Graphs and Combin. 7, 177–181 (1991)
11. Jamison, R.E., Mulder, H.M.: Constant tolerance intersection graphs of subtrees of a tree. Discrete Mathematics 290, 27–46 (2005)
12. Jamison, R.E., Mulder, H.M.: Tolerance intersection graphs on binary trees with constant tolerance 3. Discrete Mathematics 215, 115–131 (2000)
13. Walter, J.R.: Representations of rigid cycle graphs, Ph.D. Thesis, Wayne State Univ. (1972)

Parameterized Graph Cleaning Problems*

Dániel Marx and Ildikó Schlotter

Department of Computer Science and Information Theory,
Budapest University of Technology and Economics,
H-1521 Budapest, Hungary
{dmarx,ildi}@cs.bme.hu

Abstract. We investigate the Induced Subgraph Isomorphism problem with non-standard parametrization, where the parameter is the difference $|V(G)| - |V(H)|$ with H and G being the smaller and the larger input graph, respectively. Intuitively, we can interpret this problem as "cleaning" the graph G, regarded as a pattern containing extra vertices indicating errors, in order to obtain the graph H representing the original pattern. We show fixed-parameter tractability of the cases where both H and G are planar and H is 3-connected, or H is a tree and G is arbitrary.

1 Introduction

Problems related to graph isomorphisms play a significant role in algorithmic graph theory. The Induced Subgraph Isomorphism problem is one of the basic problems of this area: given two graphs H and G, find an induced subgraph of G isomorphic to H, if this is possible. In this general form, Induced Subgraph Isomorphism is NP-hard, since it contains several well-known NP-hard problems, such as Independent Set or Longest Induced Path.

As Induced Subgraph Isomorphism has a wide range of important applications, polynomial time algorithms have been given for numerous special cases, such as the case when both input graphs are trees [16] or 2-connected outerplanar graphs [14]. However, Induced Subgraph Isomorphism remains NP-hard even if H is a forest and G is a tree, or if H is a path and G is a cubic planar graph [10]. In many fields where researchers face hard problems, parameterized complexity theory (see e.g. [7] or [9]) has proved to be successful in the analysis and design of algorithms that have a tractable running time in many applications. In parameterized complexity, a parameter k is introduced besides the input I of the problem. A parameterized problem is *fixed-parameter tractable* (FPT) if it admits an algorithm with running time $O(f(k)|I|^c)$ where f is an arbitrary function and c is a constant independent of k.

Note that Induced Subgraph Isomorphism is trivially solvable in time $O(|V(G)|^{|V(H)|}|V(H)|^2)$ on input graphs H and G. As H is typically much

* Research funded by the Hungarian National Research Fund (OTKA grant 67651). The first author is supported by Magyary Zoltán Felsőoktatási Közalapítvány.

smaller than G in applications related to pattern matching, the usual parametrization of Induced Subgraph Isomorphism is to define the parameter to be $|V(H)|$. FPT algorithms are known if G is planar [8], has bounded degree [3], or if H is a log-bounded fragmentation graph and G has bounded treewidth [11].

We consider another parametrization of Induced Subgraph Isomorphism, where the parameter is the difference $|V(G)| - |V(H)|$. Considering the presence of extra vertices as some kind of error or noise, the problem of finding the original graph H in the "dirty" graph G containing errors is clearly meaningful. In other words, the task is to "clean" the graph G containing errors in order to obtain H. For two graph classes $\mathcal{H}$ and $\mathcal{G}$ we define the Cleaning($\mathcal{H}, \mathcal{G}$) problem: given a pair of graphs (H, G) with $H \in \mathcal{H}$ and $G \in \mathcal{G}$, find a set of vertices S in G such that $G - S$ is isomorphic to H. The parameter associated with the input (H, G) is $|V(G)| - |V(H)|$. For the case when $\mathcal{G}$ or $\mathcal{H}$ is the class of all graphs, we will use the notation Cleaning($\mathcal{H}, -$) or Cleaning($-, \mathcal{G}$), respectively.

In the special case when the parameter is 0, the problem is equivalent to the Graph Isomorphism problem, so we cannot hope to give an FPT algorithm for the general problem Cleaning($-, -$). Thus, we consider two special cases. We give FPT algorithms for the problems Cleaning(*Tree*,$-$) and Cleaning(*3-Connected-Planar, Planar*) where *Tree*, *Planar*, and *3-Connected-Planar* denote the class of trees, planar graphs, and 3-connected planar graphs, respectively. Note that these problems differ from the Feedback Vertex Set and the Minimum Apex problems, where the task is to delete a minimum number of vertices from the input graph to get an *arbitrary* acyclic or planar graph, respectively. Both these problems are FPT [2,15].

Without parametrization, Cleaning(*Tree*,$-$) is NP-hard because it contains Longest Induced Path, and we show NP-hardness for Cleaning(*3-Connected-Planar, 3-Connected-Planar*) too. A polynomial time algorithm is known for Cleaning(*Tree, Tree*) [16], and an FPT algorithm is known for Cleaning(*Grid*,$-$) where *Grid* is the class of rectangular grids [5].

2 Notation

We write $[n]$ for $\{1, \ldots, n\}$. The set of the neighbors of $x \in V(G)$ is $N_G(x)$, and for some $X \subseteq V(G)$ we let $N_G(X) = \bigcup_{x \in X} N_G(x)$. The degree of x in G is $d_G(x) = |N_G(x)|$. If $Z \subseteq V(G)$ and G is clear from the context, then we let $N_Z(x) = N_G(x) \cap Z$ and $N_Z(X) = N_G(X) \cap Z$. For some $X \subseteq V(G)$, $G - X$ is obtained from G by deleting X, and $G[X] = G - V(G - X)$. For a subgraph H of G, let $G - H = G - V(H)$. By contracting a vertex of degree 2, we mean deleting it and adding an edge between its neighbors.

A *plane graph* is a planar graph together with a planar embedding. For a subgraph H of a plane graph G, an edge $e \in E(H)$ is called an *outer edge* of (H, G) if G has a face F_e incident to e which is not in H. In this case, F_e is an *outer face* of e w.r.t. (H, G). An *isomorphism* from H into G is a bijection $\varphi : V(H) \cup E(H) \to V(G) \cup E(G)$ preserving incidency. For a subgraph H' of H, $\varphi(H')$ consists of the images of the vertices and edges of H'.

3 The Cleaning(*3-Connected-Planar, Planar*) Problem

In this section, we present an algorithm for Cleaning(*3-Connected-Planar, Planar*). Since 3-connected planar graphs can be considered as "rigid" graphs in the sense that they cannot be embedded in the plane in essentially different ways, this problem seems to be easy. However, Theorem 1 shows that it is NP-hard.

Theorem 1. *Cleaning(3-Connected-Planar, 3-Connected-Planar) is NP-hard.*

Proof. We give a reduction from the NP-complete Planar 3-Colorability problem [10]. Let F be the planar input graph given. W.l.o.g. we assume that F is connected. We construct 3-connected planar graphs H and G such that Cleaning(*3-Connected-Planar, 3-Connected-Planar*) with input (H, G) is solvable if and only if F is 3-colorable.

The gadgets we construct are shown in Fig. 1. For every $x \in V(F)$ we set an integer $9|V(F)| \leq b(x) \leq 10|V(F)|$ such that $b(x) \neq b(y)$ for any $x \neq y \in V(F)$. For every vertex $x \in V(F)$ we build a *node-gadget* N_x in G by taking vertices $a^x, b_1^x, \ldots, b_{6b(x)}^x$ and $c_1^x, \ldots, c_{3b(x)}^x$ and edge set $\{a^x b_j^x, b_j^x b_{j+1}^x \mid j \in [6b(x)]\} \cup \{c_j^x b_{2j-1}^x, c_j^x b_{2j}^x, c_j^x b_{2j+1}^x | j \in [3b(x)]\}$ where $b_{6b(x)+1}^x = b_1^x$. The node-gadget N_x can be considered as a plane graph, supposing that the vertices $b_1^x, b_2^x, \ldots, b_{6b(x)}^x$ (and so $c_1^x, c_2^x, \ldots, c_{3b(x)}^x$) are embedded in a clockwise order around a^x. We define the j-th *block* B_j^x of N_x to be $(c_{3j-2}^x, c_{3j-1}^x, c_{3j}^x)$, for every $j \in [b(x)]$. The *type* of c_j^x can be 0, 1 or 2, according to the value of j modulo 3. We set $C_x = \{c_j^x | j \in [3b(x)]\}$.

For each edge $xy \in E(F)$ we build a *connection* E_{xy} in G that uses 9-9 consecutive blocks from N_x and N_y, say $B_i^x, \ldots, B_{i+8}^x$ and $B_j^y, \ldots, B_{j+8}^y$. These blocks are the *base blocks* for E_{xy}, and we also define $b(x, y) = (i, j)$. Note that since $b(x) \geq 9|V(F)| > 9d_F(x)$, we can define connections such that no block is a base block for different connections. To build E_{xy} with $b(x, y) = (i, j)$, we introduce three new vertices $d_1^{xy}, d_2^{xy}, d_3^{xy}$ and edges $\{c_{3(i+8)-6m+\ell}^x d_m^{xy}, c_{3j-2+6m-\ell}^y d_m^{xy} | m \in [3], \ell \in [6]\} \cup \{c_{3i+6}^x c_{3j+18}^y, c_{3i+2}^x c_{3j+20}^y, c_{3i-2}^x c_{3j+22}^y\}$ (see Fig. 1). By choosing the base blocks for each connection in a way that the order of the connections around a node-gadget is the same as the order of the corresponding edges around the corresponding vertex for some fixed planar embedding of F, we can give a planar embedding of G. Moreover, it is easy to see that G is also 3-connected.

To construct H, we make a disjoint copy $\bar{G}$ of G, and delete some edges and vertices from it as follows. For the copy of c_j^x (a^x, C_x, etc.) we write $\bar{c}_j^x$ ($\bar{a}^x$, $\bar{C}^x$, etc. respectively). To get H, we delete from $\bar{G}$ the three edges connecting vertices of $\bar{C}_x$ and $\bar{C}_y$ for every $x \neq y$, and vertices $\bar{c}_{3j-2}^x$ and $\bar{c}_{3j-1}^x$ for every $x \in V(F), j \in [b(x)]$. Clearly, H is planar, and observe that it remains 3-connected.

Now, we prove that if Cleaning(*3-Connected-Planar, 3-Connected-Planar*) has a solution S for the input (H, G), then F is 3-colorable. Let φ be an isomorphism from H to $G - S$. First, observe that since $b(x) \neq b(y)$ if $x \neq y$, and the integers $\{b(x) | x \in V(F)\}$ are large enough, φ must map $\bar{a}^x$ to a^x because of its degree. For each $x \in V(F)$, the vertices in $C_x \setminus S$ must have the same type, so let the color of x be this type. If $xy \in E(F)$, then the color of x and y must

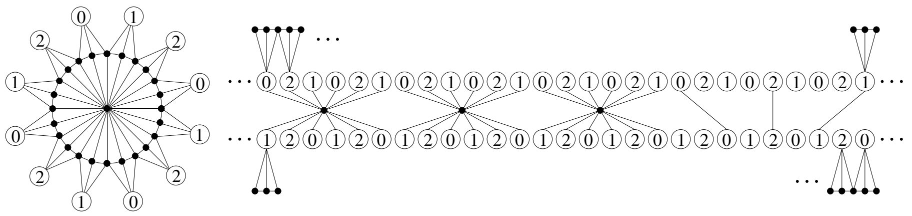


Fig. 1. A node-gadget and a connection for the proof of Theorem 1

differ, otherwise one of the edges $c^x_{3i+6}c^y_{3j+18}, c^x_{3i+2}c^y_{3j+20}, c^x_{3i-2}c^y_{3j+22}$ would be in $G - S$ where $b(x, y) = (i, j)$, as for every type t, one of these edges connects two vertices of type t. Thus the coloring is proper.

For the other direction, let $t : V(F) \to \{1, 2, 3\}$ be a coloring of F. For each $x \in V(F)$, let S contain those vertices in C_x whose type is not $t(x)$ modulo 3. Let φ map $\bar{a}^x$ and $\bar{d}^{xy}_m$ (for every meaningful x, y, m) to a^x and d^{xy}_m, respectively, and let φ map $\bar{c}^x_j$ to $c^x_{j+t(x)-3}$ (in a cyclic order). By adjusting φ on the vertices $\bar{b}^x_i$ in the natural way, we can prove that φ is an isomorphism. It is clear that the restriction of φ on $\bar{N}_x$ is an isomorphism. Note that the only vertex of B^x_j present in $G - S$ is $c^x_{3j+t(x)-3} = \varphi(\bar{c}^x_{3j})$, so independently from $t(x)$ and $t(y)$, the neighborhood of $\bar{d}^{xy}_m$ is also preserved. We only have to check that the edges connecting C_x and C_y are not present in $G - S$. This is implied by the properness of the coloring, as all such edges connect vertices of the same type, but for $xy \in E(F)$ the types of the vertices in $C_x \setminus S$ and $C_y \setminus S$ differ. □

We present an FPT algorithm for Cleaning(*3-Connected-Planar, Planar*) where the parameter is $k = |V(G)| - |V(H)|$ for input (H, G). We assume $n = |V(H)| > k + 2$ as otherwise we can solve the problem by brute force. We also assume that H and G are simple graphs.

Let S be a solution. First observe that if C is a set of at most 2 vertices such that $G - C$ is not connected, then there is a component K of $G - C$ such that $G - S$ is contained in $G[V(K) \cup C]$. Clearly, $|V(K)| \geq n - 2$, and it is unique by $n > k + 2$. Since such a separating set C can be found in linear time [12], K can also be found in linear time. If no component of $G - C$ has size at least $n - 2$, the algorithm outputs 'No', otherwise it proceeds with $G[V(K) \cup C]$ as input.

So we can assume that G is 3-connected. First the algorithm determines a planar embedding of H and G. Every planar embedding determines a circular order of the edges incident to a given vertex. Two embeddings are equivalent, if these orderings are the same for each vertex in both of the embeddings. It is well-known that a 3-connected planar graph has exactly two planar embeddings, and these are reflections of each other (see e.g. [6]). Let us fix an arbitrary embedding θ of H. By the 3-connectivity of G, one of the two possible embeddings of G yields an embedding of $G - S$ that is equivalent to θ. The algorithm checks both possibilities. From now on, we regard H and G as plane graphs, and we are looking for an isomorphism φ from H into $G - S$ which preserves the embedding.

In a general step of the algorithm, we grow a partial mapping, which is a restriction of φ. We assume that φ is already determined on a subgraph D of H having at least one edge, such that the vertices of $H - D$ are embedded in the unbounded face of D. As implied implicitly, $\varphi(V(D)) \cap S = \emptyset$, so if at some point the algorithm would have to delete vertices from $\varphi(D)$, it outputs 'No'.

The algorithm grows the subgraph D on which φ is determined step by step. At each step, it chooses an outer edge e of (D, H), and either deletes some vertices of $G - \varphi(D)$ or adds to D an outer face F of e w.r.t. (D, H). This implies that the outer edges of (D, H) correspond to the outer edges of $(\varphi(D), G)$. Moreover, the algorithm chooses e and F in a way such that after the first step it will always hold that the outer edges of (D, H) form a cycle. We refer to this as choosing an *appropriate* face. This method ensures that every vertex in $V(H) \setminus V(D)$ is embedded in the unique unbounded region determined by the border of D in H. (The *border* of D in H is the subgraph formed by the outer edges of (D, H)). Note that it also follows that the vertices of $V(G) \setminus \varphi(V(D))$ are embedded in the unique unbounded region determined by the border of $\varphi(D)$ in G.

To find an initial partial mapping, we try to find a pair of edges ab and $a'b'$ in H and G, respectively, such that $\varphi(a) = a'$ and $\varphi(b) = b'$. To do that, the algorithm fixes an arbitrary edge ab in H and guesses $\varphi(a)$ and $\varphi(b)$. This yields $2|E(G)|$ possibilities. After this, the algorithm applies one of the following steps.

3-connectivity test. As we can delete vertices from G, it may happen that G ceases to be 3-connected. This can be handled as described above, by finding a separating set C of size at most 2, and determining the component K of $G - C$ with at least $|V(H)| - 2$ vertices. If no such component exists, or if it does not include $\varphi(D)$, then the algorithm outputs 'No', otherwise it deletes $V(G-C-K)$.

Common neighbors test. Let $M = \{\varphi(v) | v \in V(D), d_H(v) < d_G(\varphi(v))\}$. First, note that every vertex in M must have a neighbor in S, thus if $|M| > 2k$, then some vertex in S is adjacent to at least three vertices in M. As the vertices of $S \subseteq V(G) \setminus \varphi(V(D))$ are embedded in the unbounded region determined by the border of $\varphi(D)$ in G, the vertices of M lie on this border. The algorithm checks every vertex q having at least three neighbors on the border of $\varphi(D)$ in G, and determines whether $q \in S$, using Lemma 1. If no such vertex of S can be found in spite of $|M| > 2k$, then the algorithm outputs 'No'.

Lemma 1. *Let q in $V(G) \setminus \varphi(V(D))$ be adjacent to different vertices x, y and z on the border of $\varphi(D)$ in G. Then $q \in S$ if and only if there is no vertex $p \in V(H) \setminus V(D)$ which is a common neighbor of $\varphi^{-1}(x), \varphi^{-1}(y)$ and $\varphi^{-1}(z)$.*

Proof. For contradiction, let us assume $q \in S$ and let a vertex p exist as described. As D is connected and φ preserves the embedding, the outer edges of $(\varphi(D), G)$ and the edges $\varphi(p)x, \varphi(p)y$ and $\varphi(p)z$ cut the plane into four regions, and the only region among these containing all three of x, y and z is the bounded region determined by the outer edges of $(\varphi(D), G)$. But as no vertex in S can be embedded in this region (by our assumption on D), q cannot be adjacent to all of x, y and z, a contradiction. On the other hand, if there is no vertex in $V(H) \setminus V(D)$ adjacent to $\varphi^{-1}(x), \varphi^{-1}(y)$ and $\varphi^{-1}(z)$, then $q \in S$ is trivial. □

Examining an outer face. In this step, the algorithm takes an outer edge $e = xy$ of (D, H) with an appropriate outer face F in H, and the corresponding outer face F' of $\varphi(e)$ w.r.t. $(\varphi(D), G)$. If the algorithm finds that $V(F') \cap S = \emptyset$ must hold because of a sufficient condition given in Lemma 2, then it extends φ by adding F to D. Otherwise, $V(F')$ may contain vertices in S, so the algorithm branches into a bounded number of directions.

In the branch assuming $V(F') \cap S = \emptyset$, the extension of φ is performed. In the branches when $V(F') \cap S \neq \emptyset$ is assumed, the algorithm tries to find and delete the first vertex s on the border of F' in S, and branches according to the choice of s. Lemma 2 bounds the possibilities to choose s.

Lemma 2. *Let $e = t_0 t_f$ be an outer edge of (D, H) and F its outer face w.r.t. (D, H) such that its vertices in clockwise ordering are $t_0, t_1, \dots, t_f$. Similarly, let F' be an outer face of $\varphi(e)$ w.r.t. $(\varphi(D), G)$, where the vertices of F' in clockwise ordering are $t'_0 = \varphi(t_0), t'_1, \dots, t'_{f'-1}$ and $t'_{f'} = \varphi(t_f)$. Let also $R = \{j \mid 0 \leq j \leq \min(f, f'), d_H(t_j) \neq d_G(t'_j)\}$ and let the indices in R be $r_1 < \dots < r_{|R|}$.*

(1) If $|R| \leq 1$ and $f = f'$, then $V(F') \cap S = \emptyset$ and $\varphi(t_i) = t'_i$ for every $i \in [f]$.
(2) If $V(F') \cap S \neq \emptyset$ and t'_{i^*} is the first vertex on the border of F' that is in S, then $i^* - 1 \in \{r_j \mid j \in [\min(|R|, 2k + 1)]\}$.

Proof. Let $e_i = t_i t_{i-1}$ for every $i \in [f]$, so e_{i+1} is followed by e_i in the clockwise circular order of the edges incident to t_i. Now, if e'_{i+1} is followed by $\varphi(e_i)$ in the clockwise circular order of the edges incident to $\varphi(v_i)$ and $e'_{i+1} \in E(G - S)$, then $\varphi(e_{i+1}) = e'_{i+1}$ as φ preserves the embedding. Thus if $V(F') \cap S = \emptyset$, then applying this argument iteratively, from $\varphi(t_0 t_f) = t'_0 t'_{f'}$ we can deduce $\varphi(t_i) = t'_i$ for every $i \in [f]$.

Now, if $V(F') \cap S \neq \emptyset$ and t'_{i^*} is the first vertex on the border of F' that is in S, then the vertices $t'_0, \dots, t'_{i^*-1}$ are not in S, so by applying the above argument we get $\varphi(t_\ell) = t'_\ell$ for all $\ell < i^*$. But t'_{i^*-1} has a neighbor in S, hence $d_G(t'_{i^*-1}) > d_{G-S}(t'_{i^*-1}) = d_{G-S}(\varphi(t_{i^*-1})) = d_H(t_{i^*-1})$. This implies $i^* - 1 \in R$. Letting j^* to be the last vertex on the border of F' that is in S, and using $f = f'$ and the same argument as above, we get $j^* + 1 \in R$. Clearly $i^* - 1 < j^* + 1$, so $V(F') \cap S \neq \emptyset$ would imply $|R| \geq 2$. Hence, the conditions of (1) imply $V(F') \cap S = \emptyset$, proving also $\varphi(t_i) = t'_i$ for every $i \in [f]$.

To prove (2), suppose $V(F') \cap S \neq \emptyset$. As $i^* - 1 \in R$, if ℓ^* is the last index in R such that for any $\ell \leq r_{\ell^*} + 1$ the vertex t'_ℓ is not in S, we get $i^* - 1 = r_{\ell^*+1}$. We claim $\ell^* \leq 2k$, which clearly implies $i^* - 1 \in \{r_j \mid j \in [\min(|R|, 2k + 1)]\}$. To see the claim, suppose $\ell \leq \ell^*$. Since $d_H(t_{r_\ell}) \neq d_G(t'_{r_\ell})$ but $\varphi(t_{r_\ell}) = t'_{r_\ell}$, we get that t'_ℓ is adjacent to a vertex $s \in S$, and by the definition of ℓ^* we know that $s \notin V(F')$, so s is not in the region of G corresponding to the face F of H. Note that in a 3-connected graph no three vertices on the border of a single face can also lie on the border of another face, so no three vertices in $V(F')$ can be adjacent to the same $s \in S$. Using this we obtain $\ell^* \leq 2|S| = 2k$. □

Now let us describe the key mechanism of our algorithm. The essential work is done by a recursive algorithm that we call *GrowSolution*, described in Fig. 2.

```
GrowSolution(H, G, D, φ)

1. If $k = |V(G)| - |V(H)| < 0$ then output('No').
2. Perform the 3-connectivity test.
3. If $D$ equals $H$ then output('Yes').
4. Perform the common neighbors test.
5. Examine an outer face. If for the chosen pair $(F, F')$ of faces $|V(F)| = |V(F')|$
   and $|R| \leq 1$, then extend $\varphi$ on $F$, and go to Step 3. Otherwise branch as follows:
     – for all $j \in [2k+1]$: let $i^* = r_j + 1$ and call $GrowSolution(H, G - t'_{i^*}, D, \varphi)$.
     – if $|V(F)| = |V(F')|$ then extend $\varphi$ on $F$ and call $GrowSolution(H, G, D, \varphi)$.
```

Fig. 2. The algorithm *GrowSolution*

The input of *GrowSolution* is a 4-tuple (H, G, D, φ), where H and G are plane graphs, H is 3-connected, D is a subgraph of H which is either an edge (in the first step) or the union of faces whose border in H is a cycle, and φ is an embedding preserving isomorphism from D to an induced subgraph of G, such that the border of $\varphi(D)$ in G is also a cycle. The algorithm finds out whether there is an $S \subseteq V(G)$ such that φ can be extended to map H to $G - S$ while remaining an isomorphism that preserves embedding. In each call, *GrowSolution* may stop or branch into a few directions. According to this, we will speak of *terminal* and *branching calls.* In each branch of a branching call, *GrowSolution* either deletes a vertex from G, or extends φ by adding a new face to D. If at the end of a branch a vertex is deleted, then this is a *deletion branch*, otherwise it is an *extension branch.* (Actually, the algorithm may extend φ also in the deletion branches before performing the deletion.) At the end of each branch, *GrowSolution* calls itself recursively with the modified input.

In a single call, the algorithm first checks whether $|V(G)| < |V(H)|$, and if so, then correctly outputs 'No'. Next, it handles the case when G is not 3-connected. If D equals H, then Step 3 outputs 'Yes'. Then it searches for common neighbors, as described above. Now, if the algorithm does not stop or delete vertices, it examines an outer face. If for the chosen pair of faces (F, F') the conditions of (1) in Lemma 2 are fulfilled, then we know $V(F') \cap S = \emptyset$, so the algorithm proceeds by extending φ on F according to the lemma. When *GrowSolution* performs this extension, it also adds F to D, and checks whether φ is still an isomorphism on D, and if not, outputs 'No'. This is correct by Lemma 2. This extension step is iterated until either a vertex is deleted or the algorithm stops in Step 3, 4 or 5, or the conditions of (1) in Lemma 2 do not hold.

In the last case, we don't know whether $V(F') \cap S$ is empty or not, so the algorithm branches into at most $2k+2$ directions. First we assume $V(F') \cap S \neq \emptyset$, in this case statement (2) of Lemma 2 implies that $i^* \in \{r_j + 1 \mid j \in [\min(2k+1, |R|)]\}$ where t'_{i^*} is the first vertex on the border of F' being in S. The algorithm branches on these at most $2k+1$ possibilities to delete t'_{i^*}. The last branch is an extension branch corresponding to the case $V(F') \cap S = \emptyset$. Here, *GrowSolution* performs the extension of φ on F as described above. Note that this branch is only necessary if $|V(F)| = |V(F')|$.

Observe that the correctness of the algorithm directly follows from Lemmas 1 and 2. Although *GrowSolution* only answers the decision problem, it is straightforward to modify it in order to output the set S.

To analyze the running time of the algorithm, we assign a search tree $T(I)$ to a run of *GrowSolution* with a given input I. The nodes of this tree correspond to the calls of *GrowSolution*. The leaves represent the terminal calls and the internal nodes represent branching calls. The edge(s) leaving a node represent the branch(es) of the corresponding call of *GrowSolution*, so e heads from x to y if y is called in the branch represented by e in the call corresponding to x. The parameter of a node with input $I = (H, G, D, \varphi)$ is $k_I = |V(G)| - |V(H)|$. The parameter clearly decreases in a deletion branch, which cannot happen more than $k+1$ times. However, in the extension branches this is not true, which seems to make it problematic to bound the size of the search tree. The following lemma shows that this problem does not arise, thanks to Step 4 of the algorithm.

Lemma 3. *The size of $T(I)$ is bounded by a function $f(k)$ where $k = k_I$.*

Proof. Let E^* denote the edges in $T(I)$ that correspond to extension branches. The value of the parameter decreases in each deletion branch, and it can only be negative in a leaf. Thus a path P leading from the root to a leaf in $T(I)$ can include at most $k+1$ edges which are not in E^*. Let $Q = v_0 v_1 \dots v_q$ be a subpath of P containing only edges in E^*.

First, we observe the fact that given a set L of vertices in a simple 3-connected planar graph G and a set $\mathcal{F}$ of faces each having at least 2 vertices from L on their border, we have $|\mathcal{F}| \leq 6|L| - 12$. To see this, we define the planar graph G' such that $V(G') = L$ and for each face $F \in \mathcal{F}$ there is an edge in G' connecting two vertices in $V(F) \cap L$. As G is 3-connected, every edge in G' has multiplicity at most 2, so the planarity of G' yields $|E(G')| \leq 2(3|L| - 6)$. For each face in $\mathcal{F}$ we defined an edge in G', so $|\mathcal{F}| \leq |E(G')| \leq 6|L| - 12$.

For a node w representing a call with input (H, G, D, φ), we define $M(w)$ to be the set containing those vertices $\varphi(t)$ on the border of $\varphi(D)$ in G such that $d_H(t) < d_G(\varphi(t))$. As $|M(v_i)|$ can only decrease after the deletion of some vertices, we get $M(v_{i-1}) \subseteq M(v_i)$ for every $i \in [q]$. Observe that in Step 5 of the branch represented by the edge $v_{i-1}v_i$, a face is added to $\varphi(D)$ that has at least two vertices in $M(v_i) \subseteq M(v_q)$. This follows because the conditions of (1) in Lemma 2 cannot hold in this step, and so the set $R \subseteq M(v_i)$ in Step 5 has cardinality least 2. By Step 4 of the algorithm, $|M(v_q)| \leq 2k$. As shown above, there can be at most $12k - 12$ faces in G that are adjacent to at least 2 vertices in $M(v_q)$, so the number of extensions branches in Q, i.e. the length of Q is at most $12k - 12$. This enables us to bound the length of P, which is at most $k + 1 + (k+1)(12k - 12) < 13k^2$. As every node in $T(I)$ has at most $2k+2$ children, the number of nodes in $T(I)$ is at most $f(k) = (2k+2)^{13k^2}$. □

By careful implementation, it can be assured that the amount of work done when extending φ on a face F is linear in $|V(F)|$, as we only spend constant time at a given vertex. This implies that the consecutive iteration of Steps 3, 4, and 5 can be performed in a total of linear time in $|V(G)|$. As other steps also can

```
3-Connected Planar Cleaning $(H, G)$

1. Perform the 3-connectivity test.
2. Let $H_\theta$ denote an embedded version of $H$, and let $G_{\theta_1}$ and $G_{\theta_2}$ be the two
   possible embedded versions of $G$. For $i = 1, 2$ do:
    3. Let $xy \in E(H)$ be arbitrary. For all $(a, b)$ where $ab \in E(G)$ do:
        4. Let $\varphi_{a,b}$ denote the function mapping $x$ to $a$ and $y$ to $b$.
           Output('Yes') if GrowSolution$(H_\theta, G_{\theta_i}, xy, \varphi_{a,b})$ returns 'Yes'.
5. Output('No').
```

Fig. 3. The algorithm solving Cleaning(*3-Connected-Planar, Planar*)

be performed in time linear in $|V(G)|$, by Lemma 3 we can conclude that the running time of *GrowSolution* on input (H, G, D, φ) is $O(f(k)|V(G)|)$ for some function f, where $k = |V(G)| - |V(H)|$.

As a result, there is an algorithm that solves Cleaning(*3-Connected-Planar, Planar*) in FPT time. The steps of the decision version of this algorithm are described in Fig. 3. Its correctness easily follows from the discussion above. As it calls *GrowSolution* at most $4|E(G)|$ times, we can conclude:

Theorem 2. *The* Cleaning(*3-Connected-Planar, Planar*) *problem on input* (H, G) *can be solved in time* $O(f(k)n^2)$*, where* $n = |V(H)|$ *and* $|V(G)| = n + k$*.*

4 The Cleaning(*Tree*,−) Problem

The aim of this section is to present an FPT algorithm for Cleaning(*Tree*,−). Note that since Cleaning(*Tree*,−) contains the Longest Induced Path problem, the standard parametrization where the parameter is $|V(H)|$ yields a W[2]-hard problem [4].

W.l.o.g. we can assume that G is simple, $n = |V(T)| > k$ (otherwise we can solve the problem by a brute force algorithm) and $e = |E(G)| = O(kn)$ (as we can automatically refuse instances where $e > n - 1 + k(n + k - 1)$). Let S be a fixed solution, i.e. let $G - S = T_S$ be a tree isomorphic to T. Throughout the run of the algorithm, we can assume that G is connected, since by $n > k$ it is trivial to find the unique connected component of G containing T_S.

4.1 Preprocessing

First, we introduce two kinds of reductions, each deleting some vertices from G which must be included in S.

Reduction $\mathcal{A}$: cycles with one common vertex. If for some vertex $x \in V(G)$ there exist cycles $C_1, C_2, \ldots, C_{k+1}$ such that $V(C_i) \cap V(C_j) = \{x\}$ if $i \neq j$, then x must be included in any solution. To see this, observe that if x is not in the solution S, then S must contain at least one vertex from each cycle C_i, but this would imply $|S| \geq k+1$. For each x, we can find such cycles by solving a flow problem in an appropriately defined directed graph. Since we need to find flows with value at most $k+1$, this can be done in time $O(ke)$ for a single vertex x. This means that Reduction $\mathcal{A}$ can be performed in time $O(ken) = O(k^2n^2)$.

Reduction $\mathcal{B}$: disjoint paths between two vertices. Let $x, y \in V(G)$ be vertices such that there exist at least $k+2$ paths from x to y which are disjoint apart from their endpoints. Then x or y must be included in any solution S of size at most k, as assuming $x, y \notin S$ implies the existence of a cycle through x and y in $G-S$. Using standard flow techniques we can check in time $O(ke)$ whether (x,y) is such a pair of vertices, so finding such a pair takes time $O(ken^2) = O(k^2n^3)$. Given such a pair of vertices yields two possibilities for a reduction, so the algorithm branches in two directions. Since $|S| = k$, we can apply Reduction $\mathcal{B}$ at most k times, which means a total of at most 2^k branches.

Now denote by K the minimal connected subgraph of G containing every cycle of G. Note that K is unique, and is an induced subgraph of G. We can construct K from G easily in linear time, as the 2-connected components of a graph can be determined in linear time, e.g. by applying depth first search. Let K_3 denote the vertices of K whose degree in K is at least 3.

Lemma 4. *If Reduction $\mathcal{A}$ and $\mathcal{B}$ cannot be applied, then $d_K(x) \leq k^2+k$ for every $x \in V(K-S)$ and $|K_3| < g(k) = 2k^3(k+1)+3k = O(k^4)$.*

Proof. Let us assume that $x \in V(K-S)$ has neighbors $v_1, v_2, \ldots, v_{k^2+k+1}$ in K. Then the edges xv_i (for $i \in [k^2+k+1]$) can be extended to innerly disjoint paths in K starting from x and ending in a vertex of S. As $|S| \leq k$, there must exist a vertex $s \in S$ such that at least $\lceil (k^2+k+1)/k \rceil = k+2$ of these paths end in s. These paths form at least $k+2$ innerly disjoint paths between x and s, yielding a possibility for Reduction $\mathcal{B}$, a contradiction.

We claim that given a tree T' with maximum degree d and a set $Z \subseteq V(T')$ with cardinality at least $pd+2$, there always exists a set $\mathcal{P}$ of $p+1$ disjoint paths connecting vertices of Z. This is easy to see if we regard T' as a rooted tree and we always choose a new path to put in $\mathcal{P}$ such that its distance from the root is the largest possible. For a vertex $s \in S$, let T_s denote the unique minimal subtree of $K-S$ containing $Z_s = N_{V(K-S)}(s)$. Suppose $|Z_s| \geq k(k^2+k)+2$ for some s. As every vertex in T_s has maximum degree k^2+k by the first claim of the lemma, we get that there are $k+1$ disjoint paths in T_s connecting vertices of Z_s. These paths together with s form $k+1$ cycles whose only common vertex is s, contradicting our assumption that Reduction $\mathcal{A}$ is not applicable.

Thus, we get $|Z_s| \leq k(k^2+k)+1 = k^2(k+1)+1$ for each $s \in S$. Let L denote the leaves of $K-S$. Every vertex in L has a neighbor in S, so $L \subseteq N_{V(K-S)}(S) = \bigcup_{s \in S} Z_s$, implying $|L| \leq |N_{V(K-S)}(S)| \leq k^3(k+1)+k$. Observe that every vertex in $K_3 \setminus (S \cup N_{V(K-S)}(S))$ has degree at least 3 also in $K-S$. Since the number of vertices in the tree $K-S$ having degree at least 3 is less than the number $|L|$ of leaves, we get $|K_3| < |S| + |N_{V(K-S)}(S)| + |L| \leq |S| + 2|L|$, implying $|K_3| < 2k^3(k+1)+3k$. □

4.2 Growing a Mapping

From now on, we assume that Reductions $\mathcal{A}$ and $\mathcal{B}$ cannot be applied. Let ϕ denote the isomorphism from T to T_S that we are looking for. As in Sect. 3, we try to grow a partial mapping from T to T_S, which is always a restriction of ϕ. To begin, the algorithm chooses an arbitrary starting vertex r_0 in T, and branches on the choice of $\phi(r_0)$ in G, which means $|V(G)|$ possibilities.

Assume now that the algorithm has a subtree D of T on which ϕ is already known. The algorithm proceeds step by step, at each step choosing a leaf r of D not yet examined. For the chosen vertex r, it determines ϕ on $N_T(r)$. This means also that it adds $N_T(r)$ to D, deletes $N_G(\phi(r)) \cap S$ from G and checks whether ϕ is still an isomorphism. When determining ϕ on $N_T(r)$, the algorithm may branch into a bounded number of branches, or may proceed with a single branch. Accordingly, we distinguish between *branching* and *simple cases.*

Let us describe the details of a single step of the algorithm. Let $t_1, \dots, t_{n_1}$ denote the neighbors of r in T not in D, and let T_i be the tree component of $T - r$ containing t_i. Similarly, let $t'_1, \dots, t'_{n_2}$ be the neighbors of $r' = \phi(r)$ not in $\phi(D)$ that are connected to r' by edges not in K. Let T'_i denote the component of $G - r'$ that includes t'_i. Observe that either T'_i is a tree, or $r' \notin V(K)$ and T'_i contains K. Finally, let n_3 be the number of vertices in $N_G(r')$ not in $\phi(D)$ that are connected to r' by edges in K. Clearly, $n_1 \leq n_2 + n_3$, and the equality holds if and only if $N_G(r') \cap S = \emptyset$.

First, let us observe that if the tree T_i is isomorphic to T'_j for some i and j, then w.l.o.g. we can assume that $\phi(T_i) = T'_j$. As the trees of a forest can be classified into equivalence classes with respect to isomorphism in time linear in the size of the forest [1,13], this case can be noticed easily. Given two isomorphic trees, an isomorphism between them can also be found in linear time, so the algorithm can extend ϕ on T_i, adding also T_i to the subgraph D. Hence, we only have to deal with the following case: no tree T_i ($i \in [n_1]$) is isomorphic to one of the graphs T'_j ($j \in [n_2]$). This argument makes our situation significantly easier, since every graph T'_j must contain some vertex from S. Therefore $n_2 \leq |S| = k$. By Lemma 4, r' can have degree at most $k^2 + k$ in K, so we get $n_3 \leq k^2 + k$, implying also $n_1 \leq n_2 + n_3 \leq k^2 + 2k$. If these bounds do not hold in some step, then the algorithm outputs 'No'.

The algorithm faces one of the following two cases at each step.

Simple case: $n_2 + n_3 \leq 1$. In this case $n_1 \leq 1$. If $n_2 + n_3 = 0$ then the algorithm proceeds with the next step. Otherwise, let v be the unique vertex in $N_G(r') \setminus V(\phi(D))$. If $n_1 = 0$ then v must be in S, otherwise $\phi(t_1) = v$. According to this, the algorithm deletes v or extends ϕ on t_1, adding also t_1 to D.

Branching case: $n_2 + n_3 \geq 2$. In this case, the algorithm branches on every possible choice of determining ϕ on $N_T(r)$. Guessing $\phi(v)$ for a vertex $v \in N_{V(T-D)}(r)$ can result in at most $n_2 + n_3$ possibilities, so the number of possible branches in a branching step is at most $(n_2 + n_3)^{n_1} \leq (k^2 + 2k)^{k^2+2k}$.

We claim that in a single branch of a run of the algorithm on a solvable input, there can be at most $g(k) + 2k - 2$ branching steps. Observe that $n_3 \geq 2$ implies that r' is either the first vertex in $\phi(D)$ that is in K or $r' \in K_3$, so $n_3 \geq 2$ can happen at most $|K_3| + 1 \leq g(k)$ times, by Lemma 4. If $n_2 \geq 2$, then $G - \phi(D)$ has more connected components containing vertices of S than $G - \phi(D - r)$ has. It is easy to see that this can be true for only at most $|S| - 1$ such vertex r, so this case can happen at most $k - 1$ times. Finally, let S^* denote those vertices of S that are not contained in K. Clearly, if $s \in S^*$, then $|N_{V(T_S)}(s)| \leq 1$. Now, if $n_2 = n_3 = 1$, then $r' \in V(K)$ and the edge $r't'_1$ must be one of the edges that connect to K a tree in $G - K$ containing a vertex in S^*. Observe that there can be at most $|S^*| \leq k - 1$ such edges, thus the claim follows. Therefore, the algorithm only executes at most $g(k) + 2k - 2$ branching steps.

At each vertex the algorithm uses time at most linear in $|V(G)|$. The number of steps performed is at most $|V(T)|$. As both the number of branching cases and the number of branches in a branching case is bounded by a function of k, the algorithm needs quadratic time after choosing $\phi(r_0)$ for the starting vertex r_0. Trying all possibilities on $\phi(r_0)$ enhances this to a cubic time. Reductions $\mathcal{A}$ and $\mathcal{B}$ can also be executed in cubic time, as argued before, so we can conclude:

Theorem 3. *The* Cleaning(*Tree*,–) *problem on input* (T, G) *can be solved in time* $O(f(k)n^3)$*, where* $n = |V(T)|$ *and* $|V(G)| = n + k$.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: The design and analysis of computer algorithms. Addison-Wesley, Reading (1974)
2. Bodlaender, H.: On disjoint cycles. Int. J. Found. Comput. Sci. 5, 59–68 (1994)
3. Cai, L., Chan, S.M., Chan, S.O.: Random separation: a new method for solving fixed-cardinality optimization problems. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 239–250. Springer, Heidelberg (2006)
4. Chen, Y., Flum, J.: On parameterized path and chordless path problems. In: 22nd Annual IEEE Conference on Computational Complexity, pp. 250–263 (2007)
5. Díaz, J., Thilikos, D.M.: Fast FPT-algorithms for cleaning grids. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 361–371. Springer, Heidelberg (2006)
6. Diestel, R.: Graph Theory. Springer, Berlin (2000)
7. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
8. Eppstein, D.: Subgraph isomorphism in planar graphs and related problems. J. Graph Algorithms Appl. 3(3), 1–27 (1999)
9. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability. A Guide to the Theory of NP-Completeness. Freeman, San Francisco (1979)
11. Hajiaghayi, M.T., Nishimura, N.: Subgraph isomorphism, log-bounded fragmentation and graphs of (locally) bounded treewidth. J. Comput. Syst. Sci. 73(5), 755–768 (2007)

12. Hopcroft, J.E., Tarjan, R.E.: Dividing a graph into triconnected components. SIAM J. Computing 2(3), 135–158 (1973)
13. Hopcroft, J.E., Tarjan, R.E.: Efficient planarity testing. J. Assoc. Comput. Mach. 21, 549–568 (1974)
14. Lingas, A.: Subgraph isomorphism for biconnected outerplanar graphs in cubic time. Theoret. Comput. Sci. 63(3), 295–302 (1989)
15. Marx, D., Schlotter, I.: Obtaining a planar graph by vertex deletion. In: Brandstädt, A., Kratsch, D., Müller, H. (eds.) WG 2007. LNCS, vol. 4769, pp. 292–303. Springer, Heidelberg (2007)
16. Matula, D.: Subtree isomorphism in $O(n^{5/2})$. Ann. Discrete Math. 2, 91–106 (1978)

Traffic Grooming in Unidirectional WDM Rings with Bounded Degree Request Graph*

Xavier Muñoz[1] and Ignasi Sau[1,2]

[1] Graph Theory and Combinatorics Group at Applied Mathematics IV Department of UPC - Barcelona, Spain
`xml@ma4.upc.edu`
[2] Mascotte joint Project - INRIA/CNRS-I3S/UNSA - 2004 route des Lucioles - Sophia-Antipolis, France
`Ignasi.Sau@sophia.inria.fr`

Abstract. Traffic grooming is a major issue in optical networks. It refers to grouping low rate signals into higher speed streams, in order to reduce the equipment cost. In SONET WDM networks, this cost is mostly given by the number of electronic terminations, namely Add-Drop Multiplexers (ADMs for short). We consider the unidirectional ring topology with a generic *grooming factor* C, and in this case, in graph-theoretical terms, the traffic grooming problem consists in partitioning the edges of a request graph into subgraphs with at most C edges, while minimizing the total number of vertices of the decomposition.

We consider the case when the request graph has bounded degree Δ, and our aim is to design a network (namely, place the ADMs at each node) being able to support *any* request graph with maximum degree at most Δ. The existing theoretical models in the literature are much more rigid, and do not allow such adaptability. We formalize the problem, and we solve the cases $\Delta = 2$ (for all values of C) and $\Delta = 3$ (except the case $C = 4$). We also provide lower and upper bounds for the general case.

Keywords: optical networks, SONET over WDM, traffic grooming, ADM, graph decomposition, cubic graph, bridgeless graph.

1 Introduction

Traffic grooming is the generic term for packing low rate signals into higher speed streams (see the surveys [3,9,16,18,22]). By using traffic grooming, it is possible to bypass the electronics at the nodes which are not sources or destinations of traffic, and therefore reducing the cost of the network. Typically, in a WDM (Wavelength Division Multiplexing) network, instead of having one SONET Add

* This work has been partially supported by European project IST FET AEOLUS, PACA region of France, Ministerio de Educación y Ciencia of Spain, European Regional Development Fund under project TEC2005-03575, Catalan Research Council under project 2005SGR00256, and COST action 293 GRAAL, and has been done in the context of the CRC CORSO with France Telecom.

Drop Multiplexer (ADM) on every wavelength at every node, it may be possible to have ADMs only for the wavelengths used at that node (the other wavelengths being optically routed without electronic switching).

The so called traffic grooming problem consists in minimizing the total number of ADMs to be used, in order to reduce the overall cost of the network. The problem is easily seen to be NP-complete for an arbitrary set of requests. See [1, 11, 12] for hardness and approximation results of traffic grooming in rings, trees and star networks.

Here we consider unidirectional SONET/WDM ring networks. In that case the routing is unique and we have to assign to each request between two nodes a wavelength and some bandwidth on this wavelength. If the traffic is uniform and if a given wavelength can carry at most C requests, we can assign to each request at most $\frac{1}{C}$ of the bandwidth. C is known as the *grooming ratio* or *grooming factor*. Furthermore if the traffic requirement is symmetric, it can be easily shown (by exchanging wavelengths) that there always exists an optimal solution in which the same wavelength is given to a pair of symmetric requests. Then without loss of generality we will assign to each pair of symmetric requests, called a *circle*, the same wavelength. Then each circle uses $\frac{1}{C}$ of the bandwidth in the whole ring. If the two end-nodes are i and j, we need one ADM at node i and one at node j. The main point is that if two requests have a common end-node, they can share an ADM if they are assigned the same wavelength.

The traffic grooming problem for a unidirectional SONET ring with n nodes and a grooming ratio C has been modeled as a graph partition problem in both [2] and [15] when the request graph is given by a symmetric graph R. To a wavelength λ is associated a subgraph $B_\lambda \subset R$ in which each edge corresponds to a pair of symmetric requests (that is, a circle) and each node to an ADM. The grooming constraint, i.e. the fact that a wavelength can carry at most C requests, corresponds to the fact that the number of edges $|E(B_\lambda)|$ of each subgraph B_λ is at most C. The cost corresponds to the total number of vertices used in the subgraphs involved in the partition, and the objective is therefore to minimize this number.

This problem has been well studied when the network is a unidirectional ring [3, 4, 7, 8, 9, 13, 15, 14, 16, 20, 21]. With the all-to-all set of requests, optimal constructions for a given grooming ratio C were obtained using tools of graph and design theory, in particular for grooming ratio $C = 3, 4, 5, 6$ and $C \geq N(N-1)/6$ [3].

Most of the research efforts in this grooming problem have been devoted to finding the minimum number of ADMs required either for a given set of connection requests (typically a uniform all-to-all communication pattern) or for a general traffic pattern. In all this articles the traffic pattern has been considered as an input for the problem of placing the ADMs. In this paper we consider the traffic grooming problem from a different point of view: assuming a given network topology, it would be desirable to place the minimum number of ADMs as possible at each node in such a way that they could be configured to handle different traffic patterns or graphs of requests. One cannot expect to change the equipment of the network each time the traffic requirements change.

Without any restriction on the graph of requests, the number of required ADMs is given by the worst case, i.e. when the Graph of Requests is the complete graph. However, in many cases some restrictions on the graph of requests might be assumed. From a practical point of view, it is interesting to design a network being able to support any request graph with maximum degree not exceeding a given constant. This situation is usual in real optical networks, since due to technology constraints the number of allowed communications for each node is usually bounded. This flexibility can also be thought from another point of view: if we have a limited number of available ADMs to place at the nodes of the network, then it is interesting to know which is the maximum degree of a request graph that our network is able to support, depending on the grooming factor. Equivalently, given a maximum degree and a number of available ADMs, it is useful to know which values of the grooming factor the network will support.

The aim of this article is to provide a theoretical framework to design such networks with dynamically changing traffic. We study the case when the physical network is given by an unidirectional ring, which is a widely used topology (for instance, SONET rings). In [6] the authors consider this problem from a more practical point of view: they call *t-allowable* a traffic matrix where the number of circuits terminated at each node is at most t, and the objective is also to minimize the number of electronic terminations. They give lower bounds on the number of ADMs and provide some heuristics.

In addition, we also suppose that each pair of communicating nodes establishes a two-way communication. That is, each pair (i, j) of communicating nodes in the ring represents two requests: from i to j, and from j to i. Thus, such a pair uses all the edges of the ring, therefore inducing one unity of load. Hence, we can use the notation introduced in [4] and consider each request as an edge, and then again the grooming constraint, i.e. the fact that a wavelength can carry at most C requests, corresponds to the fact that the number of edges $|E(B_\lambda)|$ of each subgraph B_λ is at most C. The cost corresponds to the total number of vertices used in the subgraphs involved in the partition.

Namely, we consider the problem of placing the minimum number of ADMs in the nodes of the ring in such a way that the network could support *any* request graph with maximum degree bounded by a constant Δ. Note that using this approach, as far as the degree of each node does not exceed Δ, the network can support a wide range of traffic demands without the installation of additional electronic switches at the nodes. The problem can be formally stated as follows: Traffic Grooming in Unidirectional Rings with Bounded-Degree Request Graph

Input: Three integers n, C, and Δ.

Output: An assignment of $A(v)$ ADMs to each node $v \in V(C_n)$, in such a way that *for any undirected request graph R* with maximum degree at most Δ, it exists a partition of $E(R)$ into subgraphs B_λ, $1 \leq \lambda \leq \Lambda$, such that:

(*i*) $|E(B_\lambda)| \leq C$ for all λ; and
(*ii*) each vertex $v \in V(C_n)$ appears in at most $A(v)$ subgraphs.

Objective: Minimize $\sum_{v \in V(C_n)} A(v)$, and the optimum is denoted $A(n, C, \Delta)$.

When the request graph is restricted to belong to a subclass of graphs $\mathcal{C}$ of the class of graphs with maximum degree at most Δ, then the optimum is denoted $A(n, C, \Delta, \mathcal{C})$. Obviously, for any subclass of graph $\mathcal{C}$, $A(n, C, \Delta, \mathcal{C}) \leq A(n, C, \Delta)$.

In this article we solve the cases corresponding to $\Delta = 2$ and $\Delta = 3$ (giving a conjecture for the case $C = 4$), and give lower bounds for the general case. The remainder of the article is structured as follows: in Section 2 we give some properties of the function $A(n, C, \Delta)$, to be used in the following sections. In Section 3 we focus on the case $\Delta = 2$, giving a closed formula for all values of C. In Section 4 we study the case $\Delta = 3$, solving all cases except the case $C = 4$, for which we conjecture the solution. Finally, Section 5 is devoted to conclusions and open problems.

2 Properties of $A(n, C, \Delta)$

In this section we describe some properties of the function $A(n, C, \Delta)$.

Lemma 1. *The following statements hold:*

(i) $A(n, C, 1) = n$.
(ii) $A(n, 1, \Delta) = \Delta n$.
(iii) *If* $C' \geq C$, *then* $A(n, C', \Delta) \leq A(n, C, \Delta)$.
(iv) *If* $\Delta' \geq \Delta$, *then* $A(n, C, \Delta') \geq A(n, C, \Delta)$.
(v) $A(n, C, \Delta) \geq n$ *for all* $\Delta \geq 1$.
(vi) *If* $C \geq \frac{n\Delta}{2}$, $A(n, C, \Delta) = n$.

Proof. *(i)* The request graph can consist in a perfect matching, so any solution uses 1 ADM per node.
(ii) A Δ-regular graph can be partitioned into $\frac{n\Delta}{2}$ disjoint edges.
(iii) Any solution for C is also a solution for C'.
(iv) If $\Delta' \geq \Delta$, the subgraphs with maximum degree at most Δ are a subclass of the class of graphs with maximum degree at most Δ'.
(v) Combine *(i)* and *(iv)*.
(vi) In this case all the edges of the request graph fit into one subgraph.

Since we are interested in the number of ADMs required at each node, let us consider the following definition:

Definition 1. *Let* $M(C, \Delta)$ *be the least positive number* M *such that, for any* $n \geq 1$, *the inequality* $A(n, C, \Delta) \leq Mn$ *holds.*

Lemma 2. $M(C, \Delta)$ *is a natural number.*

Proof. First of all, we know by Lemma 1 that, for any $C \geq 1$, $A(n, C, \Delta) \leq A(n, 1, \Delta) = \Delta n$. Thus $A(n, C, \Delta)$ is upper-bounded by Δn. On the other hand, since any vertex may appear in the request graph, $A(n, C, \Delta)$ is lower-bounded by n. Suppose now that M is not a natural number. That is, suppose that $r < M < r+1$ for some positive integer r. Therefore, there must be at least $(r+1-M)n$ vertices with at most r ADMs each. For each n, let $V_{n,r}$ be the subset of vertices of the request graph with at most r ADMs. Then, since $r+1-M > 0$, we have that $\lim_{n\to\infty} |V_{n,r}| = \infty$. In other words, there is an arbitrarily big subset of vertices with at most r ADMs per vertex. But we can consider a request graph with maximum degree at most Δ on the set of vertices $V_{n,r}$, and this means that r ADMs per node is enough, a contradiction with the optimality of M.

If the request graph is restricted to belong to a subclass of graphs $\mathcal{C}$ of the class of graphs with maximum degree at most Δ, then the corresponding positive integer is denoted $M(C, \Delta, \mathcal{C})$. Again, for any subclass $\mathcal{C}$, $M(C, \Delta, \mathcal{C}) \leq M(C, \Delta)$.
We provide now a lower bound on $M(C, \Delta)$.

Proposition 1 (General Lower Bound). $M(C, \Delta) \geq \left\lceil \frac{C+1}{C} \frac{\Delta}{2} \right\rceil$.

Proof. Erdös and Sachs [10] proved that for any integer k there exist k-regular graphs with arbitrary large girth. For each value of C and Δ, let G be a Δ-regular graph with girth at least $C+1$, and let $n = |V(G)|$. Clearly all the subgraphs (with at most C edges) involved of the partition of the $\Delta n/2$ edges of G are trees. Therefore, the total number of vertices of any partition is at least $\frac{\Delta(C+1)}{2C} n$ (this can be easily seen using that the function $(x+1)/x$ with $1 \leq x \leq C$ is minimized when $x = C$). Then necessarily a vertex must occur in at least $\frac{\Delta(C+1)}{2C}$ subgraphs, yielding the desired bound.

Let a Δ-*graph* be a graph with maximum degree at most Δ. By Lemma 2, $A(n, C, \Delta)$ is of the form $A(n, C, \Delta) = M(C, \Delta)n - \alpha(C, \Delta)$, where $M(C, \Delta)$ and $\alpha(C, \Delta)$ are natural numbers depending only on C and Δ. Since the network must be designed in order to support *any* Δ-graph, if there exists a Δ-graph H that requires strictly more than $M(C, \Delta)$ ADMs at some vertex of the network, then by considering the same graph H on different subsets of vertices of the network, we could force at least $M(C, \Delta) + 1$ ADMs in $\Omega(n)$ nodes of the network, which would be in contradiction with the definition of $M(C, \Delta)$. That is, each vertex can appear in at most $M(C, \Delta)$ subgraphs.

In other words, for each value of C and Δ, the problem reduces to finding the least positive integer M such that the edges of any Δ-graph can be partitioned into subgraphs with at most C edges, in such a way that each vertex appears in at most M subgraphs.
Due to the above discussion, in the sequel we focus on determining the parameter $M(C, \Delta)$ for each value of C and Δ. Observe also that any Δ-graph H is a subgraph of some Δ-regular graph G (with possible more vertices). Note also that if we restrict a partition of G to the vertices of H, the number of occurrences of the vertices cannot increase. Said otherwise, to determine the value of $M(C, \Delta)$ it is enough to consider Δ-regular graphs.

Lemma 3. *The following statements hold trivially:*

(i) $M(C, 1) = 1$ *for all* $C \geq 1$.
(ii) $M(1, \Delta) = \Delta$ *for all* $\Delta \geq 1$.
(iii) *If* $C' \geq C$, *then* $M(C', \Delta) \leq M(C, \Delta)$.
(iv) *If* $\Delta' \geq \Delta$, *then* $M(C, \Delta') \geq M(C, \Delta)$.
(v) $M(C, \Delta) \leq \Delta$ *for all* $C, \Delta \geq 1$.

3 Case $\boldsymbol{\Delta} = 2$

In this case we provide not only the value of $M(C, 2)$, but also the exact expression of the cost function $A(n, C, 2)$.

Proposition 2. $A(n, C, 2) = 2n - (C - 1)$.

Proof. Consider the case when the request graph is 2-regular and has girth greater than C (such a graph exists by [10]). Then, a feasible solution is obtained by placing 2 ADMs at each vertex. We count in how many vertices we can assure that we can place only one ADM.
Let us see first that we cannot use 1 ADM in more than $C - 1$ vertices. Suppose this, i.e. that we have 1 ADM in C vertices and 2 in all the others. Then, consider a set of requests given by a cycle H of length $C + 1$ containing all the C vertices with 1 ADM inside it, and other cycles containing the remaining vertices. In this situation, we are forced to use 2 subgraphs for the vertices of H, and at least 2 vertices of H must appear in both subgraphs. Hence we will need more than 1 ADM in some vertex that had initially only 1 ADM.

Now, let us see that we can always save $C - 1$ ADMs. Let $\{a_0, a_1, \ldots, a_{C-2}\}$ be the set of vertices with only 1 ADM, that we can choose arbitrarily. We will see that we can decompose the set of requests in such a way that the vertices a_i always appear with degree 2 in some subgraph, covering in this way both requests of each vertex with only 1 ADM. Indeed, suppose first that two of these vertices (namely, a_i and a_j) do not appear consecutively in one of the disjoint cycles of the set of requests. Let b_i be the nearest vertex to a_i in the cycle in the direction of a_j, and conversely for b_j (b_i may be equal to b_j if a_i and a_j are separated by exactly one vertex). Then, consider two paths (eventually, cycles) of the form $\{b_i, a_i, \ldots\}$ and $\{b_j, a_j, \ldots\}$, to assure that both a_i and a_j lie in the middle of the subgraph. We do the same construction for each pair of non-consecutive vertices.

Now, consider all the vertices $\{a_0, \ldots, a_i, \ldots, a_{t-1}\}$ which are adjacent in the same cycle of the request graph, with $t \leq C - 1$. Let b_0 be the nearest vertex to a_0 different from a_1, and let b_{t-1} be the nearest vertex to a_{t-1} different from a_{t-2}. Then, consider a subgraph with the path (or cycle, if $b_0 = b_{t-1}$) $\{b_0 a_0 a_1 \ldots a_{t-1} b_{t-1}\}$.

4 Case $\boldsymbol{\Delta} = 3$

We study the cases $C = 3$ and $C \geq 5$ in Sections 4.1 and 4.2, respectively. We discuss the open case $C = 4$ in Section 5.

4.1 Case $C = 3$

We study first the case when the request graph is a bridgeless cubic graph, and then the case of a general request graph.

Bridgeless Cubic Request Graph. We will need some preliminary graph theoretical concepts. Let $G = (V, E)$ be a graph. For $A, B \subseteq V$, an *A-B path* in G is a path from x to y, with $x \in A$ and $y \in B$.

If $A, B \subseteq V$ and $X \subseteq V \cup E$ are such that every A-B path in G contains a vertex or an edge from X, we say that X *separates* the sets A and B in G. More generally we say that X *separates* G if $G - X$ is disconnected, that is, if X separates in G some two vertices that are not in X. A separating set of vertices is a *separator.*

A vertex which separates two other vertices of the same component is a *cut-vertex*, and an edge separating its ends is a *bridge.* Thus, the bridges in a graph are precisely those edges that do not lie on any cycle. A set M of independent edges in a graph $G = (V, E)$ is called a *matching.* A matching that covers all the vertices in V is called *perfect.* A k-regular spanning subgraph is called a k-factor. Thus, a subgraph $H \subseteq G$ is a 1-factor of G if and only if $E(H)$ is a perfect matching of V. Such a matching is also called *perfect matching.*

We recall a well known result from matching theory proved by Petersen in 1891 [17]:

Theorem 1 ([17]). *Every bridgeless cubic graph has a 1-factor.*

If we remove a 1-factor from a cubic graph, what it remains is a disjoint set of cycles.

Corollary 1. *Every bridgeless cubic graph has a decomposition into a 1-factor and disjoint cycles.*

An example of a decomposition of a bridgeless cubic graph into disjoints cycles and a 1-factor is depicted in Fig. 1a.

Proposition 3. *Let $\mathcal{C}$ be the class of cubic graphs with a perfect matching. Then,*

$$M(3, 3, \mathcal{C}) = 2.$$

Proof. Let us proof that we can always partition the request graph into paths with 3 edges in such a way that each vertex appears in 2 paths. To do so, we take the decomposition given by Proposition 1, together with a clockwise orientation of the edges of each cycle. With this orientation, each edge of the 1-factor has two *incoming* and two *outgoing* edges of the cycles. For each edge of the 1-factor we take its two incoming edges, and form in this way a path of length 3. It is easy to verify that this is indeed a decomposition into paths of length three. For instance, if we do this construction in the graph of Fig. 1a, and we label the edges of the 1-factor as {A,B,. . . ,G} and the ones of the cycles as {1,2,. . . ,14} (see Fig. 1b), we obtain the following decomposition:

$$\{1, A, 6\}, \{5, B, 2\}, \{3, C, 8\}, \{7, D, 9\}, \{14, E, 11\}, \{10, F, 12\}, \{4, G, 13\}$$

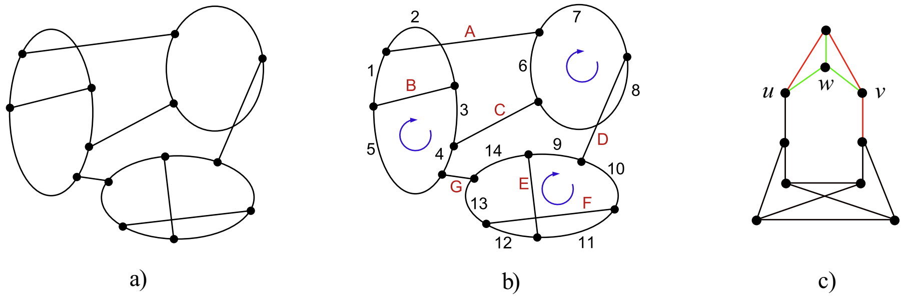


Fig. 1. a) Decomposition of a bridgeless cubic graph into disjoints cycles and a 1-factor. **b)** Decomposition of a bridgeless cubic graph into paths of length 3. **c)** Cubic bridgeless graph used in the proof of Proposition 3.

Now let us see that we cannot do better, i.e. with $2n-1$ ADMs. If such a solution exists, there would be at least one vertex with only 1 ADM, and the average of the number of ADMs of all the other vertices must not exceed 2. In order to see that this is not always possible, consider the cubic bridgeless graph on 10 vertices of Fig. 1c. Let w be the vertex with only 1 ADM. This graph has no triangles except those containing w. Since we can use only 1 ADM in w, we must take all its requests in one subgraph. It is not possible to cover the 4 remaining requests of the nodes u and v in one subgraph, and thus without loss of generality we will need 3 ADMs in u. With these constraints, one can check that the best solution uses 20 ADMs, that is $2n > 2n-1$.

General Request Graph. It turns out that when the request graph is not restricted to be bridgeless we have that $M(3,3) = 3$.

Proposition 4. $M(3,3) = 3$.

Proof. By *(ii)* and *(iii)* of Lemma 3 we know that $M(3,3) \leq 3$. We shall exhibit a counterexample showing that $M(3,3) > 2$, proving the result. Consider the cubic graph G depicted in Fig. 2a. We will prove that it is not possible to partition the edges of G into subgraphs with at most 3 edges in such a way that each vertex appears in at most 2 subgraphs.

Indeed, suppose the opposite, i.e. that we can partition the edges of G into subgraphs $B_1, \ldots, B_k$ with $|E(B_i)| \leq 3$ in such a way that each vertex appears in at most 2 subgraphs, and let us reach a contradiction.

Following the notation illustrated in Fig. 2a, let A_1, A_2, A_3 be the connected components of $G \setminus \{e_1, e_2, e_3\}$. Let also, with abuse of notation, $a_i = A_i \cap e_i$, $i = 1, 2, 3$, and $a_0 = e_1 \cap e_2 \cap e_3$.

Claim. There exist an index $i^* \in \{1, 2, 3\}$ and a subgraph B_{k^*} containing a_0, such that $B_{k^*} \cap A_{i^*} = \{a_{i*}\}$.

Proof. Among all the subgraphs $B_1, \ldots, B_k$ involved in the decomposition of G, consider the ℓ subgraphs $B_{j_1}, \ldots, B_{j_\ell}$ covering the edges $\{e_1, e_2, e_3\}$. If $\ell = 1$, then the subgraph B_{j_1} is a star with three edges and center a_0, and then

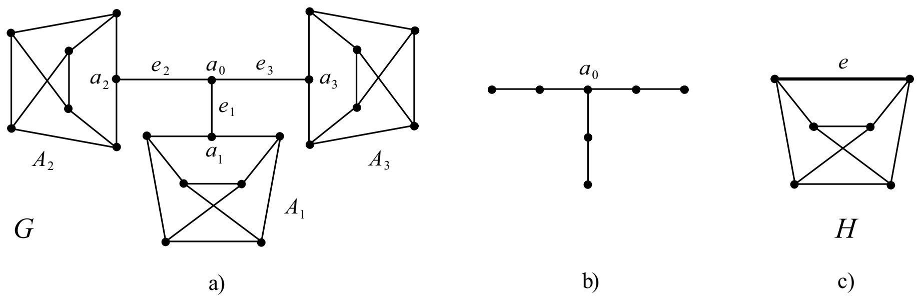


Fig. 2. a) Cubic graph G that can not be edge-partitioned into subgraphs with at most 3 edges in such a way that each vertex appears in at most 2 subgraphs. **b)** Graph that cannot be partitioned into 2 connected subgraphs with at most 3 edges. **c)** Counterexample of Proposition 4 showing that $M(3,3) = 3$.

$B_{j_1} \cap A_i = \{a_i\}$ for each $i = 1, 2, 3$. If $\ell \geq 3$, then the vertex a_0 appears in 3 subgraphs, a contradiction. Hence it remains to handle the case $\ell = 2$. If the claim was not true, it would imply that for each $i = 1, 2, 3$ it would exist $j_{f(i)} \in \{j_1, j_2\}$ such that $B_{f(i)} \cap A_i$ contains at least one edge. In particular, this would imply that the graph depicted in Fig. 2b could be partitioned into two connected subgraphs with at most three edges, which is clearly not possible.

Suppose without loss of generality that the index i^* given by Claim 4.1 is equal to 1. Thus, a_1 appears in a subgraph B_{k^*} that does not contain any edge of A_1. Therefore, the edges of A_1 must be partitioned into connected subgraphs with at most 3 edges, in such a way that a_1 appears in only 1 subgraph, and all its other vertices in at most 2 subgraphs, each. Let us now see that this is not possible, obtaining the contradiction we are looking for.

Indeed, since a_1 has degree 2 in A_1 and it can appear in only one subgraph, it must have degree two in the subgraph in which it appears, i.e. in the middle of a P_3 or a P_4, because A_1 is triangle-free. It is easy to see that this is equivalent to partitioning the edges of the graph H depicted in Fig. 2c into subgraphs with at most 3 edges, in such a way that the thick edge e appears in a subgraph with at most 2 edges, and each vertex appears in at most 2 subgraphs. Observe that H is cubic and triangle-free. Let n_1 be the total number of vertices of degree 1 in all the subgraphs of the decomposition of H. Since each vertex of H can appear in at most 2 subgraphs and H is cubic, each vertex can appear with degree 1 in at most 1 subgraph. Thus, $n_1 \leq |V(H)| = 6$.

Since we have to use at least 1 subgraph with at most 2 edges and $|E(H)| = 9$, there are at least $1 + \lceil \frac{9-2}{3} \rceil = 4$ subgraphs in the decomposition of H. But each subgraph involved in the decomposition of H has at least 2 vertices of degree 1, because H is triangle-free. Therefore, $n_1 \geq 8$, a contradiction.

4.2 Case $C \geq 5$

For $C \geq 5$ we can easily prove that $M(C, 3) = 2$, making use of a conjecture made by Bermond *et al.* in 1984 [5] and proved by Thomassen in 1999 [19]:

Theorem 2 ([19]). *The edges of a cubic graph can be 2-colored such that each monochromatic component is a path of length at most 5.*

A *linear k-forest* is a forest consisting of paths of length at most k. The *linear k-arboricity* of a graph G is the minimum number of linear k-forests required to partition $E(G)$, and is denoted by $la_k(G)$ [5]. Theorem 2 is equivalent to saying that, if G is cubic, then $la_5(G) = 2$.

Let us now see that Theorem 2 implies that $M(C,3) = 2$ for all $C \geq 5$. Indeed, all the paths of the linear forests have at most 5 edges, and each vertex will appear in exactly 2 linear 5-forests, so the decomposition given by Theorem [19] is a partition of the edges of a cubic graph into subgraphs with at most 5 edges, in such a way that each vertex appears in at most 2 subgraphs. In fact the result of [19] is stronger, in the sense that G can be any graph of maximum degree at most 3. Thus, we deduce that

Corollary 2. *For any $C \geq 5$, $M(C,3) = 2$.*

Thomassen also proved [19] that 5 cannot be replaced by 4 in Theorem 2. This fact do not imply that $M(4,3) = 3$, because of the following reasons: (i) the subgraphs of the decomposition of the request graph are not restricted to be paths, and (ii) it is not necessary to be able to find a 2-coloring of the subgraphs of the decomposition (a *coloring* in this context means that each subgraph receives a color, and 2 subgraphs with the same color must have empty intersection).

5 Conclusions

We considered the traffic grooming problem in unidirectional WDM rings when the request graph belongs to the class of graph with maximum degree Δ. This formulation allows the network to support dynamic traffic without reconfiguring the electronic equipment at the nodes. We formally defined the problem, and we focused mainly on the cases $\Delta = 2$ and $\Delta = 3$, solving completely the former and solving all the cases of the latter, except the case when the grooming value C equals 4. We proved in Section 4.1 that $M(3,3) = 3$, and in Section 4.2 that $M(C,3) = 2$ for all $C \geq 5$. Proposition 3 states that $M(4,3,\mathcal{C}) = 2$, $\mathcal{C}$ being the

Table 1. Values of $M(C,\Delta)$. The case $C = 4, \Delta = 3$ is a conjectured value.

$C \mid \Delta$	1	2	3	4	5	6	...	Δ
1	1	2	3	4	5	6	...	Δ
2	1	2	3	≥ 3	≥ 4	≥ 5	...	$\geq \lceil \frac{3\Delta}{4} \rceil$
3	1	2	3	≥ 3	≥ 4	≥ 4	...	$\geq \lceil \frac{2\Delta}{3} \rceil$
4	1	2	2?	≥ 3	≥ 4	≥ 4	...	$\geq \lceil \frac{5\Delta}{8} \rceil$
5	1	2	2	≥ 3	≥ 3	≥ 4	...	$\geq \lceil \frac{3\Delta}{5} \rceil$
...	...	...	...	...	...	...	...	...
C	1	2	2	≥ 3	$\geq \lceil \frac{C+1}{C} \frac{5}{2} \rceil$	≥ 4	...	$\geq \lceil \frac{C+1}{C} \frac{\Delta}{2} \rceil$

class of cubic graph with a perfect matching. Because of the integrality of $M(C, \Delta)$ and Lemma 3, $M(4, 3)$ equals either 2 or 3. We conjecture that $M(4, 3) = 2$.

We also deduced lower and upper bounds in the general case (any value of C and Δ). Table 1 summarizes the values of $M(C, \Delta)$ that we obtained.

This problem can find wide applications in the design of optical networks using WDM technology. It would be interesting to continue the study for larger values of Δ, which will certainly rely on graph decomposition results. Another generalization could be to restrict the request graph to belong to other classes of graphs for which there exist powerful decomposition tools, like graphs with bounded tree-width, or families of graphs excluding a fixed graph as a minor.

Acknowledgement. The authors want to thank the anonymous referees for very helpful remarks that have substantially improved the paper.

References

1. Amini, O., Pérennes, S., Sau, I.: Hardness and Approximation of Traffic Grooming. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 561–573. Springer, Heidelberg (2007)
2. Bermond, J.-C., Coudert, D.: Traffic grooming in unidirectional WDM ring networks using design theory. In: IEEE ICC, Anchorage, Alaska, vol. 2, pp. 1402–1406 (May 2003)
3. Bermond, J.-C., Coudert, D.: The CRC Handbook of Combinatorial Designs, 2nd edition. In: Colbourn, C.J., Dinitz, J.H. (eds.) Discrete Mathematics and Its Applications, ch. VI.27, Grooming, 2nd edn. Discrete Mathematics and Its Applications, vol. 42, pp. 493–496. CRC Press, Boca Raton (2006)
4. Bermond, J.-C., Coudert, D., Muñoz, X.: Traffic grooming in unidirectional WDM ring networks: the all-to-all unitary case. In: The 7th IFIP Working Conference on Optical Network Design & Modelling – ONDM 2003, pp. 1135–1153 (February 2003)
5. Bermond, J.-C., Fouquet, J.-L., Habib, M., Péroche, B.: On linear k-arboricity. Discrete Math. 52(2-3), 123–132 (1984)
6. Berry, R., Modiano, E.: Reducing electronic multiplexing costs in SONET/WDM rings with dynamically changing traffic. IEEE J. on Selected Areas in Comm. 18, 1961–1971 (2000)
7. Chiu, A.L., Modiano, E.H.: Traffic grooming algorithms for reducing electronic multiplexing costs in WDM ring networks. IEEE/OSA Journal of Lightwave Technology 18(1), 2–12 (2000)
8. Dutta, R., Rouskas, N.: On optimal traffic grooming in WDM rings. IEEE Journal of Selected Areas in Communications 20(1), 1–12 (2002)
9. Dutta, R., Rouskas, N.: Traffic grooming in WDM networks: Past and future. IEEE Network 16(6), 46–56 (2002)
10. Erdös, P., Sachs, H.: Reguläre graphe gegebener taillenweite mit minimaler knotenzahl. Wiss. Z. Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe 12, 251–257 (1963)
11. Flammini, M., Monaco, G., Moscardelli, L., Shalom, M., Zaks, S.: Approximating the traffic grooming problem in tree and star networks. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 147–158. Springer, Heidelberg (2006)

12. Flammini, M., Moscardelli, L., Shalom, M., Zaks, S.: Approximating the traffic grooming problem. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 915–924. Springer, Heidelberg (2005)
13. Gerstel, O., Lin, P., Sasaki, G.: Wavelength assignment in a WDM ring to minimize cost of embedded SONET rings. In: IEEE Infocom, San Francisco, California, pp. 94–101 (1998)
14. Gerstel, O., Ramaswani, R., Sasaki, G.: Cost-effective traffic grooming in WDM rings. IEEE/ACM Transactions on Networking 8(5), 618–630 (2000)
15. Goldschmidt, O., Hochbaum, D., Levin, A., Olinick, E.: The SONET edge-partition problem. Networks 41(1), 13–23 (2003)
16. Modiano, E., Lin, P.: Traffic grooming in WDM networks. IEEE Communications Magazine 39(7), 124–129 (2001)
17. Petersen, J.P.: Die Theorie der Regulären Graphs (The Theory of Regular Graphs). Acta Mathematica 15, 193–220 (1891)
18. Somani, A.: Survivable traffic grooming in WDM networks. In: Gautam, D.K. (ed.) Broad band optical fiber communications technology – BBOFCT, Jalgaon, India, December 2001, pp. 17–45. Nirtali Prakashan (2001) (Invited paper)
19. Thomassen, C.: Two-coloring the edges of a cubic graph such that each monochromatic component is a path of length at most 5. J. Comb. Theory Ser. B 75(1), 100–109 (1999)
20. Wan, P.-J., Calinescu, G., Liu, L., Frieder, O.: Grooming of arbitrary traffic in SONET/WDM BLSRs. IEEE Journal of Selected Areas in Communications 18(10), 1995–2003 (2000)
21. Wang, J., Cho, W., Vemuri, V., Mukherjee, B.: Improved approches for cost-effective traffic grooming in WDM ring networks: Ilp formulations and single-hop and multihop connections. IEEE/OSA Journal of Lightwave Technology 19(11), 1645–1653 (2001)
22. Zhu, K., Mukherjee, B.: A review of traffic grooming in WDM optical networks: Architectures and challenges. Optical Networks Magazine 4(2), 55–64 (2003)

Fast Robber in Planar Graphs[⋆]

Nicolas Nisse and Karol Suchan

DIM, Universidad de Chile, Santiago, Chile
karol@suchan.info, nisse.nicolas@gmail.com

Abstract. In the *cops and robber game*, two players play alternately by moving their tokens along the edges of a graph. The first one plays with the *cops* and the second one with one *robber*. The cops aim at capturing the robber, while the robber tries to infinitely evade the cops. The main problem consists in minimizing the number of cops used to capture the robber in a graph. This minimum number is called the *cop-number* of the graph. If the cops and the robber have the same velocity, $3 + \frac{3}{2}g$ cops are sufficient to capture one robber in any graph with genus g (Schröder, 2001). In the particular case of a grid, 2 cops are sufficient.

We investigate the game in which the robber is slightly faster than the cops. In this setting, we prove that the cop-number of planar graphs becomes unbounded. More precisely, we prove that $\Omega(\sqrt{\log n})$ cops are necessary to capture a fast robber in the $n \times n$ square-grid. This proof consists in designing an elegant evasion-strategy for the robber. Then, it is interesting to ask whether a high value of the cop-number of a planar graph H is related to a large grid G somehow contained in H. We prove that it is not the case when the notion of containment is related to the classical transformations of edge removal, vertex removal, and edge contraction. For instance, we prove that there are graphs with cop-number at most 2 and that are subdivisions of arbitrary large grids. On the positive side, we prove that, if a planar graph H planar a large grid as an induced subgraph, then H has large cop-number. Note that, generally, the cop-number of a graph H is not closed by taking induced subgraphs G, even if H is planar and G is an distance-hereditary induced-subgraph.

Keywords: Cops and robber, planar graph, minor, subdivision, grid.

1 Introduction

Introduced by Nowakowsky and Winkler [NW83], and by Quilliot [Qui83], cops and robber game is a two player game in a graph G (see [Als04] for a survey). The first player, $\mathcal{C}$, plays with the *cops* that are aiming at capturing the *robber*, played by the second player $\mathcal{R}$. First, $\mathcal{C}$ chooses a subset of vertices of G and places his cops on these vertices. Then, $\mathcal{R}$ places his robber on some vertex of G. Then, $\mathcal{C}$ and $\mathcal{R}$ play alternately. At each step, $\mathcal{C}$ chooses a subset of his cops and move each of them along some path of length at most $v_{cop} \geq 1$ edges. Then, $\mathcal{R}$ moves his robber along some path of length at most $v_{robber} \geq 1$ edges. Note that

⋆ Authors acknowledge the support of CONICYT via Anillo en Redes ACT08.

both players have perfect knowledge of the position(s) of their adversary. The robber is caught as soon as it occupies the same vertex as a cop. The purpose of $\mathcal{C}$ is to capture the robber, while $\mathcal{R}$ tries to infinitely avoid being caught. In the following, we refer to v_{cop} and v_{robber} as the cops' speed and the robber's speed, respectively. For any graph G, the (p,q)-cop-number, denoted $c_{p,q}(G)$, is the smallest number of cops with speed p sufficient to capture any robber with speed q in G.

The case $p = q = 1$ has received particular attention in the literature, and $c_{1,1}(G)$ is generally called the *cop-number* of the graph G. The main result in [NW83, Qui83] is a characterization of the graphs with cop-number one, called *cop-win* graphs. In particular, trees, chordal graphs and, more generally, graphs with chordality at most 4 are cop-win [Far87, AF88, Che97]. This characterization also allowed Hahn and MacGillivray [HM06] to design an algorithm deciding in time $O(n^k)$ if the cop-number of a n-node graph is at most $k \geq 1$ (see also [BI93]). Goldstein and Reingold [GR95] prove that the problem of computing the cop-number of a directed graph (in this setting, the cops and the robber are constrained to follow the orientation of the arcs) is EXPTIME-complete. This problem is EXPTIME-complete as well in undirected graphs when cops and robber are given their initial positions [GR95].

From a combinatorial point of view, the cop-number of bounded genus graphs has been widely studied. In [AF84], Aigner and Fromme proved that the cop-number of any planar graph is at most three. In particular, the cop-number of any grid is two. The result of Aigner and Fromme is based on the simple following Proposition 1.

Proposition 1. *[AF84] In any graph G and for any shortest path P of G, after a finite number of steps, a single cop can prevent the robber with speed one from entering P.*

Then, Aigner and Fromme [AF84] prove that it is possible to recursively divide any planar graph using three shortest paths chosen in such a way that the area accessible to the robber only decreases. Using the same kind of techniques, Quilliot [Qui85] proves that the cop-number of any graph with genus $g \geq 0$ is at most $3 + 2g$. Schröder [Sch01] improves this bound to $3 + \frac{3}{2}g$. Proposition 1 is also essential in the proof of the fact that the cop-number of any H-minor-free graph is upper-bounded by the number of edges of H [And86].

It is noticeable that very few lower bounds on the cop-number of graphs have been proved. Aigner and Fromme [AF84] prove that the cop-number of any graph with girth at least 5 is lower-bounded by its minimum degree. Frankl [Fra87] improves this bound to d^t for any graph with girth at least $8t-3$ and minimum degree $d+1$. Note also that, for any $k \geq 3$ and $n \geq 1$, it exists a k-regular graph with cop-number at least n [And84].

We investigate the cops and robber game in planar graphs when the robber is slightly faster than the cops, i.e., $v_{robber} > v_{cop}$. It is easy to be convinced that Proposition 1 becomes false as soon as the robber is faster than the cops. In particular, we prove that allowing the robber to be faster than the cops may drastically increase the number of cops necessary to capture it in a square-grid.

We then generalize this result to a larger class of planar graphs. More precisely, we propose a certificate that is sufficient to decide that "many" cops are necessary to capture a fast robber. We leave as an open question, if our certificate is also a necessary condition.

For ease of description, in this paper, we consider that the robber's speed is two while the cops' speed is one, and we refer to $c_{1,2}(G)$ as the *cop-number* of the graph G. However our results can easily be generalized for any $v_{robber}/v_{cop} > 1$.

1.1 Our Results

Our main result consists in proving that the cop-number of square-grids is not bounded. We prove that the cop-number of a $n \times n$ square-grid is at least $\Omega(\sqrt{\log n})$. The proof is constructive since we give a simple and explicit evasion-strategy for the robber. More precisely, we prove that, for any $k \geq 1$, there are two constants $a > 0$ and $b > 2$, such that, one robber with speed 2 can infinitely evade k cops with speed one in any $n \times n$ square-grid with $n \geq 4\ a^k b^{k(k+1)/2} = f(k)$.

A natural question is then to ask whether this lower bound still holds for planar graphs somehow containing a large grid. In other words, is a high value of the cop-number of a planar graph H related to a large grid G somehow contained in H? On the negative side, the classical transformations of edge removal, vertex removal, and edge contraction do not preserve "small" cop-number. For instance, for any $k \geq 1$, we design a subdivision H of a $n \times n$ square-grid with $n \geq f(k)$, such that the cop-number of H is at most 2. The converse also holds: we prove that the cop-number of a planar graph may drastically decrease by contracting edges incident to degree-2 vertices. This confirms the intuition according to which the cop-number of a graph is more related to the distances rather than to the connectivity of the graph.

On the other hand, we prove that if a planar graph H contains a large grid as an induced subgraph, then H has large cop-number. More precisely, any planar graph H that contains a $n \times n$ square-grid G with $n \geq 2f(k)$ as an induced subgraph has cop-number at least k. Note that this latter result is not trivial because the cop-number of a graph is generally not closed by taking induced subgraphs, even if H is planar and G is a distance-hereditary induced-subgraph. Indeed, consider the cycle C_4, and let H be the graph obtained from C_4 by adding a universal vertex. The cop-number of H equals one, whereas it equals two in C_4.

2 A Fast Robber in Large Grid

This section is devoted to prove the following theorem.

Theorem 1. *For any grid G of size n, $c_{1,2}(G) = \Omega(\sqrt{\log(n)})$.*

To prove Theorem 1, we propose an evasion strategy for the robber. This strategy is formally described in section 2.2. The proof of its correctness mainly follows

Lemma 3. Lemmata 1 and 2 are technical results that allow us to prove Lemma 3. Due to lack of space, most of the proofs are omitted and can be found in [NS08].

2.1 Definitions

We consider a robber that is slightly faster than the cops running in a square grid. A square grid G on $n \times n$ vertices is the graph where the vertices can be naturally assigned to the points of positive integer coordinates in the square $n \times n$ of the plane, with edges joining each vertex to its closest neighbors (with respect to the Euclidean metric). We say that the *size* of G is n. In order to prove that the number of cops needed to capture the robber is unbounded, for each number k of cops, we will construct a grid $G(k)$ of size $f(k)$, and a corresponding strategy, by which the robber can infinitely evade the cops.

Given the number of cops k and the corresponding grid $G = G(k)$, a key to our analysis lies in fixing a recursive partition of the grid into gradually smaller subgrids of levels k down to 0. Each level i corresponds to the game played on a subgrid of size $size_i$, with only i cops taken into consideration. At each step s, the subgrid of level i, or *i-subgrid*, currently occupied by the robber is denoted by R^i_s. Let us fix an ordering of the cops: $cop_1, \ldots, cop_k$. The sizes of subgrids are chosen such that there is a strategy allowing the robber to successfully evade i cops in the i-subgrid R^i around him, and to move to neighboring i-subgrids fast enough not to let other cops enter into R^i. And that is for each i between 0 and k. Let us introduce some notation that we use in order to describe the above mentioned strategy on the graph $G = G(k)$.

$zoom = (zoom_1, \cdots, zoom_k)$ is a sequence of scaling factors, that is, an i-subgrid contains $zoom_i \times zoom_i$ vertex disjoint $(i-1)$-subgrids. This means that $size_i$ is equal to $zoom_i \times size_{i-1}$, where we fix $size_0 = 2$ as a starting point. We say that an i-subgrid H^i is adjacent to a j-subgrid F^j if there is an edge in G incident to a vertex in each of them. When i is clear from the context, an $(i-1)$-subgrid relative to an i-subgrid is called a *square* on a *board*. A *path of squares* is a sequence of squares such that any square is adjacent to its predecessor, and its *length* is simply the number of squares. In this way, we can notice a fractal-like structure of G, with the grid topology of squares on boards of corresponding levels. Let us introduce a *coordinate system* for subgrids at each level. The coordinates of an i-subgrid H are (abs_H, ord_H), which correspond to the row (bottom-up) and column (left-right) occupied by H in the partition of G into subgrids of $size_i$. In other words, a vertex v is in H iff the abscissa of v is between $(abs_H - 1) * size_i + 1$ and $abs_H * size_i$, and the ordinate of v is between $(ord_H - 1) * size_i + 1$ and $ord_H * size_i$.

$margin = (margin_1, \cdots, margin_k)$ is a sequence of safety distances. Given an i-subgrid H^i, we note by $around(H^i)$ the subgrid induced by the i-subgrids that are near H^i. More formally, $around(H^i)$ is the subgrid induced by the i-subgrids H, such that $|ord_{H^i} - ord_H| \leq 1$ and $|abs_{H^i} - abs_H| \leq 1$. Similarly, we define the *margin* of H^i, denoted by $margin(H^i)$, as the subgrid induced by the i-subgrids H, such that $|ord_{H^i} - ord_H| \leq m_i$ and $|abs_{H^i} - abs_H| \leq m_i$.

For any i, a configuration in which cop_i is outside the subgrid $around(R^{i-1})$ (i.e., cop_i does not occupy any vertex of it), where R^{i-1} is the square occupied by the robber, is a *valid position* at level i. If moreover, the cop cop_i is also outside $margin(R^{i-1})$ and $margin(R^{i-1})$ is a subgraph of R^i, the position of the robber is called a *nice position* at level i.

Definition 1. *The robber occupies an i-nice position if it occupies a nice position at level j, for all $1 \leq j \leq i$.*

Suppose the robber is in a nice position at level i. If cop_i occupies a square adjacent to the margin, we say that the cop is *blocking a side*. If cop_i occupies a square in a corner (adjacent to two other squares blocking different sides), we say that the cop is *blocking a corner*.

$detour = (detour_1, \cdots, detour_k)$ is a sequence of extra distances. At level i, $detour_i$ is an upper bound on the additional distance that the robber needs to travel in order to evade cop_i. More precisely, starting from a nice position at level i, the length of the path of squares that the robber will follow to go into a nice position in a neighboring board is upper-bounded by $zoom_i + detour_i$. Notice that $zoom_i$ is the minimum number of squares that the robber needs to cross in order to get from the left extreme (resp., up extreme) of a board H^i to the board F^i next to the right (resp., down) of H^i.

$time = (time_0, \cdots, time_k)$ is a sequence of numbers of rounds. At level i, $time_i$ is an upper bound on the time needed by the robber in order to get from a nice position on a board H^i to a nice position on a neighboring board F^i. Moreover, we set the sequence $velocity = (velocity_0, \cdots, velocity_k)$, with $velocity_i = size_i/time_i$, as the "relative" speed of the robber at level i. Since the robber has speed $velocity_0 = 2$, which is its "absolute" speed, and $size_0 = 2$, we get $time_0 = 1$.

2.2 Informal Description of the Robber'S Strategy

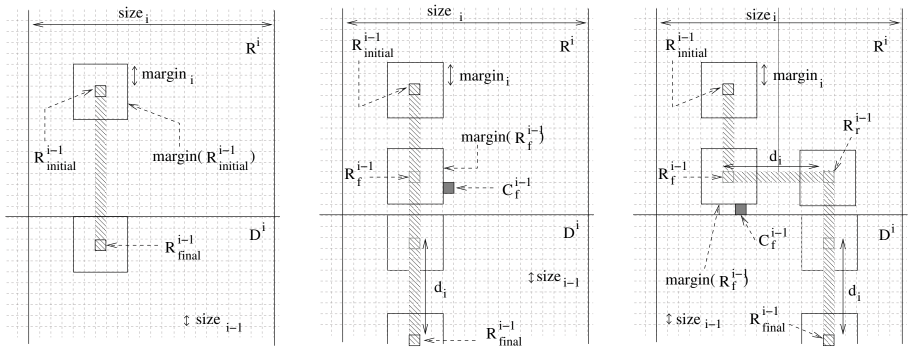
In this section we give an intuitive description of the robber's strategy in order to explain the relations between the sequences defined in the previous section.

Let $1 \leq i \leq k$. Recall that a square denotes an $(i-1)$-subgrid and a board denotes an i-subgrid.

Let us first describe our induction hypothesis at level $i-1$. Let R^{i-1} be the square occupied by the robber. We assume that if all of the cops cop_j, $i-1 < j$, remain outside $around(R^{i-1})$, and if the initial position of the robber is $(i-1)$-nice, then the robber can reach an $(i-1)$-nice position in any square adjacent to R^{i-1}, in at most $time_{i-1} < size_{i-1}$ rounds. Let us describe the robber's strategy that ensures that the induction hypothesis remains valid at the level i.

Let R^i be the board that is occupied by the robber. We assume that all of the cops cop_j, $j > i$, remain outside $around(R^i)$ during the whole game that we will describe. Moreover, let us assume that, for any $1 \leq j \leq i$, the initial position of the robber is nice at level j. In other words, the robber occupies an i-nice position. In particular, it means that cop_i is outside the subgrid $margin(R^{i-1})$ and $margin(R^{i-1})$ is a subgraph of R^i, where R^{i-1} denotes the $(i-1)$-subgrid initially occupied by the robber. Let D^i be a board that is a neighbor of R^i. We describe a strategy for the robber that ensures that (1) the robber reaches an i-nice position in D^i in at most $time_i < size_i$ steps, and (2) cop_i remains outside $around(R^{i-1})$ during the whole game.

For ease of description, we assume that D^i is below R^i (i.e., D^i has smaller ordinate than R^i). This strategy is depicted in Figure 1. In Figures 1(a), 1(b), and 1(c), the hatched zone corresponds to the path of squares covered by the robber during the game.



(a) Illustration of Strategy 1

(b) Illustration of Strategy 2

(c) Illustration of Strategy 3

Fig. 1. Strategy performed by the robber if (a) cop_i never blocks a side of $margin(R^i_s)$, (b) cop_i blocks any but the bottom side of $margin(R^i_s)$, and (c) cop_i blocks the bottom side of $margin(R^i_s)$

At each step $s \geq 0$ of the game, let R^{i-1}_s and C^{i-1}_s be the squares occupied by the robber and cop_i respectively. Roughly speaking, the strategy consists in the following. While cop_i is outside $margin(R^{i-1}_s)$ and does not block neither a side nor a corner of $margin(R^{i-1}_s)$, the robber goes down, that is, it goes to the square H^{i-1} that is the below-neighbor of R^{i-1}_s. By applying the induction hypothesis, the robber reaches an $(i-1)$-nice position in H^{i-1} in at most $time_{i-1}$ steps. If, performing that way, the robber reaches a square H^{i-1} such that $margin(H^{i-1})$ is a subgraph of D^i, we are done. Moreover, it has taken at most $zoom_i * time_{i-1}$ steps. This strategy is illustrated in Figure 1(a). If at some step f of the game, cop_i is blocking a side or a corner of $margin(R^{i-1}_f)$, we consider different cases according to which side or corner of $margin(R^{i-1}_f)$ is blocked.

– Let us first assume that cop_i blocks a side or a corner above $margin(R^{i-1}_f)$. That is, C^{i-1}_f has greater ordinate than any square in $margin(R^{i-1}_f)$. Then, the strategy remains the same: the robber goes down (cf. Figure 1(a)). The robber traverses a square in at most $time_{i-1} < size_{i-1}$ steps, whereas cop_i needs $size_{i-1}$ steps. By Lemma 1, $size_{i-1} > t_{i-1}$. Therefore, each time the robber moves to a new square R^{i-1}_s, cop_i is outside $margin(R^{i-1}_s)$. In particular, this is the case at the step when $margin(R^{i-1}_s)$ is contained in D^i for the first time. Then the strategy achieves.

– Let us now assume that cop_i blocks a side or a corner at the left (resp. at the right) of $margin(R^{i-1}_f)$. That is, C^{i-1}_f has smaller (resp., greater) abscissa than any square in $margin(R^{i-1}_f)$. Again, the strategy consists in going down. However, this time, after the first step when $margin(R^{i-1}_s)$ is a subgraph of D^i, the robber continues going down along a path of d_i extra squares in D^i. This is because the cop may enter in $margin(R^{i-1}_s)$ during this passage, and we want to have it outside the margin in the end. This strategy is illustrated in Figure 1(b). d_i corresponds to an extra distance that the robber must cover in order to avoid cop_i. $detour_i$ will be taken equal to an upper bound of this extra distance in any of the strategies described below. For our strategy to be valid, we must ensure two properties. First, in order to apply the induction hypothesis, cop_i must permanently remain outside $around(R^{i-1}_s)$. Second, at some step $s \leq time_i$, the robber must reach an i-nice position in D^i, that is, cop_i must be outside $margin(R^{i-1}_s)$ while $margin(R^{i-1}_s)$ is a subgraph of D^i. In order to ensure the above two properties, we set several inequalities between $size_{i-1}, zoom_i, margin_i, detour_i, time_{i-1}$, $velocity_{i-1}$ and $time_i$.
  - For the first property to be satisfied, it is sufficient to ensure that, if C^{i-1}_f is blocking the left-bottom corner of $margin(R^{i-1}_f)$ and cop_i goes to the right while the robber is going down, then cop_i cannot enter $around(R^{i-1}_s)$. Indeed, one can observe that the cop occupying this position yields the worst possible case of blocking a side or a corner. Let N be the minimum number of steps that are necessary for the cop to intercept the robber, and let M be the maximum number of steps that are necessary for the robber to cross the place of this hypothetical interception. In other words, we want that $M < N$. Recall that $size_{i-1}$ is the minimum number of steps for a cop to traverse a square (from one of its sides to cross the opposite one), whereas $time_{i-1}$ is the maximum number of steps for the robber to cover the same distance. By looking at Figure 2(a), it is easy to be convinced that $N > (margin_i - 1)size_{i-1}$, and $M < (4 + margin_i)time_{i-1}$. In Figure 2(a), $M_1 = 4 + margin_i$ and $N_1 = margin_i - 1$. Hence, we get our first inequality. For any i, $1 \leq i \leq k$:
$$margin_i \geq \lceil \frac{4 + velocity_{i-1}}{velocity_{i-1} - 1} \rceil \tag{1}$$
  - For the second property to be satisfied, it is sufficient to ensure that, if the step f is such that the squares of $margin(R^{i-1}_f)$ with the greatest ordinate are still in R^i and all the other squares of it are in D^i, and C^{i-1}_f is the left-bottom corner of $margin(R^{i-1}_f)$, then cop_i is above $margin(R^{i-1}_h)$ at the last step h of the game, and $margin(R^{i-1}_h)$ is a subgraph of D^i. Again, this position of R^{i-1}_f leads to the worse possible configuration of this case. Let N be the minimum number of steps that are necessary for the cop to reach $margin(R^{i-1}_h)$, and let M be the

maximum number of steps that are necessary for the robber to reach R_h^{i-1}. Again, we want that $M < N$. Moreover, $h \leq t_i$. Looking at Figure 2(b), it is easy to be convinced that $N > (d_i - 2*margin_i - 2)size_{i-1}$, and $M < d_i * time_{i-1}$. Hence, we get our second equation. For any i, $1 \leq i \leq k$:

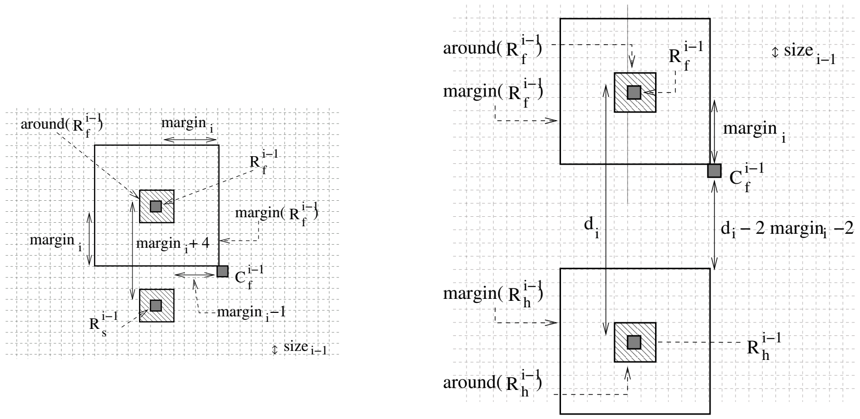
$$d_i \geq \lceil \frac{(2 * margin_i + 2)velocity_{i-1}}{velocity_{i-1} - 1} \rceil \tag{2}$$

For the final position of the robber to be nice, we also need $margin(R_h^{i-1})$ to be a subgraph of D^i, that is:

$$d_i + 2margin_i + 1 < zoom_i$$

Finally, the whole game must take at most $time_i < size_i$ steps, therefore:

$$(zoom_i + d_i)time_{i-1} < (zoom_i + detour_i)time_{i-1} \leq time_i < size_i$$



(a) $(margin_i - 1)size_{i-1} \geq (4 + margin_i)time_{i-1}$ (b) $(d_i - 2*margin_i - 2)size_{i-1} \geq d_i * time_{i-1}$

Fig. 2. Illustration of Inequality 1 and 2: (a) cop_i must never enter in $around(R_s^i)$, and (b) the robber must reach a nice position, i.e., cop_i must not enter in $margin(R_h^i)$

– It remains the case when cop_i blocks a side below $margin(R_f^{i-1})$. That is, C_f^{i-1} has smaller ordinate than any square in $margin(R_f^{i-1})$. In this case, the robber chooses the right side, if R_f^{i-1} is closest to this side of R^i, and the left side otherwise. W.l.o.g., let the robber choose the right side. Then, the robber first goes to the right, along a path of d_i squares. Let R_r^{i-1} be the last of these squares, at which the robber arrives at step r. Note that, by Inequality 1, cop_i never enters $around(R_s^{i-1})$ during this phase. Moreover, by Inequality 2, at step r, cop_i is to the left of $margin(R_r^{i-1})$. Starting from step r, the strategy is the same as in the previous case: the robber goes down, and after the first step when $margin(R_s^{i-1})$ is a subgraph of D^i, the robber continues going down along a path of d_i extra squares in D^i. This strategy is illustrated in Figure 1(c). Again, by applying Inequalities 1 and 2, we get that cop_i never enters $around(R_s^{i-1})$ during the whole game, and, at the last step h, cop_i is outside $margin(R_h^{i-1})$. In order to ensure $margin(R_h^{i-1})$ to be a subgraph of D^i, we need the following inequality:

$$d_i + 2 * margin_i + 1 < zoom_i/2 \tag{3}$$

Finally, the whole game must take at most $time_i < size_i$ steps, therefore:

$$(zoom_i + 2d_i)time_{i-1} \leq (zoom_i + detour_i)time_{i-1} \leq time_i < size_i \tag{4}$$

In the following, we turn Inequalities 1 and 2 into equalities, we set $detour_i = 2d_i$, and we prove that, for a sequence *zoom* well chosen, Inequalities 3 and 4 are satisfied.

2.3 Proof of Theorem 1

We first prove that there are two constants $a > 0$ and $b > 2$, such that to set $zoom_i = ab^i$ (for all $i \leq k$) ensures that Inequalities 3 and 4 are valid.

Let $k \geq 1$ and $velocity_0 = 2$. We will now precisely define the sequence *zoom*, and define the relations between the sequences *zoom*, *margin*, *detour*, *time*, *size* and *velocity*. For any $1 \leq i \leq k$, let us turn Inequalities 1 and 2 into equalities:

$$\begin{aligned} margin_i &= \lceil \frac{4 + velocity_{i-1}}{velocity_{i-1} - 1} \rceil, \\ and,\ detour_i = 2d_i &= 2 * \lceil \frac{(2 * margin_i + 2)velocity_{i-1}}{velocity_{i-1} - 1} \rceil. \end{aligned} \tag{5}$$

We also set:

$$time_i = (zoom_i + detour_i)time_{i-1} \tag{6}$$

From Equations 5, 6 and the fact that $velocity_i = size_i/time_i$, we get that $velocity_i = \frac{zoom_i}{zoom_i + detour_i} * velocity_{i-1} \geq \beta_i * velocity_{i-1}$, where β_i is defined by:

$$\begin{aligned} \beta_i &= \frac{zoom_i}{zoom_i + 2 * (\frac{(2*(\frac{4+velocity_{i-1}}{velocity_{i-1}-1}+1)+2)velocity_{i-1}}{velocity_{i-1}-1} + 1)} \\ &= \frac{zoom_i}{zoom_i + \frac{2+4*velocity_{i-1}+14*velocity_{i-1}^2}{(velocity_{i-1}-1)^2}} \end{aligned}$$

Finally, let us assign some values to the sequence *zoom*, in order to satisfy Inequalities 3 and 4. For this purpose, let us set $2 > \alpha > 1$, and let $a = \lceil \frac{20}{(\alpha-1)^2} \rceil * \lceil \frac{2}{\ln(velocity_0/\alpha)} \rceil$ and let b be an integer such that $b > \max\{2, \frac{\ln(velocity_0/\alpha)}{2}\}$. For any $1 \leq i \leq k$, we set

$$zoom_i = ab^i. \tag{7}$$

Due to lack of space, formal proofs of Lemmata 1 2 and 3 are omitted and can be found in [NS08].

Lemma 1. *Inequality 4 is satisfied. That is, for any* $0 \leq i \leq k$, $velocity_i = size_i/time_i > 1$.

Sketch of the Proof. We prove by induction on i, $0 \leq i \leq k$, that $2 \geq velocity_i > \alpha$. This induction allows us to prove that $1/\beta_i < 1+1/(\frac{2}{\ln(velocity_0/\alpha)} * b^i)$. Then, the result follows from some calculations. □
Lemma 2 is straightforward from Equality 5 and Lemma 1.

Lemma 2. *Inequality 3 is satisfied. That is, for any* $1 \leq i \leq k$, $detour_i + 4 * margin_i + 2 < zoom_i$

Both previous Lemmata allow us to prove the following:

Lemma 3. *Let* i, $1 \leq i \leq k$. *Let us assume that the robber occupies an* i*-nice position in a level-*i *subgrid* R^i *in* G. *Moreover, let us assume that, all of the cops* cop_j, $j > i$, *permanently remain outside* $around(R^i)$. *Let* D^i *be any level-*i *subgrid adjacent to* R^i.

The strategy described in section 2.2 ensures that

1. *the robber reaches an* i*-nice position in* D^i *in at most* $time_i < size_i$ *steps*
2. cop_i *remains outside* $around(R^{i-1})$ *during the whole game.*

Sketch of the Proof. The proof is by induction on i, $0 \leq i \leq k$. To prove both items, it is sufficient to follow the description of the strategy (section 2.2), and to use Lemmata 1 and 2. □
We are now able to prove Theorem 1

Proof of Theorem 1. More precisely, we prove that, for any $k \geq 1$, one robber with speed $velocity_0 = 2$ can infinitely evade k cops with speed one in any grid of size more than $4a^k b^{k(k+1)/2}$, where a and b are defined as previously.

Let G be the grid of size $2*size_k = 2*size_0*\prod_{1\leq i\leq k} zoom_i = 4*a^k*b^{k(k+1)/2}$. Note that, if one robber can infinitely evade k cops in G, it can perform the same strategy and evade k cops as well in any bigger grid. It remains to prove that the strategy described in Section 2.2 enables the robber to infinitely evade k cops in G.

Now, let us assume that k cops are placed on vertices of G. G is divided into 4 vertex-disjoint subgrids of size $size_k$ (i.e., level-k subgrids). Let us fix an ordering of the cops $(cop_1, \ldots, cop_k)$. Choose one of the level-k subgrids not occupied by cop_k, and denote it by R^k. Notice that, by Equation 3, R^k contains at least four $(k-1)$-subgrids $R_1^{k-1}, \ldots, R_1^{k-1}$ such that $margin(R_i^{k-1})$, $1 \leq i \leq 4$, are disjoint and entirely contained in R^k. Any position inside these subgrids is nice at level k. Recursively, choose one not occupied by cop_{k-1} to be R^{k-1}, and proceed until finding R^0. Any position inside R^0 is k-nice and we may pick it as the initial position for the robber. The top level strategy consists in traversing the four level-k subgrids of G along the cycle given by their adjacencies. Lemma 3 (by taking $i = k$) proves that, starting from a k-nice position in some level-k subgrid R^k, the robber can reach a k-nice position in any level-k subgrid adjacent to R^k, without being caught by the cops. By repeating this process infinitely, the robber can infinitely evade k cops in G, which proves Theorem 1. □

Corollary 1. *For any grid G of size n, and for any $1 \leq p < q$, $c_{p,q}(G) > \Omega(\sqrt{\log(n)})$.*

Proof. Let $0 < \alpha < 1$. By setting, $size_0 = q$, $a = \lceil \frac{2+4q+14q^2}{(p+\alpha-1)^2} \rceil$, and $b > \max\{2, \frac{\ln(q/(p+\alpha))}{2}\}$, the proof the corollary follows the proof of Theorem 1.

3 Fast Robber Cop-Number with Respect to Graph Containment

We have seen that the number of cops needed to capture a fast robber in a grid G may be arbitrarily large. It would be interesting to see if a high value of the cop-number of a planar graph H is related to a large grid G somehow contained in H. On the negative side, the classical transformations of edge removal, vertex removal, and edge contraction do not preserve bounded cop-number. Moreover, there are graphs of arbitrarily large tree-width [Bod98] (that is, somehow containing a large grid) and cop-number two. Due to lack of space, the formal proof of the following proposition is omitted and can be found in [NS08].

Proposition 2. *For any $k \geq 1$, there is a planar graph H with $c_{1,2}(H) \leq 2$, such that a graph G with $c_{1,2}(G) \geq k$ can be obtained from H by contracting edges (resp., by removing edges, resp., by removing vertices).*

Sketch of the Proof. We sketch the proof for G obtained from H by contracting edges. Let $k \geq 1$. Let G be a grid of size $n \geq f(k)$. Let P be a column (vertical path) of G, and let H be the graph obtained by replacing each vertical edge but those of P by a path of length $6n$. Roughly, the strategy for two cops consists in moving along P from one line to another, until they occupy two consecutive lines while the robber is occupying a path P' of length $6n$ between those two lines. Then, one cop moves to block the robber's way back, and the other cop moves to block the other end of P'. The length of P' is such that once the robber's way back is blocked, it can not reach the other end of P' before the other cop blocks it. □

Nevertheless, we can define a larger family of planar graphs of high cop-number than the grids themselves. Due to lack of space, the formal proof of the following theorem is omitted and can be found in [NS08].

Theorem 2. *Let H be a planar graph containing a grid G of size $4 * size_k$ as an induced subgraph, then $c_{1,2}(H) \geq k$.*

Sketch of the Proof. A theorem of Whitney (see Theorem 4.3.2 of [Die05]) proves that G admits a unique embedding in the sphere. This allows to prove that H contains a subgrid G' of size $size_k$ as a distance-hereditary induced-subgraph. The evasion strategy described in section 2.2 can be easily adapted to H, with the robber restricted to stay in G'. Since the correctness of this strategy is mainly based on the distance between the robber and the cops, it is easy to adapt the proof of Theorem 1 to H. □

References

[AF84] Aigner, M., Fromme, M.: A game of cops and robbers. Discrete Applied Mathematics 8, 1–12 (1984)

[AF88] Anstee, R.P., Farber, M.: On bridged graphs and cop-win graphs. J. Comb. Theory, Ser. B 44(1), 22–28 (1988)

[Als04] Alspach, B.: Searching and sweeping graphs: a brief survey. In: Le Matematiche, pp. 5–37 (2004)

[And84] Andreae, T.: Note on a pursuit game played on graphs. Discrete Applied Mathematics 9, 111–115 (1984)

[And86] Andreae, T.: On a pursuit game played on graphs for which a minor is excluded. J. Comb. Theory, Ser. B 41(1), 37–47 (1986)

[BI93] Berarducci, A., Intrigila, B.: On the cop number of a graph. Adv. in Applied Math. 14, 389–403 (1993)

[Bod98] Bodlaender, H.L.: A partial -arboretum of graphs with bounded treewidth. Theor. Comput. Sci. 209(1-2), 1–45 (1998)

[Che97] Chepoi, V.: Bridged graphs are cop-win graphs: An algorithmic proof. J. Comb. Theory, Ser. B 69(1), 97–100 (1997)

[Die05] Diestel, R.: Graph Theory, 3rd edn. Springer, Heidelberg (2005)

[Far87] Farber, M.: Bridged graphs and geodesic convexity. Discrete Applied Mathematics 66, 249–257 (1987)

[Fra87] Frankl, P.: Cops and robbers in graphs with large girth and cayley graphs. Discrete Applied Mathematics 17, 301–305 (1987)

[GR95] Goldstein, A.S., Reingold, E.M.: The complexity of pursuit on a graph. Theor. Comput. Sci. 143(1), 93–112 (1995)

[HM06] Hahn, G., MacGillivray, G.: A note on k-cop, l-robber games on graphs. Discrete Math. 306, 2492–2497 (2006)

[NS08] Nisse, N., Suchan, K.: Fast robber in planar graphs. Technical Report, CMM-B-08/03-200, Santiago, Chili (March 2008)

[NW83] Nowakowski, R.J., Winkler, P.: Vertex-to-vertex pursuit in a graph. Discrete Mathematics 43, 235–239 (1983)

[Qui83] Quilliot, A.: Thèse de doctorat d'état. Ph.D thesis, Université de Paris VI, France (1983)

[Qui85] Quilliot, A.: A short note about pursuit games played on a graph with a given genus. J. Comb. Theory, Ser. B 38(1), 89–92 (1985)

[Sch01] Schröder, B.S.W.: The copnumber of a graph is bounded by $\lfloor \frac{3}{2} genus(g) \rfloor + 3$. Trends Math., 243–263 (2001)

From a Circular-Arc Model to a Proper Circular-Arc Model

Yahav Nussbaum

School of Computer Science, Tel Aviv University, 69978 Tel Aviv, Israel
nuss@post.tau.ac.il

Abstract. We are given a circular-arc graph, represented by a circular-arc model; our goal is to decide whether the graph is a proper circular-arc graph. We do so in time linear in the number of vertices of the graph, regardless of the number of edges which may be quadratic in the number of vertices. For every input graph, we either provide a proper circular-arc model for the graph, or a forbidden subgraph induced in the graph.

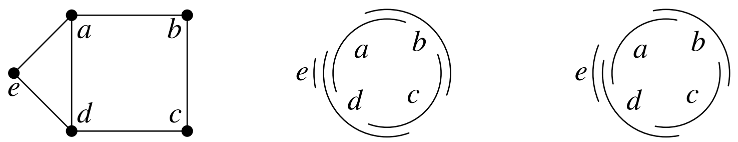
1 Introduction

A *circular-arc graph* (see Fig. 1(a)) is the intersection graph of arcs on the circle. Every vertex in the graph is mapped to an arc on the circle, such that two arcs intersect if and only if the corresponding vertices are adjacent. The set of arcs on the circle constitutes a *circular-arc model* (see Fig. 1(b)). Circular-arc graph are used to represent objects of circular or periodical nature. For an overview on circular-arc graphs, see the books by Golumbic [2] and Spinrad [16].

A circular-arc model is a representation of a circular-arc graph. Given a circular-arc model which represents a graph, we can easily answer the question "Is there an edge between u and v?", by examining the arcs which represent u and v in the model. The representation of a circular-arc graph by a circular-arc model is more space-efficient than general graph representations such as adjacency matrix or adjacency list, since the model does not represent the edges of the graph explicitly. For a graph G with n vertices and m edges, the amount of space that a general representation requires depends on n and m, while the space that a circular-arc model uses depends only n, even for dense graphs where m is quadratic in n. For an overview of efficient representation of graphs see [16]. Given a representation of a graph by its vertices and edges, it takes $O(n + m)$ time to construct a circular-arc model representation for the same graph [8,13].

A circular-arc model representation of circular-arc graph is not efficient just in space. This representation also allows an efficient implementation of several graph algorithms. For example the problems of maximum independent set [3,5,11,12], maximum clique cover [5] and minimum dominating set [5] can all be solved in $O(n)$ time on a circular-arc model, while computing them on an adjacency list representation of the same graph would take $O(n + m)$ time.

A *proper circular-arc model* (see Fig. 1(c)) is a circular-arc model in which no arc contains another arc. A circular-arc graph which can be represented by a



(a) Vertices and edges (b) A circular-arc model (not proper) (c) A proper circular-arc model

Fig. 1. Three representations of the same graphs

proper circular-arc model is a *proper circular-arc graph.* Given a representation of a graph by its vertices and edges, it takes $O(n+m)$ time to construct a proper circular-arc graph which represents the same graph [1,7].

In this paper, we present an algorithm that gets a circular-arc model as input and produces a proper circular-arc model which represents the same graph. If no such a model exists, then our algorithm provides one of the forbidden subgraphs of Tucker [17] induced in the input circular-arc model. The running time of the algorithm is $O(n)$, this running time is better than the time required to construct a proper circular-arc model without a circular-arc model as an input, as it does not depend on the number of edges in the graph, which might be as large as $\Theta(n^2)$. In addition, if the input model represents a proper interval graph, then our algorithm produces a proper interval model, even if the input model is not an interval model.

A representation by proper circular-arc model has several advantages over a representation by a circular-arc model. For example, for routing on ring networks we require a coloring algorithm to color a circular-arc graph which represents a network. While coloring a circular-arc graph in NP-complete, coloring a proper circular-arc model takes polynomial time [15]. Another example is using circular-arc model to represent periodic scheduling, such as of traffic lights or of staff shift work. A proper circular-arc model can be transformed into a unit circular-arc model in $O(n)$ time [9] for an equal scheduling.

Our algorithm uses tools similar to the algorithm of Joeris et al. [6], which produces a Helly circular-arc model from a given circular-arc model in $O(n)$ time. Other related results are Lin and Szwarcfiter [9] and Lin et al. [10] which produces a unit circular-arc model and a proper Helly circular-arc model from a proper circular-arc model in $O(n)$ time, as well as algorithms such as those mentioned above [3,5,11,12,15] which run in $O(n)$ time or other time bounds lower than $O(n+m)$ on circular-arc models.

2 Preliminaries

We consider a finite, simple, circular-arc graph G. We denote the number of vertices in G by n. We denote the complement of a graph H by $\overline{H}$.

A circular-arc model is a set of arcs on a circle. The graph G is represented by a circular-arc model M. Note that M is not necessarily the only circular-arc model which represents G. For convenience, we refer to the clockwise direction of a circle as *right* and to the counterclockwise direction of a circle as *left*, as we view them if we stand in the center of the circle. We may assume that no two endpoints of arcs in a circular-arc model coincide, since otherwise this can be fixed by a slight shift of the endpoints. We also assume that no single arc covers the entire circle of a circular-arc model, because such an arc can be replaced by an arc whose right endpoint is immediately to the left of its left endpoint.

Every vertex in G is represented by an arc in M. We do not distinguish in our notation between a vertex x in G and the arc x which represents it in M. Thus, we may use equivalent terms such as "the vertices x and y are adjacent" and "the arcs x and y intersect" interchangeably. Each arc x has two endpoints we denote the left endpoint by $\ell(x)$ and the right endpoint by $r(x)$.

A circular-arc model is represented by a cyclic (doubly-linked) list of all endpoints of all arcs in the model. Since G has n vertices, every circular-arc model of G has $2n$ endpoint. Another possible representation for a circular-arc model is by storing for each arc the positions of its endpoints in the cyclic list of endpoints. It is easy to convert each of these two representation to the other in $O(n)$ time. These representations of circular-arc models require $\Theta(n \log n)$ bits, while adjacency matrix representation of a graph requires $\Theta(n^2)$ bits and adjacency list representation takes $\Theta(m \log n)$ bits.

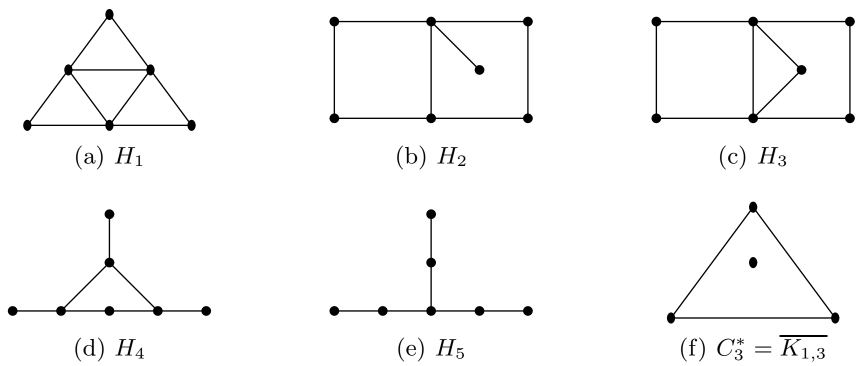
Two arcs x and y in a circular-arc model may either be *disjoint* or *intersect* each other. If x intersects y then x may *contain* y, be *contained in* y, (left or right) *cross* y, or *double overlap* y. Given the endpoints of two arcs in a circular-arc model, we can determine the type of intersection between the arcs from the relative order of their endpoints. We say that x *can contain* y if there is a circular-arc model in which x contains y, and there is an arc z which intersects x but not y. This property depends on the graph, and not on the specific model. The arcs x and y double overlap in some circular-arc model only if every arc which is disjoint from x in this model, is contained in y. A *universal arc* is an arc which intersects every other arc in the model.

A *proper circular-arc model* is a circular-arc model in which no arc contains another arc. An *interval model* is a set of intervals on the line, equivalently it is a circular-arc model in which the circle is not covered by the union of the arcs. A model which is both a proper circular-arc model and an interval model is a *proper interval model*. A graph which can be represented by a proper circular-arc (interval, proper interval) model is a proper circular-arc (respectively: interval, proper interval) graph.

We say that an endpoint e is *between* an endpoint e_1 and an endpoint e_2, if when we traverse the cyclic list of endpoints starting at e_1 and going right, we encounter e before e_2.

3 Forbidden Subgraphs

Tucker [17] gave a characterization of proper circular-arc graphs using forbidden induced subgraphs. If G is not a proper circular-arc graph, then our algorithm finds one of these forbidden subgraphs. Since our input graph is a circular-arc


(a) H_1 (b) H_2 (c) H_3

(d) H_4 (e) H_5 (f) $C_3^* = \overline{K_{1,3}}$

Fig. 2. Complements of forbidden subgraphs

graph, we use only part of the forbidden subgraphs which Tucker showed. These forbidden subgraphs are $\overline{H_1}$, $\overline{H_2}$, $\overline{H_3}$, $\overline{H_4}$, $\overline{H_5}$, and $\overline{C_i^*}$ for odd $i \geq 3$, where H_j are as described in Fig. 2 and C_i^* is an induced cycle of i vertices with an isolated vertex. The graph $\overline{C_3^*}$ (Fig. 2(f)) is also called $K_{1,3}$.

Our algorithm provides one of the forbidden subgraphs induced in the input circular-arc model M. The subgraph is provided as a set of arcs in M. The subgraphs $\overline{H_1}$, $\overline{H_2}$, $\overline{H_3}$, $\overline{H_4}$, $\overline{H_5}$ are of constant size and therefore it takes constant time to verify that the arcs represent an induced subgraph. The family of subgraphs $\overline{C_i^*}$ is made of a universal vertex and the complement of a cycle of odd length. Tucker [18] showed that there is only one way to order the endpoints of a complement of a cycle of odd length in a circular-arc model (see the set W on Sec. 6). Therefore, it takes time linear in the size of the model to verify that the arcs represent a complement of a cycle of odd length and a universal vertex.

4 Function on Arcs

Let X and Y be two sets of arcs in a circular-arc model, not necessarily disjoint. Joeris et al. [6] show a family of functions which can be computed in $O(|X|+|Y|)$ time for every $x \in X$. The functions of [6] are, for every $x \in X$: find $y \in Y$ which is contained in x such that $r(y)$ is closest to $r(x)$ from the left; find $y \in Y$ which is disjoint from x such that $\ell(y)$ is closest to $r(x)$ from the right; find $y \in Y$ which right crosses x such that $r(y)$ is farthest to the right from $r(x)$; and find $y \in Y$ which right crosses x such that $r(y)$ is closest to $r(x)$ from the right. The symmetric functions obtained by exchanging left and right are also defined.

Using the functions of [6] we define more similar functions which can also be computed in $O(|X| + |Y|)$ time. For every $x \in X$: find $y \in Y$ which is disjoint from x such that $\ell(y)$ is farthest to the right from $r(x)$; find $y \in Y$ which double overlaps x such that $r(y)$ is closest to $\ell(x)$ from the right. We refer to this collection of functions as *Functions on Arcs.*

Functions on Arcs allow us to determine for every arc x whether there exists an arc y such that x contains (is disjoint from, double overlaps) y, in $O(n)$ time.

Algorithm 1. Implement all possible arc containments in a model M.

```
1. Split $M$ into blocks. For each arc determine the blocks of its two endpoints.
2. For every block of right endpoint $B$, order the right endpoints inside $B$ such that
   $r(x)$ is to the *right* of $r(y)$ if and only if $\ell(x)$ is to the *left* of $\ell(y)$ when we traverse
   $M$ starting at $B$.
3. Repeat the previous step symmetrically for every block of left endpoints.
```

Algorithm 2. Eliminate arc containments in a circular-arc model M.

```
1. Split $M$ into blocks. For each arc determine the blocks of its two endpoints.
2. For every block of right endpoint $B$, order the right endpoints inside $B$ such that
   $r(x)$ is to the *right* of $r(y)$ if and only if $\ell(x)$ is to the *right* of $\ell(y)$ when we traverse
   $M$ starting at $B$.
3. Repeat the previous step symmetrically for every block of left endpoints.
```

A *block* [4,6] is a maximal sequence of consecutive endpoints of the same side (all right or all left) in a circular-arc model. Changing the order of endpoints inside a block in a circular-arc model does not change the graph which is represented by the model. Therefore, a circular-arc graph can be also represented by a cyclic order of blocks of endpoints rather than a cyclic order of endpoints.

Hsu [4] and Joeris et al. [6] define Algorithm 1. In the circular-arc model resulting from this algorithm, for every pair of arcs x and y, if x can contain y then x contains y [4]. This is because if x crosses y in some model, but can contain y, then x and y have one of their endpoints in the same block. We define a similar algorithm, Algorithm 2 which eliminates arc containments in the circular-arc model, for arc x which contains arc y such that x and y have an endpoint in the same block. Algorithms 1 and 2 can be implemented in $O(n)$ time using radix sort. Algorithm 3 is a linear-time implementation of an algorithm given in [4], using Functions on Arcs (a similar algorithm is also defined by [6]). Note that this algorithm is defined only for circular-arc models without universal arcs. Let M' be a model produced by Algorithm 3. Every pair of arcs x and y which double overlap in some circular-arc model of the same graph, double overlap in M' [4], this is because if x does not double overlap y, then there is an arc which is disjoint from x and not contained in y.

Golumbic and Hammer [3] give Algorithm 4 which checks whether a circular-arc model is a proper circular-arc model. If the model is not a proper circular-arc model, Algorithm 4 provides a pair of arcs such that one contains the other.

5 The Algorithm

In this section we present an algorithm which transforms M into a proper circular-arc model, if G is a proper circular-arc graph. Using Functions on Arcs we can detect all universal arcs in M in $O(n)$ time and remove them from the

Algorithm 3. Stretch arcs in a circular-arc model M without universal arcs.

```
1. Apply Algorithm 1 on $M$.
2. For every right endpoint $r(x)$, let $y$ be the arc which is disjoint from $x$ such that
   $\ell(y)$ is closest to $r(x)$ from the left. Move $r(x)$ to be immediately to the left of $\ell(y)$.
3. Repeat steps 1 and 2 symmetrically for every left endpoint.
```

Algorithm 4. Find arc containment in a circular-arc model M.

```
1. Let $Q$ be an empty queue. For each arc $x$ of $M$ we store a value that indicates
   whether $x$ is in $Q$. Let $e$ be an arbitrary point on $M$.
2. Traverse the circle of $M$ twice, starting at $e$ and going right. Process the endpoints
   as follows – For every left endpoint $\ell(z)$, add $z$ to the tail of $Q$. For every right
   endpoint $r(z)$, if $z$ is in $Q$, remove $x$ from the head of $Q$, if $x \neq z$ then the arc $x$
   contains the arc $z$.
```

model. In the next section we show how to embed these arcs back. Let M_0 be the model M without universal arcs.

Algorithm 2 eliminates arc containments in the circular-arc model, only for arc x which contains arc y such that x and y have an endpoint in the same block. Intuitively, this algorithm lets a contained arc stretch outside of the containing arc, if there is no arc which obstructs it. Assume that in a model which was produced by Algorithm 2, arc x contains arc z. Then, since the endpoints of x and z are not in the same blocks, there is an endpoint $r(u)$ between $\ell(x)$ and $\ell(z)$ and an endpoint $\ell(w)$ between $r(z)$ and $r(x)$. The arcs u and w are in one of three possible relations – they might be disjoint, cross each other, or be the same arc – we denote the configurations of x, z, u and w in these three cases by N_d, N_X and $N_=$ respectively (see Fig. 3).

The arcs in N_d are $K_{1,3}$, and therefore a model which contains an N_d does not represent a proper circular-arc graph. But, the configurations N_X and $N_=$ might be avoided by moving the endpoints of the model between the blocks. The change is made by stretching and shrinking the arcs of M. We *stretch* an arc by making it cover larger part of the circle, without changing the intersections in the model. We *shrink* an arc by making it cover smaller part of the circle, without changing the intersections in the model. Intuitively, the stretching eliminates any N_X while the shrinking eliminates any $N_=$.

Algorithm 3 stretches the arcs of the model as far as possible. Let M_1 be the result of applying Algorithm 3 on M_0.

A set of arcs X is a *clique module* if for every $x \in X$ the set of arcs which x intersects, including x itself, is identical. If x and x' are members in the same clique module, then their endpoints are in the same blocks in M_1 because x intersects x' and is disjoint from any arc which has an endpoint in the block left to the block of $\ell(x)$ or right to the one of $r(x)$. We detect all clique modules in M_1 in $O(n)$ time by radix sorting the arcs, using the blocks which contain the endpoints of the arcs as indices. For every clique module, we remove all but one

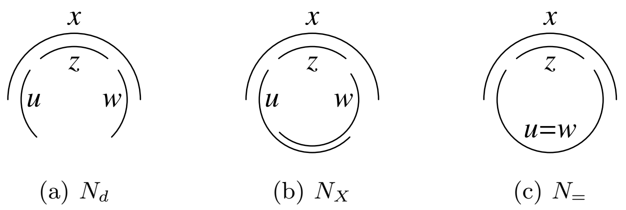


Fig. 3. Arc containments after applying Algorithm 2

of its arcs from the model, let x be this arc. We can embed back the removed arcs to any proper circular-arc model, by putting all the left endpoints next to $\ell(x)$ and all the right endpoints next to $r(x)$, in the same order. We denote the graph and model without universal arcs and clique modules larger than one by G' and M_2. Note that a forbidden subgraph which is induced in G' is also induced in G. The next lemma shows that the stretching of arcs eliminated any N_X in M_2.

Lemma 1. *If G' is a proper circular-arc graph, then M_2 does not contains an N_X.*

Proof. Assume that M_2 contains arcs x, z, u and w as in Fig. 3(b). We show how to find a forbidden induced subgraph in G'.

Since M_2 is obtained by Algorithm 3, we know that every pair of arcs which can double overlap in some model, double overlaps in M_2. The arcs u and w do not double overlap x. So, there is an arc r which is disjoint from x and is not contained in u and an arc t which is disjoint from x and is not contained in w. Since u (w) does not contain r (t) then there is an arc p (q) which intersects r (t) but not u (w).

Assume that we can choose $r = t$. If both p and q disjoint from x then $\{x, z, u, w, p, q\}$ are $\overline{H_1}$. If p or q intersects x but not z then $\{x, z, u, p\}$ or $\{x, z, w, q\}$ are $K_{1,3}$. Else, if only one of p and q intersects both x and z then $\{x, z, u, r, w, p, q\}$ are $\overline{H_3}$. If both intersect x and z then either $\{x, z, u, w, r, p, q\}$ are $\overline{H_5}$ (p and q intersect) or $\{x, z, u, w, p, q\}$ are $\overline{C_5^*}$ (otherwise).

Assume that no such $r = t$ exist. In this case u contains t. If p is disjoint from x then $\{w, x, p, t\}$ are $K_{1,3}$. If p intersects x but not z, then $\{x, z, u, p\}$ are $K_{1,3}$. If p intersects both x and z then $\{x, z, u, w, r, t, p\}$ are $\overline{H_2}$. □

We run Algorithm 1 on M_2 and get a new model of G' which we denote by M_3. For every two arcs x and y, if x can contain y then x contains y on M_3. Since Algorithm 1 changes the order of endpoints only within blocks, if M_2 does not contain an N_X then the same is true for M_3.

Even if G' is a proper circular-arc graph, then the model M_3 may still contain an $N_=$. Since we removed universal arcs, having an $N_=$ in the model is equivalent to having a pair of arcs that double overlap each other. Every proper circular-arc graph has a proper circular-arc model without such pair of arcs [17,2,7].

We eliminate any $N_=$ in M_3 by shrinking arcs of the model. The shrinking process should keep three invariants true on the model at any stage – *Representation Invariant (I_R)* The model represents G'; *Containment Invariant (I_C)* If

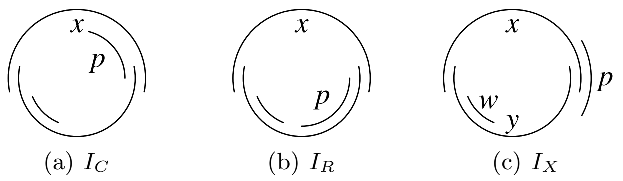


Fig. 4. Moving $r(x)$ causes a violation of one of the three invariants

x can contain y, then x contains y in the model; and N_X *Invariant (I_X)* The model does not contain any N_X. If G' is a proper circular-arc graph then the three invariants are valid in M_3.

Intuitively, I_R is required for the correctness of the algorithm. The invariant I_C is required for the linear time bound, we can keep I_C since if by shrinking an arc x which contains an arc p we both eliminate an $N_=$ and the containment of p in x, then we create an N_X (see Fig. 4(a)). The invariant I_X promises that we do not replace an $N_=$ by an N_X.

We shrink an arc by *moving* its right endpoint in the left direction. Let $r(x)$ be a right endpoint, and let e be the endpoint immediately to the left of $r(x)$.

If e is a right endpoint of an arc p (see Fig. 4(a)) then x contains p, and we do not move $r(x)$ to the left, to avoid violation of I_C.

Assume that e is a left endpoint of an arc p (see Fig. 4(b)). Only if x and p double overlap, then we can move $r(x)$ to be on the left side of $e = \ell(p)$ without violating I_R, since after the move x and p will still intersect. Moreover, in this case x double overlaps any arc which has an endpoint in the same block as e, since any such arc contains p. So, we can move $r(x)$ to be on the left of the block which contains e, without violating I_R. We also do not violate I_C in this move, since we do not change the relative order of left endpoints or of right endpoints. If x and p do not double overlap, then we do not move $r(x)$, to keep I_R.

To keep I_X, we should detect when the moving of $r(x)$ will create an N_X. Assume that x double overlaps some arc y. There is an arc w which is disjoint from x and contained in y. If there is an arc p, such that p right crosses x, does not double overlap y, and is disjoint from w (see Fig. 4(c)), then by moving $r(x)$ to the left of $\ell(y)$, we create an N_X in which y contains w, and the arcs x and p cross y from both sides. In this case we say that x is *not for shrink*. Even though x is not for shrink, we may still be able to shrink y. Another case is when there is an arc p which crosses y such that p is contained in x (see Fig. 4(a)), then moving $r(x)$ beyond $r(p)$ violates I_X, in this case we cannot move $r(x)$ by I_C.

Let x be an arc which double overlap some other arc. Let y_x be the arc which double overlap x such $r(y_x)$ is closest to $\ell(x)$ from the right. Let w_x be the arc which is disjoint from x such that w_x is farthest to the right from $r(x)$. We define x' as the arc whose left endpoint is immediately to the right of $r(y_x)$ and its right endpoint is immediately to the right of $r(x)$. Let p_x be the arc of M_3 which right crosses x', such that $r(p_x)$ is closest to $r(x')$ from the right.

If the arc p_x exists, then it right crosses x and does not double overlap y_x. So, if p_x is disjoint from w_x, then x is not for shrink. On the other hand, if x is not for shrink, then the arc p_x exists and is disjoint from w_x, by the way we choose y_x, w_x, and p_x to be closest or farthest from the endpoints of x.

The discussion above gives us a way to shrink a single arc of M_3. Now we define an algorithm to shrink all arcs of M_3 as much as possible.

Let X be the set of arcs which double overlap other arcs in M_3. We find X and detect for every arc $x \in X$ if it is not for shrink, in $O(n)$ time using Functions on Arcs. For every $x \in X$ we can find w_x and y_x as above using Function on Arcs on the model M_3. Let M_3' be the model which contains the arcs of M_3, and in addition the arc x' for every $x \in X$. If we should put two endpoints in M_3' in the same place, then we order them arbitrarily. The number of arcs in M_3' is $O(n)$. So, we find p_x, for every x, in $O(n)$ time using Functions on Arcs. For $x \in X$, if p_x is defined and is disjoint from w_x, then x is not for shrink.

We split M_3 into blocks. For each arc we keep the blocks of its two endpoints. We keep the blocks in a cyclic doubly-linked list, and for each block we keep pointers to the leftmost and the rightmost endpoints in the block.

Let s be an arc which does not contain any other arc. The arc s also does not overlap any other arc, since if u double overlaps s then u is universal arc. Let S be the block of right endpoints which contains $r(s)$. We start with $B = S$.

Let $r(x)$ be the leftmost endpoint in B (when $B = S$, the arc x is s). The endpoint immediately to the left of $r(x)$ is a left endpoint, let it be $\ell(y)$. If x is not marked as 'not for shrink', and x double overlaps y, then we move $r(x)$ to be the rightmost endpoint in the block of right endpoints that is left to the block which is left to B.

We repeat this step and try to move the leftmost endpoint in B as long as we can move it. If B became empty, then we combine the two blocks next to B into a single block of left endpoints.

Let D be the block of right endpoints which is right to the block which is right of B. If $D = S$ then we are done shrinking. Note that we never remove S from the model, since we cannot move s. Otherwise, we repeat the shrinking process with $B = D$.

Let M_4 be the model that we obtain. The arc s is always the leftmost endpoint of S, since we cannot move it, so we consider any right endpoint at most once. So, we find M_4 in $O(n)$ time since it takes constant time to move a single endpoint.

The model M_4 keeps the three invariants, since we move $r(x)$ as described above for the move of a single endpoint. Also, we cannot move any endpoint $r(x)$ in M_4, since any such endpoint must be the leftmost endpoint in its block and it was also this way when our algorithm considered the block. The next lemma shows that if G' is a proper circular-arc graph then there is no $N_=$ in M_4.

Lemma 2. *If G' is a proper circular-arc graph, then no pair of arcs double overlaps in M_4.*

Proof. (Sketch) If x and y double overlap in M_4, then moving $r(x)$ or $r(y)$ left will cause violation of one of the three invariants. Each of the three invariants defines a type of an obstacle for moving the endpoints. For each of the three possible types of obstacles for $r(x)$ and each of these three for $r(y)$, we can obtain a forbidden induced subgraph in the model, using the arcs which define the obstacles. □

Now, we apply Algorithm 2, which eliminates some arc containments, on M_4. Denote the result by M_5. Since Algorithm 2 only changes the order of endpoints within blocks, it does not create any new N_X or $N_=$.

Theorem 1. *The graph G' is a proper circular-arc graph if and only if M_5 is a proper circular-arc model.*

Proof. The model M_0 that we start with is a circular-arc model of G'. Our algorithm changes the order of endpoints in the model, but does not change the intersections, so M_5 is a model of G'. Therefore, if M_5 is a proper circular-arc model then G' is a proper circular-arc graph.

On the other hand, assume that M_5 is not a proper circular-arc model. Then, since we called Algorithm 2, we know that M_5 contains one of the configurations N_d, N_X or $N_=$. If M_5 contains an N_d then the arcs of then N_d induce a forbidden $K_{1,3}$ in G'. If M_5 contains an N_X, then this N_X exists also in M_4, and by I_X, it exists also in M_3 and M_2. By Lemma 1 we get that G' is not a proper circular-arc graph. If M_5 contains an $N_=$ then this $N_=$ exists also in M_4. By Lemma 2 we get that G' is not a proper circular-arc graph. □

We use Algorithm 4 to check if M_5 is a proper circular-arc model. If M_5 is not a proper circular-arc model then we get two arcs x and z such that x contains z. We traverse M_5 to the left from $\ell(z)$ until we find a right endpoint $r(u)$ such that u is disjoint from z. If u double overlaps x then M_5 has an $N_=$. Otherwise we find w to the right of z, symmetrically to u. The four arcs x, z, u and w forms an N_d or an N_X. By the proof of Theorem 1 (in Lemma 1 and Lemma 2), we can find a forbidden induced subgraph in G using these arcs.

Last, we discuss the case where G' is a proper interval graph. In this case, the model M does not have to be an interval model. However, the model M_5 which we produce is a proper interval model. Assume otherwise, and let c be the minimum number of arcs required to cover the circle in M_5. By Lemma 2, we know that $c > 2$. If $c > 3$, then G' contains an induced cycle of size c and so G' is not a proper interval graph [14]. If $c = 3$, then since there are no universal arcs in M_5, it contains either an induced cycle of length 4 or an induced H_1, both are not a proper interval graph [14]. When the model M is an interval model, then Algorithm 2 by itself is enough to produce a proper interval model, since N_X and $N_=$ configurations are not possible.

6 Universal Arcs

In the previous section, we showed how to find a proper circular-arc model for a circular-arc model which has no universal arc. In this section we show how to embed a universal arc into a proper circular-arc model. It is enough to show the embedding of a single universal arc, since the set universal arcs is a clique module. We begin with the model M_5 which is a proper circular-arc model of G'. We denote the universal arc which we embed by u.

First, we remove from M_5 any arc which can contain another arc. We can detect these arcs in M_4 and using Functions on Arcs, since M_4 keeps I_C. Let M_W be the model we get by removing the arcs from M_5 and let W be the set of arcs in this model.

Assume that we embed u into M_W such that $\ell(u)$ is immediately to the right of the endpoint e_ℓ and $r(u)$ is immediately to the left of e_r. Then, we can place the endpoints of u next to the same endpoints also in M_5. This way, u will intersect every arc of M_5, since every such arc is either in W or contains an arc of W. Moreover, u does not contain any arc in M_5, since u does not contain any arc in M_W. Therefore, it is enough to show how to embed u into M_W.

We index the arcs of $W = \{w_0, w_1, \ldots, w_{p-1}\}$ according to the order of their left endpoints, starting at an arbitrary endpoint $\ell(w_0)$ and going right. Arithmetic on the subscripts of arcs of W is modulo p where $p = |W|$.

If $p = 0$ then G is made only from universal arcs, constructing a proper circular-arc model is then trivial. If $p = 1$ then there is a single arc w_0, which every arc contain, in this case w_0 is a universal arc, but this cannot happen since we removed universal arcs from G'. So, we assume $p \geq 2$.

The following properties of the model M_W were given by Golumbic and Hammer [3]. The model M_W is a cyclic list of alternating left and right endpoints, since if endpoints of two arcs share the same block then one of the arcs can contain the other. The endpoint $\ell(w_0)$ is to the left of some right endpoint $r(w_{p-k})$ in M_W. Because no arc can contain another arc in M_W and because every left endpoint is to the left of a right endpoint, we get that $\ell(w_i)$ is to the left of $r(w_{i-k})$ for every $w_i \in W$. Note that if $k = 0$ then W is an independent set.

If $k \geq \frac{p-1}{2}$ then every arc of W intersects any other arc. This contradicts the fact that G' does not contain universal arcs.

If $k \leq \frac{p-3}{2}$ then G' contains $\overline{C_q}$ for some odd $3 \leq q \leq p$. We can find this complement of a cycle in $O(n)$ time, and together with u we get $\overline{C_q^*}$ as a forbidden induced subgraph.

We are left with the case where $k = \frac{p-2}{2}$. In this case, we put $\ell(u)$ between $\ell(w_0)$ and $r(w_{p-k})$ and $r(u)$ between $r(w_1)$ and $\ell(w_{k+1})$. In this place, the arc u crosses every arc from $w_{p-k} = w_{(p/2)+1}$ to $w_{k+1} = w_{p/2}$ which are all arcs of W.

Finally, we put $\ell(u)$ to the right of $\ell(w_0)$ also in M_5 and $r(u)$ to the left of $r(w_1)$. We also place back the arcs of clique modules that we removed earlier, including the universal arcs other than u. The resulting model, which we denote by M_6, is a proper circular-arc model of G.

Last, we discuss the case where G is a proper interval graph. As we noted above, M_5 is a proper interval graph in this case. Therefore, the set W is an independent set, and $k = 0$. If $p > 2$ then G is not an interval graph since $\{u\} \cup W$ contains $K_{1,3}$. If $p = 2$, then we can choose w_0 such that the circle is not covered between $r(w_1)$ and $\ell(w_0)$ in M_5. This way, the same part of the circle is not covered also in M_6, and therefore it is a proper interval model.

References

1. Deng, X., Hell, P., Huang, J.: Linear-Time Representation Algorithms for Proper Circular-Arc Graphs and Proper Interval Graphs. SIAM J. Comput. 25, 390–403 (1996)
2. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs, 2nd edn. Annals of Discrete Mathematics, vol. 57. Elsevier, Amsterdam (2004)
3. Golumbic, M.C., Hammer, P.L.: Stability in Circular Arc Graphs. J. Algorithms 9, 314–320 (1988)
4. Hsu, W.L.: $O(mn)$ Algorithms for the Recognition and Isomorphism Problems on Circular-Arc Graphs. SIAM J. Comput. 24, 411–439 (1995)
5. Hsu, W.L., Tsai, K.H.: Linear Time Algorithms on Circular-Arc Graphs. Info. Proc. Lett. 40, 123–129 (1991)
6. Joeris, B.L., Lin, M.C., McConnell, R.M., Spinrad, J.P., Szwarcfiter, J.L.: Linear Time Recognition of Helly Circular-Arc Models and Graphs (submitted)
7. Kaplan, H., Nussbaum, Y.: Certifying Algorithms for Recognizing Proper Circular-Arc Graphs and Unit Circular-Arc Graphs. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 289–300. Springer, Heidelberg (2006)
8. Kaplan, H., Nussbaum, Y.: A Simpler Linear-Time Recognition of Circular-Arc Graphs. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 41–52. Springer, Heidelberg (2006)
9. Lin, M.C., Szwarcfiter, J.L.: Efficient Construction of Unit Circular-Arc Models. In: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, pp. 309–315. ACM Press, New York (2006)
10. Lin, M.C., Soulignac, F.J., Szwarcfiter, J.L.: Proper Helly Circular-Arc Graphs. In: Brandstädt, A., Kratsch, D., Müller, H. (eds.) WG 2007. LNCS, vol. 4769, pp. 248–257. Springer, Heidelberg (2007)
11. Manacher, G.K., Mankus, T.A.: A Simple Linear Time Algorithm for Finding a Maximum Independent Set of Circular Arcs Using Intervals Alone. Networks 39, 68–72 (2002)
12. Masuda, S., Nakajima, K.: An Optimal Algorithm for Finding a Maximum Independent Set of a Circular-Arc Graph. SIAM J. Comput. 17, 41–52 (1988)
13. McConnell, R.M.: Linear-Time Recognition of Circular-Arc Graphs. Algorithmica 37, 93–147 (2003)
14. Roberts, F.S.: Indifference Graphs. In: Harary, F. (ed.) Proof Techniques in Graph Theory: Proceedings of the Second Ann Arbor Conference, pp. 139–146. Academic Press, London (1969)
15. Shih, W.K., Hsu, W.L.: An $O(n^{1.5})$ Algorithm to Color Proper Circular Arcs. Discrete Applied Mathematics 25, 321–323 (1989)
16. Spinrad, J.P.: Efficient Graph Representations. In: Fields Institute Monographs, vol. 19. American Mathematical Society, Providence (2003)
17. Tucker, A.: Structure Theorems for Some Classes of Circular-Arc Graphs. Discrete Mathematics 7, 167–195 (1974)
18. Tucker, A.: An Efficient Test for Circular-Arc Graphs. SIAM J. Comput. 9, 1–24 (1980)

Digraph Decompositions and Monotonicity in Digraph Searching

Stephan Kreutzer and Sebastian Ordyniak

Oxford University Computing Laboratory
{kreutzer,ordyniak}@comlab.ox.ac.uk

Abstract. We consider graph searching games on directed graphs and corresponding digraph decompositions. In particular we show that two important variants of these games – underlying DAG- and Kelly-decompositions – are non-monotone.

Furthermore, we explore the limits of algorithmic applicability of digraph decompositions and show that various natural candidates for problems, which potentially could benefit from digraphs having small "directed width", remain NP-complete even on almost acyclic graphs.

1 Introduction

The seminal work of Robertson and Seymour in their graph minor project has focused much attention on graph decompositions and associated measures of graph connectivity such as tree- or path-width. Aside from the interest in graph structure theory, these notions have also proved fruitful in the development of algorithms.

Intuitively, tree-width measures the similarity of a graph to a tree. Thus trees have tree-width one and graphs of small tree-width can be decomposed into parts with at most tree-width (plus one) vertices in a tree-like manner. Similarly to trees, tree-decompositions allow for recursive algorithms, whose running time is linear in the size of the underlying graph – but exponential in its width. Together with linear time parameterized algorithms for constructing tree-decompositions, this implies that a huge number of NP-complete problems become tractable on graph classes of bounded tree-width (see [8,7] for a survey on tree-width).

Graph Searching Games. Closely related to tree-width (and path-width) are so called graph searching games. Graph searching games are played by two players, the searcher and the fugitive, that simultaneously place tokens on the vertices of a graph. Whereas the fugitive has only one token and is restricted to move along paths in the graph that are not occupied by a searcher, the searcher controls an arbitrary amount of tokens and is free to move them anywhere on the graph. The aim of the searcher is to capture the fugitive, i.e. to force him into a position where he is not able to move any more. The minimum number of tokens needed by the searcher to capture the fugitive defines a natural graph invariant.

Within this general framework, there exist a range of variants defined by different abilities for both players. In particular one distinguishes between the *visible* and *invisible* variant. In the visible case, the searchers can see the fugitive and can adapt their strategy accordingly. In the invisible case, the fugitive's position is hidden from the searcher. Concerning the abilities of the fugitive one distinguishes between the so called *inert* variant, where the fugitive is only allowed to move if a searcher is placed on his current position, and the *dynamic* variant, where the fugitive can move in any step of the play. Combining this yields four main variants of which only three will be considered in this paper: visible and dynamic (*vis*), invisible and inert (*inert*), and invisible and dynamic (*invis*). The forth variant, visible and inert has recently been studied by Richerby and Thilikos [24].

An important concept in the theory of graph searching games is monotonicity. A game is *monotone*, if whenever k searchers can catch a fugitive on a graph they can do so without allowing the fugitive to re-occupy vertices. In general, restricting the searchers to monotone strategies may require additional searchers. LaPaugh [18] gave a first proof of monotonicity for a graph searching game. Since then, monotonicity has been intensely studied and a large number of monotonicity results have been established. See e.g. [3,6,9,12,13,18,19,27] or the survey [2] and references therein.

The importance of monotonicity in the context of graph decompositions results from the observation that many decompositions, like tree- and path-decompositions, can be defined in terms of monotone winning strategies for the searcher. Monotonicity for a game is often established through duality theorems for the underlying decomposition. Strategies for the fugitive provide the dual notion for the existence of a decomposition and yield natural obstructions for graphs having small decompositions. For example, the notion of a *bramble* is a natural formalisation of a winning strategy for the fugitive and provides an important obstruction for small tree-width (see [11,20]).

Despite the considerable interest and the large number of results in this field, two cases have so far resisted any attempts to solve the monotonicity problem – the graph searching game with a visible, dynamic fugitive and the game with an invisible, inert fugitive, both played on digraphs. It is these games that are closely related to DAG- and Kelly-decompositions [4,15]. In this paper,we solve the problems by showing that both games are non-monotone.

Digraph decompositions. In recent years, attempts have been made to generalise the notion of tree-decompositions and their algorithmic applications to directed graphs. Clearly, we can define the tree-width of a directed graph as the tree-width of the undirected graph we get by ignoring the direction of edges, a process which leads to some loss of information. This loss may be significant, if the algorithmic problems we are interested in are inherently directed. A good example is the problem of detecting Hamiltonian cycles. While we know that this can be solved easily on graphs with small tree-width, there are directed graphs with very simple connectivity structure which have large tree-width. Therefore, several proposals have been made to extend the notions of tree-decompositions and tree-width to directed graphs (see [3,5,15,16,23,25]). In particular, Reed [23] and Johnson, Robertson, Seymour, and Thomas [16] introduce the notion of *directed tree-width* and they show that Hamiltonicity can be solved for graphs of bounded directed tree-width in polynomial time.

Following this initial paper, several alternative definitions of directed graph decompositions have been proposed, with the aim of overcoming some shortcomings of the original definition. Berwanger, Dawar, Hunter and Kreutzer [4] and Obdržàlek [22] introduce the notion of DAG-width and Hunter and Kreutzer [15] introduce the notion of Kelly-width. All three proposals are supported by algorithmic applications and various equivalent characterisations in terms of obstructions, elimination orderings, and, in particular, variants of graph searching games on directed graphs. However, so far the algorithmic applications are restricted to few classes of problems, in particular the problem of finding disjoint paths, Hamiltonian-cycles and similar linkage problems, and certain problems in relation to combinatorial games (parity games) played on graphs motivated by the theory of computer-aided verification. Whereas the tree-width of undirected graphs has been employed to solve a huge number of problems on graphs of small tree-width, the algorithmic theory of directed graph decompositions is not nearly as rich.

It is an obvious question whether this is due to the fact that digraph decompositions are a relatively new field of research, where the fundamental machinery first needs to be developed, or whether this is due to a general limitation of this approach to algorithms on digraphs. In this paper we systematically explore the range of algorithmic applicability of digraph decompositions. For this, we look at typical NP-hard problems on graphs – as they can be found in [14], for instance – and identify those that are "suitable" for this approach, where by "suitable" we mean that the problems should be NP-hard in general but tractable on acyclic digraphs. The reason for the latter is that all digraph decompositions proposed so far measure in some way the similarity of a graph to being acyclic. In particular, acyclic graphs have small width in all of these measures. Hence, if a problem is already hard on acyclic digraphs, there is no point in studying the effect of digraph decompositions on this problem. We then identify representatives for the various types of "suitable" problems and ask whether they can be solved in polynomial time on graphs of small directed tree-, Kelly- or DAG-width or of small directed path-width.

The results we present in Section 4 show that the border for algorithmic applicability of digraph decompositions is rather tight. Essentially, as far as classical graph theoretical problems are concerned, disjoint paths and Hamiltonian-cycles can be detected efficiently on graphs of small directed tree-width, but all other problems we considered such as Minimum Equivalent Subgraph, Feedback Vertex Set (FVS), Feedback Arc Set, Graph Grundy Numbering, and several others are NP-complete even on graphs with a very low global connectivity and thus very low directed path or tree-width.

Organisation. The paper is organised as follows. In Section 2 we briefly recall basic notions from graph and game theory needed later. In Section 3 we give a formal description of graph searching games and present the first main result of this paper, the non-monotonicity of the two types of games mentioned above. In Section 4 we explore the algorithmic boundaries of the digraph decompositions known so far by showing NP-completeness for a number of problems on digraphs with bounded "width". We conclude and state some open problems in Section 5.

2 Preliminaries

We use standard notation from graph theory as can be found in, e.g., [10]. All graphs and directed graphs in this paper are finite and simple.

Let G be a (directed) graph. We denote the vertex set of G by $V(G)$ and the edge set of G by $E(G)$. For $X \subseteq V(G)$ we denote by $G[X]$ the subgraph of G induced by X and by $G \setminus X$ the subgraph of G induced by $V(G) \setminus X$. Similarly for $Y \subseteq E(G)$ we set $G \setminus Y$ to be the subgraph of G obtained by deleting all edges in Y.

Finally, if X is a set and $k \in \mathbb{N}$, we denote by $[X]^{\leq k}$ the set of all subsets of X of cardinality $\leq k$.

3 Graph Searching Games

In this section we show non-monotonicity of two important variants of graph searching on directed graphs, namely the variants underlying DAG- and Kelly-decompositions.

Graph searching games are played by two players – the searcher and the fugitive – placing tokens on the vertices of a graph. Whereas the fugitive has only one token and can only move along paths in the graph that are not blocked by a searcher, the searcher controls an arbitrary amount of tokens and is free to move them anywhere on the graph. That is, in any step of the play, the searchers can place new tokens or remove existing tokens from the board. A play begins with the fugitive choosing his initial position. In each step, the searchers first announce their intended move. The fugitive can then react to this by choosing his new position, as long as there is a path from his current to the new position that does not contain a vertex occupied by a searcher remaining on the board.

The aim of the searcher is to capture the fugitive, i.e. to force him into a position where he is not able to move any more. The minimum number of tokens needed by the searcher to capture the fugitive defines the graph invariant that we are interested in.

More formally, let G be an undirected graph. A position in the game is a pair (X, r), with $X \subseteq V(G)$ and $r \in V(G)$, and a play is a sequence of positions $((X_1, r_1), \ldots, (X_n, r_n))$, such that $X_1 = \emptyset$ and a move from one position to another is legal, if there is a path from r_i to r_{i+1} in $G \setminus (X_i \cap X_{i+1})$. A play is winning for the searcher if $r_n \in X_n$, otherwise it is winning for the fugitive.

Within this general framework, there exist a range of variants defined by different abilities for both players. In particular one distinguishes between the *visible* and *invisible* variant. In the visible case, the searchers can see the fugitive and can adapt their strategy accordingly. In the invisible case, the fugitive's position is hidden from the searcher. Concerning the abilities of the fugitive one distinguishes between the so called *inert* variant, where the fugitive is only allowed to move if a searcher is placed on his current position, and the *dynamic* variant, where the fugitive can move in any step of the play. Combining these variants yields four main variants of which only three will be considered in this paper: visible and dynamic (*vis*), invisible and inert (*inert*), and invisible and dynamic (*invis*).

We are mainly interested in the type of *strategies* the searcher can employ. One can easily verify that strategies in these games only depend on the current position in the game, i.e. are deterministic and positional. Basically, there exist two types of strategies for the searcher, depending on whether or not the fugitive is visible. In the visible case, the searcher can take the position of the fugitive into account and thus a strategy is a function $f : (X, r) \to X'$ assigning a new position X' to the searcher depending on the current position (X, r) in the game. In the invisible case, a strategy can simply be seen as a sequence of positions for the searcher. A strategy for the searcher is *winning* if all plays consistent with this strategy are, i.e. plays where the searcher always chooses the move defined by the strategy.

Let $P = ((X_1, r_1), \cdots, (X_n, r_n))$ be a play. We define the *search-width* of P, denoted by $\mathrm{sw}(P)$, to be $\mathrm{sw}(P) := \max_{1 \leq i \leq n} |X_i|$. Similarly, we define the search-width of a strategy to be the maximum search-width of all plays consistent with that strategy and the search-width of a graph G, to be $\mathrm{sw}(G) := \min\{\mathrm{sw}(f) : f$ is a winning strategy on $G\}$. Thus the search-width of a graph defines the graph invariant that we are interested in.

We are now ready to define two important properties of a graph searching game namely fugitive- and searcher-monotonicity. We say a play is *fugitive-monotone* if the fugitive is not able to reach a vertex from which he has previously been expelled. Thus in a fugitive-monotone play the set of vertices that the fugitive can reach is not increasing. A play is *searcher-monotone* if the searcher never reoccupies a previously vacated vertex. On undirected graphs, both notions are closely related: every searcher-monotone play that is winning for the searcher is also fugitive-monotone and for every fugitive-monotone play that is winning for the searcher there is a searcher-monotone play that uses the same amount of searchers. It is thus not always necessary to distinguish between both notions and we say a play is *monotone* if it is both fugitive- and searcher-monotone.

The notion of monotonicity directly applies to strategies for the searcher, so we say that a strategy is fugitive-monotone, searcher-monotone or just monotone, if all plays consistent with that strategy are. Let G be a graph. We define mon-sw$(G) := \min\{sw(f) : f$ is monotone and winning on $G\}$ and say that a game is *monotone*, if mon-sw$(G) := \mathrm{sw}(G)$ for all graphs G.

On undirected graphs all three variants we consider in this paper are monotone and satisfy:

1. vis-sw(G) = inert-sw$(G) = \mathrm{tw}(G) + 1$, for every graph G, where $\mathrm{tw}(G)$ denotes the tree-width of G (see [11] and [9]).
2. invis-sw$(G) = \mathrm{pw}(G)+1$, for every graph G, where $\mathrm{pw}(G)$ denotes the path-width of G (see [6]).

Depending on how one translates the notion of an undirected path to the directed setting, i.e. whether one regards it as a directed path from source to destination or as two directed paths, one in each direction, there are two natural variants of this game on directed graphs. We refer to the first variant, where the fugitive is allowed to move along (searcher-free) directed paths, as *reachability* variant (*reach*), and to the second one, where the fugitive is only allowed to move when there exist a path in each direction, as *strongly connected component* (*scc*) variant, as in this case the fugitive is only allowed to move in strongly connected components.

Combining these two ways of defined games on directed graphs with the variants discussed for the undirected setting yields a number of interesting games on directed graphs of which the following have been discussed in literature: strongly connected component, visible and dynamic (*scc-vis*); reachability, visible and dynamic (*reach-vis*); reachability, invisible and dynamic (*reach-invis*); and reachability, invisible and inert (*reach-inert*). We briefly relate these games to the corresponding digraph decompositions and recall what is known about monotonicity.

scc, visible, and dynamic. This variant is closely related to directed tree-width as it is known that scc-vis-sw$(D) - 1 \leq dtw(D) \leq 3 \cdot$ scc-vis-sw$(D) + 5$, for every digraph D, where dtw(D) is the directed tree-width as defined in [16]. It has been shown to be neither fugitive- nor searcher-monotone [1,16]. However, although not explicitly stated, [16] gives an upper bound for the monotonicity costs with respect to fugitive-monotonicity. It remains an interesting open question whether this holds for the searcher-monotone variant as well.

reachability, invisible, and dynamic. This variant defines directed path-width and has been shown to be monotone in [3].

reachability, visible, and dynamic. The monotone version of this variant defines DAG-width [4,22]. We therefore refer to these games as *DAG-games* and write dag-sw(D) and mon-dag-sw(D) for the non-monotone and monotone search-width of a graph D, with respect to this variant.

reachability, invisible, and inert. The monotone version of this variant defines Kelly-width [15]. We therefore refer to these games as *Kelly-games* and write kelly-sw(D) and mon-kelly-sw(D) for the non-monotone and monotone search-width of a graph D, with respect to this variant.

We are now ready to state our main results of this section, proving that DAG- and Kelly-Games are non-monotone.

3.1 Non-monotonicity of DAG-Games

Theorem 3.1. *For every $p \geq 2$ there exists a digraph D_p with* mon-dag-sw$(D_p) = 4p - 2$ *and* dag-sw$(D_p) = 3p - 1$.

proof. A schematic overview of D_p is given in Figure 1. The graph consists of three main parts with $2p - 1$ vertices each. C_0 and C_2 are cliques on $2p - 1$ vertices, C_1^2 is a clique on $p - 1$ vertices and C_1^1 forms an independent set having p vertices. A directed

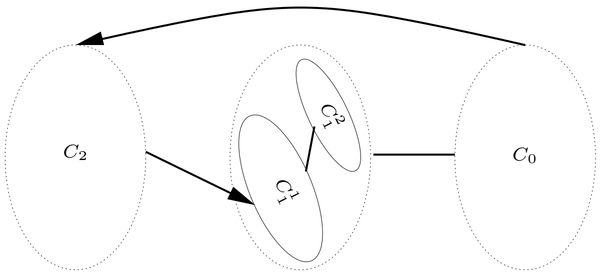


Fig. 1. The graph D_p with dag-sw$(D_p) \neq$ mon-dag-sw(D_p)

edge between two parts A and B means that there are edges from every vertex in A to every vertex in B. Undirected edges mean that there are edges between A and B in both directions. Thus there are edges in both directions between C_1^1 and C_1^2, and between C_0 and $C_1^1 \cup C_1^2$. Furthermore there are edges from C_0 to C_2, and edges from C_2 to C_1^1.

It is easy to see that dag-sw$(D_p) \geq 3p-1$ since the vertices in $C_0 \cup C_1^2$ together with a vertex of C_1^1 form a clique of size $3p-1$. To show that dag-sw$(D_p) \leq 3p-1$ consider the following strategy for $3p-1$ searchers on D_p. In the first move the searchers occupy $C_0 \cup C_1^1$. If the fugitive plays to C_2 the searchers capture him by playing on $C_1^1 \cup C_2$. Otherwise, if the fugitive plays to C_1^2 the searchers move to $C_0 \cup C_1^2$. Now the fugitive has to be on a vertex $v \in C_1^1$. Since the vertices in C_1^1 form an independent set the fugitive is now captured by playing to $\{v\} \cup C_1^2 \cup C_0$.

It remains to show that mon-dag-sw$(D_p) = 4p-2$. It is easy to see that $4p-2$ searchers can capture the fugitive on D_p by playing $C_0 \cup C_2$ and then $C_0 \cup C_1^1 \cup C_1^2$. To show that mon-dag-sw$(D_p) \geq 4p-2$ we give a strategy for the fugitive against $4p-3$ searchers playing monotonously on D_p.

First the fugitive stays in C_0 until the searchers occupy all vertices of C_0. There are two cases to consider.

1. The searchers occupy (at least) $C_0 \cup C_1^1$. In this case there is a vertex $v \in C_1^2$ which is not occupied by a searcher and which the fugitive can reach from his current position in C_0. Since every $v \in C_1^2$ has an edge to every other vertex in $C_0 \cup C_1^1 \cup C_1^2$ the searcher cannot capture the fugitive monotonously with less than $4p-2$ searchers.
2. The searchers occupy (at least) C_0 and there is at least one vertex in C_1^1 which is not occupied by a searcher. Then there exists a vertex $v \in C_2$ which is not occupied by a searcher and which the fugitive can reach from his current position in C_0. Since from every vertex in C_2 there is a path to every other vertex in the graph (as long as there is at least one vertex in C_1^1 not occupied by a searcher) the fugitive can stay in C_2 until the searchers occupy all vertices in C_1^1. And if they do the fugitive can move to a vertex in C_1^2 and play as in the first case. $\square$

3.2 Non-monotonicity of Kelly-Games

We now consider Kelly-games. Recall that in a Kelly-game, the fugitive is invisible. Hence, a strategy must be independent of the current position of the fugitive. We can therefore represent a searcher-strategy in a digraph D by a sequence $(X_1, \dots, X_n)$ of searcher-positions. Furthermore fugitive-monotone strategies can simply be given by a sequence of vertices $(v_1, \dots, v_{|D|})$, reflecting the order in which the vertices become cleared by the searcher. Note also that since Kelly-games are also inert, the notion of searcher-monotonicity cannot be applied.

Theorem 3.2. *For every $p \geq 2$ there exists a digraph D_p with* kelly-sw$(D_p) = 6p$ *and* mon-kelly-sw$(D_p) = 7p$.

proof. A schematic overview of D_p is given in Figure 2. The graph consists of five cliques with $|C_0| = p$, $|C_2| = |C_1| = |X_1| = 2p$, $|X_2| = 3p$. An edge between two

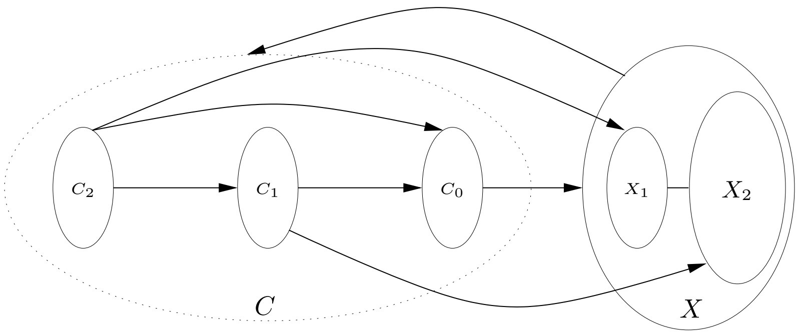


Fig. 2. The graph D_p with kelly-sw$(D_p) \neq$ mon-kelly-sw(D_p)

parts A and B means that there are edges from every vertex in A to every vertex in B, where again an undirected edge between A and B means that there are edges in D_p in both directions.

The following strategies show that mon-kelly-sw$(D_p) \leq 7p$ and kelly-sw$(D_p) \leq 6p$. For the monotone game we use the strategy $(X \cup C_0, X_2 \cup C_0 \cup C_1, X_1 \cup C_0 \cup C_1, X_1 \cup C)$, i.e. the searchers first occupy all of X and C_0, then proceed to $X_2 \cup C_0 \cup C_1$, and $X_1 \cup C_0 \cup C_1$ and finally move to $X_1 \cup C$. For the non-monotone case we use $(X \cup C_0, X_2 \cup C_0 \cup C_1, X_1 \cup C_1, X_1 \cup C_1 \cup C_2, X, X \cup C_0)$.

To see that kelly-sw$(D_p) \geq 6p$ note that $C_0 \cup X$ is a clique of size $6p$. It remains to show that mon-kelly-sw$(D_p) \geq 7p$. Suppose mon-kelly-sw$(D_p) < 7p$ and let $S = (v_1, \cdots, v_{|V(D_p)|})$ be a searcher-strategy witnessing this. For each part $Y \in \{C_0, C_1, C_2, X_1, X_2, C, X\}$ of D_p let $I(Y)$ be the greatest index of a vertex in Y, i.e. $v_{I(Y)}$ is the last vertex of Y which is searched by S. Then the following statements hold:

1. $I(X) < I(C_1)$ and $I(X) < I(C_2)$. For the sake of contradiction, suppose $I(X) > I(C_1)$ and let $v = v_{I(X)}$. Hence, when the searchers clear v, they have already cleared all vertices in X other than v and all vertices in C_1. As v has edges to every other vertex in $C_1 \cup X$, the searchers need to occupy all of $(C_1 \cup X) \setminus \{v\}$ before they can place a token on v. But this requires $7p$ searchers.
   The case of $I(X) < I(C_2)$ is analogous.
2. $I(C_0) < I(C_1)$. Again, assume the contrary, i.e. $I(C_0) > I(C_1)$. Hence, when clearing $v_{I(C_1)}$ there is a free vertex $v \in C_0$ through which the fugitive can reach all of X. As $I(X) < I(C_1)$, the searchers needs to occupy at least $(X \cup C_1) \setminus \{v_{I(C_1)}\}$ before clearing $v_{I(C_1)}$, which yields the contradiction.
3. $I(C_1) < I(C_2)$. With a similar reasoning as before we obtain that otherwise the searchers have to occupy $X \cup C_2$ when searching $v_{I(C_2)}$, using $7p$ searchers.

The statements (1)-(3) imply $I(X) < I(C_0) < I(C_1) < I(C_2)$ but now the searcher needs to occupy $|C_2 \cup C_1 \cup C_0 \cup X_1| = 7p$ vertices in order to search $v_{I(C_2)}$. So S uses at least $7p$ searchers. $\square$

4 Limits of Algorithmic Applications

In [16] it has been shown that the k-disjoint path problem as well as related problems, including the Hamiltonian-path problem, are solvable in polynomial time on graphs of bounded directed tree-width. However, up to now only few other problems are known to be solvable with the help of digraph decompositions, a further example being parity games, which are tractable on graphs of bounded DAG- and Kelly-width [4,15]. As directed tree-width is the most general of these width-measures, tractability results for directed tree-width directly extend to all other measures. The converse is not true, for example it is not known whether parity games are tractable on graphs of bounded directed tree-width.

In this section we explore the algorithmic boundaries of the digraph measures introduced so far. In our analysis we focus on NP-complete problems that are explicitly directed. All analysed problems are solvable in polynomial time on digraphs whose underlying undirected graph has bounded tree-width – but as mentioned in the introduction, tree-width is not a good measure for the global connectivity of a digraph. Furthermore, we discard problems that are not tractable on acyclic graphs, as all measures defined so far are bounded on acyclic graphs. As representatives for various types of the remaining problems, we have considered the following problems: Minimum Equivalent Subgraph, Directed Feedback Vertex / Arc Set, Graph Grundy Numbering, and Kernel.

It turns out that all of these problems remain NP-complete even on digraphs that have very low global connectivity, i.e. digraphs that can be decomposed into components of constant size just by removing a small number of vertices. In particular, these graphs have low width with respect to all digraph decompositions defined so far, i.e. small directed path width, small DAG-, Kelly-, and directed tree-width, small Entanglement and D-width. In order to state the proofs in their most general way we define the class $\mathcal{CONN}_i^j$ as follows:

Definition 4.1. *Let i and j be integers. We define $\mathcal{CONN}_i^j$ to be the class of digraphs, such that for every digraph $D \in \mathcal{CONN}_i^j$ there exists a vertex set $X \subseteq V(D)$ with $|X| \leq j$, such that every component in $D \setminus X$ has at most i vertices.*

As mentioned above it is easy to see that:

Proposition 4.2. *For all i and j the class $\mathcal{CONN}_i^j$ has bounded directed path-width, directed tree-width, D-width, DAG-width, Kelly-width and Entanglement* [1].

4.1 Minimum Equivalent Subgraph

The *Minimum Equivalent Subgraph (MES)*-problem is the problem to compute in a given digraph D an edge-minimal subgraph $D' \subseteq D$ that preserves reachability in D.

Definition 4.3. *Let D be a digraph and $k \in \mathbb{N}$.* MES *is the problem to decide, if there exists a set $E' \subseteq E(D)$ with $|E'| \leq k$, such that the digraph $D' = (V(D), E')$ contains a path between two vertices if, and only if, such a path exists in D, i.e. D and D' have the same transitive closure.*

[1] An upper bound for all given width parameters is $i + j$.

MES is NP-complete for arbitrary digraphs (see [14]), but is known to be solvable in polynomial time for acyclic and undirected graphs. In [21] it is also shown that it suffices to consider MES on connected digraphs. There MES is equivalent to a generalisation of the directed hamiltonian cycle problem, the so-called round-trip-problem, in which vertices can be used more than once. This is particularly interesting because the directed hamiltonian cycle problem is a special case of the k-linkage problem, which can be solved in polynomial time on digraphs of bounded directed tree-width.

Definition 4.4. *Let D be a connected digraph. A round-trip $R = (v_1, \cdots, v_k, v_1)$ is a sequence of $k+1$ vertices of D, such that $(v_i, v_{i+1}) \in E(D)$ and R visits every vertex of D at least once. The size of R equals the number of distinct edges used by R.*

Lemma 4.5. *[21] Let D be a connected digraph and k a natural number. Then D has a MES of size less than k if, and only if, D has a round-trip of size less than k.*

The NP-completeness of MES for digraphs in $\mathcal{CONN}_3^1$ follows from a reduction of 3-SAT to the problem of finding a minimum round-trip in a connected digraph. Due to space restrictions the proof is deferred to the appendix.

Theorem 4.6. *The MES-problem is NP-complete even when restricted to digraphs in $\mathcal{CONN}_3^1$.*

4.2 Feedback Vertex Set / Feedback Arc Set

The *Feedback Vertex/Arc Set (FVS/FAS)*-problem is the problem to find a minimum set of vertices (edges) in a digraph D, whose removal leaves D acyclic. Both problems are known to be NP-complete on arbitrary digraphs (see [17]). Trivially both problems become efficiently solvable on acyclic graphs.

We prove the NP-completeness of FVS/FAS on digraphs in $\mathcal{CONN}_4^1$ respectively $\mathcal{CONN}_8^2$ by reducing to it a special variant of 3-SAT namely 3-SAT-2, which we introduce now.

Definition 4.7. *3-SAT-2 is the variant of 3-SAT, so that every literal is used in at most two clauses.*

3-SAT-2 is NP-complete. As before the proofs can be found in the appendix.

Theorem 4.8. *FVS respectively FAS are NP-complete even when restricted to digraphs in $\mathcal{CONN}_4^1$ respectively $\mathcal{CONN}_8^2$.*

4.3 Graph Grundy Numbering and Kernel

Definition 4.9. Graph Grundy Numbering *is the problem to decide for a digraph D, if there exists a function $f : V(D) \to \mathbb{N}$, such that for all $v \in V(D)$, $f(v)$ is the smallest natural number not contained in $\{f(u) : u \in V(D), (v, u) \in E(D)\}$.*

Definition 4.10. Kernel *is the problem to decide in a digraph D, if there exists $V' \subseteq V(D)$, such that*

1. *there is no edge between two vertices in V', i.e. V' is an independent set.*
2. *for every $v \in V(D) \setminus V'$ there exists a $u \in V'$ with $(v, u) \in E(D)$.*

Observe, that on undirected graphs the maximisation version of Kernel is the *Independent Set*-problem, whereas the minimisation version of Graph Grundy Numbering equals Vertex-Colouring. In contrast to the undirected case, where every graph has an Independent Set and a Vertex Colouring, not every digraph has a Kernel or a Graph Grundy Numbering and it is already NP-complete to decide whether a Kernel or a Graph Grundy Numbering do exist [26]. A simple example of a digraph that neither has a Graph Grundy Numbering nor a Kernel is the directed cycle with three vertices. Nevertheless it is easy to see that Kernel and Graph Grundy Numbering are trivially solvable on acyclic graphs. We are now ready to prove the NP-completeness for Graph Grundy Numbering on digraphs in $\mathcal{CONN}_4^0$ - again the proofs are deferred to the appendix.

Theorem 4.11. *Graph Grundy Numbering and Kernel are NP-complete even when restricted to digraphs in $\mathcal{CONN}_4^0$.*

5 Conclusion and Open Problems

In this paper we considered graph searching games on directed graphs and established non-monotonicity for two important variants of these games. Our examples show that the monotonicity costs for these games cannot be bounded by an additive term, i.e. for any k there are digraphs where at least k additional searchers are required to catch a robber with a monotone strategy. However, so far there is no upper bound for the monotonicity costs involved. It is conceivable that there is a constant $c \in \mathbb{N}$ such that whenever n searchers suffice to catch a robber on a digraph D in any of the two variants, than $c \cdot n$ searchers suffice for a monotone strategy. This, however, is left as an open problem.

A different trait we explored in this paper are the limits of an algorithmic theory based on directed graph decompositions. We showed that while there are interesting and important examples for natural problems that become tractable on digraphs of small width, many other natural problems remain NP-complete even if the digraphs have very low global connectivity.

References

1. Adler, I.: Directed tree-width examples. Journal Combinarial Theory Series B 97(5), 718–725 (2007)
2. Alspach, B.: Searching and sweeping graphs: A brief survey. In: COMBINATORICS 2004 (2004)
3. Barát, J.: Directed path-width and monotonicity in digraph searching. Graphs and Combinatorics 22(2), 161–172 (2006)
4. Berwanger, D., Dawar, A., Hunter, P., Kreutzer, S.: DAG-width and parity games. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 524–536. Springer, Heidelberg (2006)

5. Berwanger, D., Grädel, E.: Entanglement – A measure for the complexity of directed graphs with applications to logic and games. In: Baader, F., Voronkov, A. (eds.) LPAR 2004. LNCS, vol. 3452, pp. 209–223. Springer, Heidelberg (2005)
6. Bienstock, D., Seymour, P.: Monotonicity in graph searching. Journal of Algorithms 12, 239–245 (1991)
7. Bodlaender, H.L.: Treewidth: Algorithmic techniques and results. In: Privara, I., Ružička, P. (eds.) MFCS 1997. LNCS, vol. 1295, pp. 19–36. Springer, Heidelberg (1997)
8. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theoretical Computer Science 209, 1–45 (1998)
9. Dendris, N.D., Kirousis, L.M., Thilikos, D.M.: Fugitive-search games on graphs and related parameters. Theorectical Computer Science 172(1-2), 233–254 (1997)
10. Diestel, R.: Graph theory, 3rd edn. Graduate Texts in Mathematics, vol. 173. Springer, Berlin (2005)
11. Seymour, P.D., Thomas, R.: Graph searching, and a min-max theorem for tree-width. Journal of Combinatorial Theory, Series B 58, 22–33 (1993)
12. Dyer, D.: Sweeping Graphs and Digraphs. Ph.D thesis, Simon Fraser University (2004)
13. Fomin, F.V., Thilikos, D.M.: On the monotonicity of games generated by symmetric submodular functions. In: Brandstädt, A., Le, V.B. (eds.) WG 2001. LNCS, vol. 2204, p. 177. Springer, Heidelberg (2001)
14. Garey, M.R., Johnson, D.S.: Computers and Intractibility. W. H. Freeman and Company, New York (1979)
15. Hunter, P., Kreutzer, S.: Digraph measures: Kelly decompositions, games, and orderings. In: Proceedings of the 18th ACM-SIAM Symp. on Discrete Algorithms (SODA), pp. 637–644 (2007)
16. Johnson, T., Robertson, N., Seymour, P.D., Thomas, R.: Directed tree-width. J. Combin. Theory Ser. B 82(1), 138–154 (2001)
17. Karp, R.M.: Complexity of Computer Science. Plenum Press, New York (1972)
18. LaPaugh, A.S.: Recontamination does not help to search a graph. Journal of the ACM 40, 224–254 (1993)
19. Mazoit, F., Nisse, N.: Monotonicity property of non-deterministic graph searching. In: Brandstädt, A., Kratsch, D., Müller, H. (eds.) WG 2007. LNCS, vol. 4769, pp. 33–44. Springer, Heidelberg (2007)
20. Mazoit, F., Nisse, N.: Monotonicity of non-deterministic graph searching. Theor. Comput. Sci. 399(3), 169–178 (2008)
21. Moyles, D.M., Thompson, G.L.: An algorithm for finding a minimum equivalent graph of a digraph. Journal of the ACM 16(3), 455–460 (1969)
22. Obdržálek, J.: DAG-width: connectivity measure for directed graphs. In: Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 814–821 (2006)
23. Reed, B.: Introducing directed tree width. In: 6th Twente Workshop on Graphs and Combinatorial Optimization (Enschede, 1999). Electronic Notes in Discrete Mathematics, vol. 3, 8 p. Elsevier, Amsterdam (1999) (electronic)
24. Richerby, D., Thilikos, D.: Searching for a Visible, Lazy Fugitive. In: 34th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2008) (2008)
25. Safari, M.A.: D-width: a more natural measure for directed tree width. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 745–756. Springer, Heidelberg (2005)
26. van Leeuwen, J.: Having a grundy-numbering is NP-complete. Technical report, Pennsylvania State University (1976)
27. Yang, B., Cao, Y.: Digraph strong searching: Monotonicity and complexity. In: Kao, M.-Y., Li, X.-Y. (eds.) AAIM 2007. LNCS, vol. 4508, pp. 37–46. Springer, Heidelberg (2007)

Searching for a Visible, Lazy Fugitive

David Richerby[1,⋆] and Dimitrios M. Thilikos[2,⋆⋆]

[1] School of Computing, University of Leeds, Leeds LS2 9JT, UK
richerby@comp.leeds.ac.uk
[2] Department of Mathematics, National and Kapodistrian University of Athens, Panepistimioupolis, GR157-84 Athens, Greece
sedthilk@math.uoa.gr

Abstract. Graph searching problems are described as games played on graphs, between a set of cops and a fugitive. Variants of the game restrict the abilities of the cops and the fugitive and the corresponding search numbers (the least number of cops that have a winning strategy) are related to several well-known parameters in graph theory. We study the case where the fugitive is visible (the cops' strategy can take into account his current position) and lazy (he moves only when the cops move to his position). Our results are stated and proven in a general setting where the fugitive's speed (i.e., the lengths of paths he can move along) can be unbounded or bounded by some constant. We give a min-max characterization of the corresponding parameters, which we show to be computable in polynomial time for fugitivess with unbounded speed and speed at most 3 and to be NP-complete for all other finite speeds. This is in contrast to the other standard versions of the game, where the parameters corresponding to fugitives with unbounded speed are NP-complete. Several consequences of our results are also discussed.

1 Introduction

Graph searching games are played between a group of cops and a fugitive, on the vertices and edges of a graph. The cops aim to capture the fugitive, while the fugitive tries to escape capture. The rules by which the players move lead to several variants of the game. While the definition and study of such games dates back to the late 1970s [18, 19], they have recently been studied widely, mainly due to numerous applications in security problems in networks [1, 4, 10, 11].

There are several basic variants of the game and we consider only those where the cops and the fugitive reside on the vertices of the graph. At any one time, the fugitive occupies some vertex of the graph but each cop, independently, may be either on a vertex of the graph or out of play. In *node search* games, the cops

⋆ Research carried out while the first author was a Post Doc of the Graduate Program in Logic, Algorithms, and Computation ($\mu\prod\lambda\forall$), at the Department of Mathematics of the National and Kapodistrian University of Athens. and funded by the European Social Fund and Greek National Resources — (ΕΠΕΑΕΚ II) PYTHAGORAS II.

⋆⋆ Supported by the FET pro-actives Integrated Project 15964 (AEOLUS).

are moved either by placing them on or removing them from vertices; in the more general setting of *mixed search*, a cop may, in addition, slide along an edge from the endpoint he occupies to the other, vacant, endpoint. In both variants, the fugitive moves along cop-free paths in the graph. The fugitive is captured if a cop moves to the vertex he occupies and he has no path along which to escape. If the fugitive is captured, the cops win; if he remains on the run forever, he wins. (We do not consider edge search, where the fugitive resides on edges of the graph, as this can be reduced to mixed search by standard techniques.)

Further variants of the game come from altering the properties of the fugitive. He may be either *visible* to the cops, in which case the cops may use the fugitive's current position to choose their moves, or *invisible*, in which case the cops do not know where he is and their moves may be specified in advance. He may also be *lazy*, in which case he moves only when a cop moves to his vertex, or *active*, in which case he may move at every round of the game.

Each variant of the game generates a graph parameter that is the minimum number of cops that have a winning strategy in a given graph. In this paper, we concentrate mostly on node search. For the visible–active and invisible–lazy cases, the node search number is known to be one greater than the treewidth of the graph; for the invisible–active case, it is one greater than the graph's pathwidth. Similar parameters can be defined for mixed search [5, 20, 23, 24]. The decision problems associated with these graph parameters are known to be NP-complete.

In this paper, we study the remaining case, where the fugitive is visible and lazy, which does not seem to have been considered before. Generalizing, we parameterize the game by the speed s of the fugitive, i.e., the maximum length of paths along which he may move. We write, respectively, $\mathbf{vlns}^s$ and $\mathbf{vlms}^s$ for the node- and mixed-search numbers for a fugitive with speed s, with $s = \infty$ denoting a fugitive with unbounded speed. Our main result is a min-max theorem for the two parameters, for any speed $s \in \mathbb{N} \cup \{\infty\}$. In particular, we give a characterization in terms of the existence of specific obstructing structures, which we call *hide-outs*, that guarantee an escape strategy for the fugitive.

We also introduce two hierarchies of graph parameters, defined in terms of layouts, which we write δ^s and δ^s_{m}. These parameters are equivalent, respectively, to $\mathbf{vlns}^s$ and $\mathbf{vlms}^s$. The min-max theorem implies that our search parameters, in the case of a fugitive with unbounded speed, can be computed in polynomial time, which is quite unexpected, since all other variants of the game discussed above lead to NP-complete parameters. The parameters can also be computed in polynomial time for fugitives with speed at most 3; for other finite speeds, they are NP-complete. The known results for fugitives of unbounded speed are summarized in Table 1.

δ^s is a natural generalization of the classical graph parameter of *degeneracy*, defined as $\delta^*(G) = \max\{\delta(H) \mid H \subseteq G\}$, where $\delta(H)$ is the minimum degree of H's vertices. It is known from folklore that $\delta^*(G) = \delta^1(G)$ [17,6] and $\delta^*(G) + 1$ is also known as the graph's *colouring number*, since there is an easy greedy algorithm that colours a graph G with that many colours [9].

Table 1. Graph node-search variants (for fugitives with unbounded speed), their corresponding graph parameters and their complexities

node search	Visible	Invisible
Lazy	$\delta^{\infty}+1$, in P [this paper]	$\mathbf{tw}+1$ [8], NP-c [2]
Active	$\mathbf{tw}+1$ [22], NP-c [2]	$\mathbf{pw}+1$ [5,14], NP-c [2]

We prove that each of $\mathbf{vlns}^s$ and $\mathbf{vlms}^s$ defines a nontrivial hierarchy of parameters: for any r and s with $3 \leqslant r < s < \infty$, there are graphs with $\mathbf{vlns}^r(G) < \mathbf{vlns}^s(G) < \mathbf{vlns}^{\infty}(G)$ and similarly for mixed search.

To give a lower bound for treewidth, Bodlaender, Koster and Wolle define the *contraction degeneracy* of a graph G to be $\delta\mathrm{C}(G)$, the maximum $\delta(H)$ over nontrivial minors H of G [15,25,6]. We extend contraction degeneracy by replacing the term $\delta(H)$ with $\delta^s(H)$ and show that the extension $\delta^{\infty}\mathrm{C}(G)$, where $\delta(H)$ is replaced with $\delta^{\infty}(H)$, approximates treewidth, in the sense that there is a function f such that, for all graphs, $\delta^{\infty}\mathrm{C}(G) \leqslant \mathbf{tw}(G) \leqslant f(\delta^{\infty}\mathrm{C}(G))$. This improves on contraction degeneracy, which is known to provide only a lower bound for treewidth.

The remainder of the paper is organized as follows. Section 2 gives basic definitions. The searching model for a visible, lazy fugitive is formally described in Section 3. Our main results appear in Section 4, where we define hide-outs and our generalization of graph degeneracy and state the min-max theorem. We also give the the corresponding algorithm and determine its complexity. The nontriviality of the hierarchies defined is shown in Section 5. In Section 6, we generalize contraction degeneracy and we make concluding remarks and present some open problems in Section 7.

2 Preliminaries

We write $\mathbb{N}$ for the set $\{1, 2, \ldots\}$ and $\mathbb{N}^+$ for $\mathbb{N} \cup \{\infty\}$. Given a set S and an object x, we write $S + x$ and $S - x$ for $S \cup \{x\}$ and $S \setminus \{x\}$, respectively.

All graphs considered in this paper are finite, simple and undirected. To avoid trivial exceptions, we assume that all graphs contain at least one edge.

We write $V(G)$ and $E(G)$, respectively, for the vertex set and edge set of a graph G and xy for the undirected edge $\{x, y\}$. For $X \subseteq V(G)$, $G[X]$ is the subgraph of G induced by the vertices in X and, for $Y \subseteq E(G)$, $G - Y = (V(G), E(G) \setminus Y)$. Given a vertex $x \in V(G)$, we let $E_G(x)$ be the set of all edges of G incident with x. We write $G - x$ for the graph $G[V(G) - x]$ and $\delta(G)$ and $\Delta(G)$ for the minimum and maximum degree of G's vertices, respectively.

The operation of *dissolving* a vertex $x \in V(G)$ of degree two is the removal of x from G and the addition of an edge connecting its two former neighbours. A graph H is a *topological minor* of G if it can be made from some subgraph of G by dissolving vertices of degree two. A graph H is a *minor* of G if it can be made from a subgraph of G by contracting edges (i.e., identifying the two endpoints of the edge and deleting the resulting loop).

A *tree decomposition* of a graph G is a pair (X, T) where T is a tree and $X = \{X_i \mid i \in V(T)\}$ is a collection of subsets of $V(G)$ such that:

- $\bigcup_{i \in V(T)} X_i = V(G)$;
- for each edge $xy \in E(G)$, $\{x, y\} \subseteq X_i$ for some $i \in V(T)$; and
- for each $x \in V(G)$, $\{i \mid x \in X_i\}$ induces a connected subtree of T.

The *width* of a tree decomposition $(\{X_i \mid i \in V(T)\}, T)$ is defined to be $\max_{i \in V(T)} \{|X_i|\} - 1$ and the *treewidth* of a graph G is the minimum width over all tree decompositions of G. If we restrict the tree T to be a path, then we define the notions of *path decomposition* and *pathwidth*. We write $\mathbf{tw}(G)$ and $\mathbf{pw}(G)$, respectively, for the treewidth and pathwidth of a graph G.

3 The Searching Model

In this section, we define a model for the graph search game against a visible, lazy fugitive. The players have complete information about each other's position and may use this to decide their next move. The cops' goal is to capture the fugitive who tries, of course, to evade capture. Initially, there are no cops in the graph but, at any moment before his capture, the fugitive is on some vertex. The fugitive is *lazy*, in that he may move only when a cop is moved to his current vertex. When he moves, he does so with speed $s \in \mathbb{N}^+$; that is, he moves along cop-free paths of length at most s.

A play of the game consists of a sequence of rounds, with each round being composed of three parts, as follows.

Announcement. The cops announce their intended move to the fugitive. This can be: the *placement* of a cop on a vertex x, not currently occupied by a cop; the *removal* of a cop from an occupied vertex x; or *sliding* a cop from one end x of an edge xy to the other, which is initially not occupied.

Avoidance. If a cop is to be placed on or slid to the fugitive's current vertex, the latter may move along any cop-free path of length at most s. In the case of placement to x, that vertex is not considered blocked at this round; for sliding from x to y, the edge xy is considered blocked but the vertices x and y are not.

Realization. The cops carry out the announced action.

The fugitive is captured if a cop moves to his vertex and he has no move to escape. We may assume that the fugitive has full knowledge of the cops' strategy and will take the optimal decision towards avoiding capture. The fugitive is visible, so the cops' moves take his position into account and the game is interactive.

We denote the position of the fugitive in the graph at the ith round by a vertex $v_i \in V(G)$. Since, at any time, there is at most one cop on eacy vertex, we may represent the position of the cops after the ith move as a set $S_i \subseteq V(G)$.

We say that a finite or infinite sequence $S_0, S_1, \ldots$ of subsets of $V(G)$ is *consistent* if, for all $i \geqslant 0$, $G[S_i \vartriangle S_{i+1}]$ is either a single vertex or a two-clique.

Thus, the sequence corresponds to a sequence of cop positions in a play of the game and either $S_{i+1} = S_i + x$ for some $x \notin S_i$ (placement to x), $S_{i+1} = S_i - x$ for some $x \in S_i$ (removal from x) or $S_i \vartriangle S_{i+1}$ is an edge of G (sliding from the unique vertex in $S_i \setminus S_{i+1}$ to the unique vertex in $S_{i+1} \setminus S_i$).

Given two consecutive sets S and S' of a consistent sequence, we say that a path P in G is *(S, S')-avoiding* if its internal vertices avoid $S \cap S'$ and its edges avoid the edge $e = S \vartriangle S'$, in the case that $|e| = 2$.

Let $k \in \mathbb{N}$ and $s \in \mathbb{N}^+$. A *(k, s)-play* of the game on a graph G is a finite or countably infinite sequence of alternating vertex sets and vertices

$$\langle S_i, v_i \mid 0 \leqslant i < \kappa \rangle$$

for some $\kappa \in \mathbb{N}^+$, such that:

- $S_0 = \emptyset$;
- the sequence S_0, $S_1, \ldots$ is consistent;
- $|S_i| \leqslant k$ for all i; and
- for each i with $0 < i < \kappa$, either
  - $v_{i-1} \notin S_i$ and $v_i = v_{i-1}$ (the cops did not move to the fugitive's vertex so he did not move);
  - $v_{i-1} \in S_i$, there is an (S_{i-1}, S_i)-avoiding path of length at most s from v_{i-1} to v_i and $v_i \notin S_i$ (the cops moved to the fugitive's vertex and he ran along a cop-free path of length at most s); or
  - $i = \kappa - 1$, $v_i = v_{i-1} \in S_i$ and there are no (S_{i-1}, S_i)-avoiding paths from v_i (we are at the last move of a finite play, a cop has moved to the fugitive's vertex and he has no cop-free path on which to escape: the fugitive has been captured).

Each move made by the cops may depend both on their current position and that of the fugitive. A *(k, s)-strategy* is a function

$$\mu : V(G)^{[\leqslant k]} \times V(G) \to V(G)^{[\leqslant k]},$$

whose inputs are the position S of the cops and the position v of the fugitive and whose output is S', the new position of the cops, which is reached by a single placement, removal or sliding move and with $|S'| \leqslant k$. The strategy is to be used against a fugitive who has speed $s \in \mathbb{N}^+$. (A consequence of our min-max theorem is that we do not need to consider more general notions of strategy, where the cops' move may depend on the whole history of the game, in addition to the current position.)

Given a (k, s)-strategy μ, a *μ-play* is any (k, s)-play $\langle S_i, v_i : 0 \leqslant i < \kappa \rangle$ where $S_{i+1} = \mu(S_i, v_i)$ for all i. A strategy μ is said to be *winning* for the cops against a fugitive with speed s if every μ-play is finite (i.e., results in the capture of the fugitive). We define the *visible, lazy mixed search number* against a fugitive with speed s for a graph G to be

$$\mathbf{vlms}^s(G) = \min \{k \mid \text{ there is a winning } (k, s)\text{-strategy for } G\}\,.$$

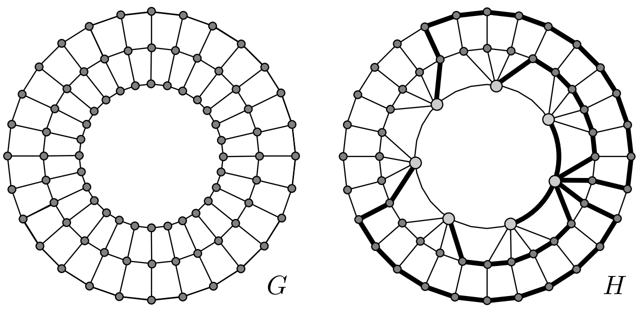


Fig. 1. The graph H is a minor of G, however $\mathbf{vlns}^{\infty}(G) = 4$, while $\mathbf{vlns}^{\infty}(H) = 7$. The vertices of degree six in H form a $(6, \infty)$-hide-out, as shown by the bold paths (see Section 4).

Applying the same definitions but allowing only placement and removal moves (i.e., $|S_i \vartriangle S_{i+1}| = 1$ for all i), gives the analogous *visible, lazy node search number* against a fugitive with speed s for a graph G, $\mathbf{vlns}^s(G)$.

The following lemma gives key properties of the defined parameters.

Lemma 1. *For any graphs G and H and any $s \in \mathbb{N}^+$,*

1. $\delta(G) + 1 \leqslant \mathbf{vlns}^s(G) \leqslant \Delta(G) + 1$ *and* $\delta(G) \leqslant \mathbf{vlms}^s(G) \leqslant \Delta(G)$*;*
2. $\mathbf{vlms}^s(G) \leqslant \mathbf{vlns}^s(G) \leqslant \mathbf{vlms}^s(G) + 1$*;*
3. *if $H \subseteq G$, then* $\mathbf{vlns}^s(H) \leqslant \mathbf{vlns}^s(G)$ *and* $\mathbf{vlms}^s(H) \leqslant \mathbf{vlms}^s(G)$*;*
4. *if H is a topological minor of G, then we have* $\mathbf{vlns}^{\infty}(H) \leqslant \mathbf{vlns}^{\infty}(G)$ *and* $\mathbf{vlms}^{\infty}(H) \leqslant \mathbf{vlms}^{\infty}(G)$*.*

Thus, $\mathbf{vlns}^{\infty}$ and $\mathbf{vlms}^{\infty}$ are closed under taking subgraphs and topological minors. However, they are not closed under taking minors, since every graph G is a minor of some graph H with $\Delta(H) \leqslant 3$. G may have arbitarily large search numbers but, by Lemma 1.1, $\mathbf{vlns}^{\infty}(H) \leqslant 4$ and $\mathbf{vlms}^{\infty}(H) \leqslant 3$. Lemma 1.4 cannot be extended to $\mathbf{vlns}^s$ or $\mathbf{vlms}^s$ for finite s. For $s \in \mathbb{N}$, there are graphs G with topological minors H such that $\mathbf{vlns}^s(H) > \mathbf{vlns}^s(G) = 3$ or $\mathbf{vlms}^s(H) > \mathbf{vlms}^s(G) = 2$: for example, take H to be any clique and replace the edges with independent $(s+1)$-paths to make G. (See also Figure 1.)

Variants of the above model have already appeared in the literature, for fugitives of unbounded speed. The version where the fugitive is visible and active is due to Seymour and Thomas [22], who show that the corresponding node-search number is $\mathbf{tw}(G) + 1$; the node-search number is the same for an invisible, lazy fugitive [8]. Finally, the version with an invisible, active fugitive was introduced by Kirousis and Papadimitriou [14] and studied further by Bienstock and Seymour [5]. In this case, the node-search number is $\mathbf{pw}(G) + 1$ [13, 14]. It follows immediately that determining any of the above search numbers is NP-complete and the same can also be shown for the mixed search variants of all of these games. However, we prove that the parameters $\mathbf{vlns}^{\infty}(G)$ and $\mathbf{vlms}^{\infty}(G)$ are polynomial-time computable. These results are summarized in Table 1.

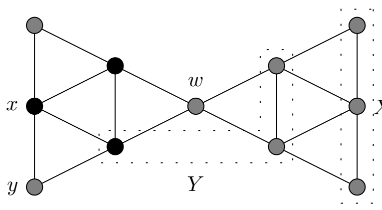


Fig. 2. An example graph with $\delta^s(G) = 2$ for any $s \in \mathbb{N}^+$

4 Degeneracy and Hide-Outs

Let G be a graph, $x \in V(G)$ and $X \subseteq V(G) - x$. For any $s \in \mathbb{N}^+$, we say that a set $A \subseteq V(G) - x$ is an (s, x, X)-*separator* if $G - A$ contains no path from x to X of length at most s. Define $\mathbf{sep}^s_G(x, X)$ to be the minimum size of any (s, x, X)-separator in G. For example, for the graph G in Figure 2, $\mathbf{sep}^\infty_G(x, X) = 1$, as w is a cut vertex. However, $\mathbf{sep}^s_G(x, X) = 0$ for any $s < 4$. Moreover, $\mathbf{sep}^3_G(x, Y) = 2$ and $\mathbf{sep}^2_G(x, Y) = \mathbf{sep}^1_G(x, Y) = 1$.

For $s \leqslant 3$ and $s = \infty$, $\mathbf{sep}^s_G(x, X)$ is the maximum cardinality of any set of x–X paths of length at most s that are vertex-disjoint apart from the common endpoint x. This is immediate from Menger's theorem in the case $s = \infty$ and can be shown by a simple modification to the proof of Lovász et al. [16, Theorem 3] for $s \leqslant 3$. On the other hand, for finite $s \geqslant 4$, there are graphs where $\mathbf{sep}^s(x, X)$ is greater than the maximum number of disjoint s-paths from x to X [16].

Lemma 2. *Let $s \in \mathbb{N}^+$. Given a graph G, $x \in V(G)$, $X \subseteq V(G) - x$ and $k \in \mathbb{N}$, the problem of determining whether $\mathbf{sep}^s_G(x, X) \leqslant k$ is in polynomial time for $s \in \{1, 2, 3, \infty\}$ and is* NP*-complete for all other s.*

Proof. The case $s = \infty$ is immediate from the max-flow min-cut theorem and the existence of polynomial-time algorithms for computing maximal flows. For finite values of s, the result follows from [12] and [16]. $\square$

In fact, for any fixed $s \geqslant 4$, it is NP-hard even to approximate $\mathbf{sep}^s(x, X)$ to within a constant factor of 1.1377 [3].

Let G be a graph, $k \in \mathbb{N}$ and $s \in \mathbb{N}^+$. A (k, s)-*layout* of G is an ordering $v_1, \ldots, v_n$ of $V(G)$ such that, for every $i \in \{1, \ldots, n\}$, $\mathbf{sep}^s_G(v_i, \{v_1, \ldots, v_{i-1}\}) \leqslant k$. We define the s-*degeneracy* of G to be $\delta^s(G)$, the least k for which G has a (k, s)-layout. Notice that the 1-degeneracy is identical to the classical graph-theoretic parameter of degeneracy.

A (k, s)-*hide-out* in a graph G is any set $R \subseteq V(G)$ such that, for every $x \in R$, $\mathbf{sep}^s_G(x, R - x) \geqslant k$. In Figure 1, the set of vertices in H of degree six is a $(6, \infty)$-hide-out; one of the sets of paths is shown in bold.

The s-degeneracy of a graph and the presence or absence of (k, s)-hide-outs within it are closely linked with the node search number for a visible, lazy fugitive. We now adapt these two concepts for mixed-search. First, set

$$\mathbf{msep}^s_G(x, X) = \min \{\mathbf{sep}^s_{G-Y}(x, X) \mid Y \subseteq E_G(x) \text{ and } |Y| \leqslant 1\}.$$

Algorithm 1. check-s-degen: check whether $\delta^s(G) \leqslant k$

Input: an n-vertex graph G and an integer $k \geqslant 1$.
Output: if $\delta^s(G) \leqslant k$, a (k, s)-layout; if not, a (k, s)-hide-out.

```
$S \leftarrow V(G)$.
for $i = n, \ldots, 1$,
    if there is $x \in S$ with $\mathbf{sep}^s_G(x, S - x) \leqslant k$ then $v_i \leftarrow x$;
    else output "$\delta^s(G) \geqslant k + 1$, witnessed by hide-out $S$."
    $S \leftarrow S - v_i$.
Output "$\delta^s(G) \leqslant k$, witnessed by layout $v_1, \ldots, v_n$."
```

That is, Y can be either empty, or a singleton containing one edge incident with x. Notice that $\mathbf{msep}^1_G(x, X) + 1 = \mathbf{sep}^1_G(x, X) = d_{G[X+x]}(x)$. Also, for $s \geqslant 2$, $\mathbf{msep}^s_G(x, X) \leqslant \mathbf{sep}^s_G(x, X) \leqslant \mathbf{msep}^s_G(x, X) + 1$.

For example, in graph G of Figure 2, we have $\mathbf{sep}^3_G(x, Y) = \mathbf{msep}^3_G(x, Y) = 2$, while $\mathbf{sep}^3_G(y, Y) = 2 = \mathbf{msep}^3_G(y, Y) + 1$.

Lemma 3. *Let $s \in \mathbb{N}^+$. Given a graph G, $x \in V(G)$, $X \subseteq V(G) - x$ and $k \in \mathbb{N}$, the problem of determining whether $\mathbf{msep}^s_G(x, X) \leqslant k$ is in polynomial time for $s \in \{1, 2, 3, \infty\}$ and is* NP*-complete for all other s.*

Proof (sketch). Let z be a new vertex, that does not appear in G. It can be shown that $\mathbf{msep}^s_{G+xz}(x, X + z) = \mathbf{sep}^s_G(x, X)$ and the result is then immediate from Lemma 2. □

Let G be a graph, $k \in \mathbb{N}$ and $s \in \mathbb{N}^+$. A *mixed (k, s)-layout* is an ordering $v_1, \ldots, v_n$ of $V(G)$ such that $\mathbf{msep}^s_G(v_i, \{v_1, \ldots, v_{i-1}\}) \leqslant k$ for every $i \in \{1, \ldots, n\}$. Let the *mixed s-degeneracy* of G be $\delta^s_{\mathrm{m}}(G)$, the least k for which G has a mixed (k, s)-layout. In Figure 2, $\delta^1_{\mathrm{m}}(G) = 1$ and $\delta^s_{\mathrm{m}}(G) = 2$ for $s \geqslant 2$.

We define a *mixed (k, s)-hide-out* in a graph G to be any set $R \subseteq V(G)$ such that for every $x \in R$, $\mathbf{msep}^s_G(x, R - x) \geqslant k$. In the graph G of Figure 2, the black vertices form a mixed $(2, s)$-hide-out for any $s \geqslant 2$.

We are now ready to give our min-max characterizations for both $\mathbf{vlns}^s(G)$ and $\mathbf{vlms}^s(G)$ for all $s \in \mathbb{N}^+$.

Theorem 4. *For any graph G and any $s \in \mathbb{N}^+$,*

$$\mathbf{vlns}^s(G) - 1 = \delta^s(G) = \max \{k \mid G \textit{ contains a } (k, s)\textit{-hide-out}\}$$
$$\mathbf{vlms}^s(G) - 1 = \delta^s_{\mathrm{m}}(G) = \max \{k \mid G \textit{ contains a mixed } (k, s)\textit{-hide-out}\} .$$

It follows that Algorithm 1 can be used to compute $\delta^s(G)$ for any $s \in \mathbb{N}^+$. The algorithm attempts to construct a (k, s)-layout of G greedily, which would show that $\delta^s(G) \leqslant k$. If this fails, a (k, s)-hide-out of G has been detected and, since the hide-out itself has no (k, s)-layout, nor does G. A straightforward modification of the algorithm, replacing $\mathbf{sep}^s$ with $\mathbf{msep}^s$, determines whether $\delta^s_{\mathrm{m}}(G) \leqslant k$, giving either a mixed (k,s)-layout or a mixed (k,s)-hide-out. By Lemmata 2 and 3, both variants of the algorithm run in polynomial time for $s = \infty$ and $s \leqslant 3$.

Theorem 5. *Let $s \in \mathbb{N}^+$. Given a graph G and $k \in \mathbb{N}$, the problem of determining whether $\mathbf{vlns}^s(G) \leqslant k$ (respectively, $\mathbf{vlms}^s(G) \leqslant k$) is computable in time polynomial in $|V(G)|$ if $s \in \{1,2,3,\infty\}$ and is* NP*-complete otherwise.*

Proof (sketch). The polynomial-time cases are covered above. The remaining cases for node search are in NP because the fact that $\mathbf{vlns}^s(G) \leqslant k$ is witnessed by a (k,s)-layout $v_1,\ldots,v_n$, along with an $(s, v_i, \{v_1,\ldots,v_{i-1}\})$-separator of size at most k for each $i \in \{1,\ldots,n\}$ and the validity of such a witness can be checked in polynomial time. NP-completeness is proven by reduction from the problem of determining whether $\mathbf{sep}^s_G(x,X) \leqslant k$. The mixed search versions are similar. □

The strategies we have considered only allow the cops to choose their next move based on the current position in the game. A consequence of Theorem 4 is that we do not need to consider more general forms of strategy: if there is no (k,s)-strategy for a graph, then there is a hide-out and the existence of this hide-out allows the fugitive to escape from any kind of k-cop strategy.

5 Comparisons

In this section, we show that increasing the speed of the fugitive, even by just one, or moving from bounded to unbounded speed or from a lazy fugitive with unbounded speed to an active one can increase the search number of a graph by an arbitrary amount. We analyse the simpler node-search parameters but the same conclusions can be derived for mixed search using the same ideas.

Proposition 6. *For any G, $\delta^1(G) \leqslant \delta^2(G) \leqslant \delta^3(G) \leqslant \cdots \leqslant \delta^\infty(G) \leqslant \mathbf{tw}(G)$.*

Proof. Immediate from the game characterizations. From left to right, the fugitive becomes stronger: from lazy with unit speed, through lazy with increasing bounded speed, to lazy with unbounded speed and, finally, active with unbounded speed. □

Thus, the existence of a (k,∞)-hide out can provide easy lower bounds for the treewidth of graphs. For instance, the graph H in Figure 1 contains a $(6,\infty)$-hide-out, so $\mathbf{tw}(H) \geqslant \delta^\infty(H) \geqslant 6$. Since H is a minor of G, we have $\mathbf{tw}(G) \geqslant 6$. It is easy to see that seven cops have a winning node search strategy on G against a visible, active fugitive, so $\mathbf{tw}(G) < 7$ and we conclude that $\mathbf{tw}(G) = 6$.

The main result of this section is the following.

Theorem 7. *Let $3 \leqslant d_1 \leqslant d_2 \leqslant \cdots \leqslant d_r \leqslant d_{r+1} \leqslant d_{r+2}$. There is a connected graph G such that $\delta^s(G) = d_s$ for $s \leqslant r$, $\delta^\infty(G) = d_{r+1}$ and $\mathbf{tw}(G) = d_{r+2}$.*

6 Extending Contraction Degeneracy

A popular approach to estimating treewidth is to look for algorithms or heuristics that compute lower bounds for it. By Proposition 6, degeneracy gives such a lower bound but, by considering variants of grid graphs with degree at most three, it is easily seen that there are graphs G of arbitrary treewidth but with $\delta^1(G) = 3$.

Bodlaender, Koster and Wolle [6,25,15] define the *contraction degeneracy* of a graph to be

$$\delta\mathrm{C}(G) = \max\left\{\delta^1(H) \mid H \text{ is a non-trivial minor of } G\right\}.$$

Contraction degeneracy seemed to be a good lower bound for treewidth — notice that $\delta^1(G) \leqslant \delta\mathrm{C}(G) \leqslant \mathbf{tw}(G)$. Bodlaender et al. prove the problem of determining, given G and k, whether $\delta\mathrm{C}(G) = k$ is NP-complete and propose heuristics for computing the parameter [6].

We have defined the hierarchy δ^s for $s \in \mathbb{N}^+$, which can be seen as an extension of degeneracy. As $\delta^1(G) \leqslant \delta^\infty(G) \leqslant \mathbf{tw}(G)$, δ^∞ is, itself, a better lower bound for treewidth than degeneracy and is still polynomial-time computable, though the same examples as before show that there are graphs G with $\delta^\infty(G) = 3$ and arbitrary treewidth. However, we can follow the approach of Bodlaender at al. and define, for any $s \in \mathbb{N}^+$, the parameter

$$\delta^s\mathrm{C}(G) = \max\left\{\delta^s(H) \mid H \text{ is a non-trivial minor of } G\right\}.$$

Note that $\delta^1\mathrm{C}(G) = \delta\mathrm{C}(G)$. The following is immediate from Proposition 6.

Proposition 8. *For any graph G,*

$$\delta^1\mathrm{C}(G) \leqslant \delta^2\mathrm{C}(G) \leqslant \delta^3\mathrm{C}(G) \leqslant \cdots \leqslant \delta^\infty\mathrm{C}(G) \leqslant \mathbf{tw}(G).$$

Thus, one can expect $\delta^\infty\mathrm{C}(G)$ to give a better lower bound for treewidth than contraction degeneracy. Unfortunately, $\delta^\infty\mathrm{C}(G)$ can, itself, be shown to be NP-complete — the proof is almost identical to the proof for contraction degeneracy in [6]. However, treewidth is bounded above by a function of $\delta^\infty\mathrm{C}(G)$, while contraction degeneracy gives only a lower bound, because $\delta\mathrm{C}(G) \leqslant 5$ for any planar G [6] but $\mathbf{tw}(G)$ can be arbitrarily large.

Theorem 9. *There is a function $f : \mathbb{N} \to \mathbb{N}$ such that, for all graphs G, $\delta^\infty\mathrm{C}(G) \leqslant \mathbf{tw}(G) \leqslant f(\delta^\infty\mathrm{C}(G))$.*

If a planar graph does not contain the $k \times k$ grid as a minor, then $\mathbf{tw}(G) \leqslant \mathcal{O}(k)$ [21]. Therefore, for planar G, $\mathbf{tw}(G) \leqslant \mathcal{O}((\delta^\infty\mathrm{C}(G))^{3/2})$. The same observation can be extended to any class of graphs with an excluded minor [7].

7 Concluding Remarks

We have studied the number of cops required to catch a lazy, visible fugitive moving with bounded or unbounded speed in a graph, using node search and the more general mixed search. We have shown that the associated search numbers correspond to graph parameters that are generalizations of the classical notion of degeneracy and characterized these parameters in terms of forbidden substructures, which we call hide-outs. Most parameters associated with graph searching are NP-complete in the case of fugitives with unbounded speed. However, our parameters are polynomial-time computable for fugitives with unbounded speed or speed at most three, and NP-complete for all other finite speeds.

For most other graph searching parameters, an important issue for proving membership in NP is proving monotonicity of the game [11]. In the monotone versions of the games, the cops are only allowed to use strategies that gradually restrict the fugitive to smaller regions of the graph such that, once he has been cut off from a vertex in the graph, he can never return there. A game is said to be *monotone* if restricting the cops to using montone strategies does not increase the number of cops required on any graph. This is not required in the cases of $\mathbf{vlns}^{\infty}$ and $\mathbf{vlms}^{\infty}$ since our complexity bounds are proven by other means. Monotonicity is a natural property of graph searching games in its own right and it is natural to ask whether the games we have defined are equivalent to monotone versions. However, in the case of a visible, lazy fugitive with bounded speed, it is not obvious how one should define montonicity and we leave this as an open issue.

We have shown that δ^{∞} can serve as a lower bound for treewidth and pathwidth and that $\delta^{\infty}\mathrm{C}$ approximates treewidth. It would be interesting to know whether there are graph classes where δ^{∞} approximates treewidth or pathwidth or on which $\delta^{\infty}\mathrm{C}$ serves as a good approximation.

References

1. Alspach, B.: Searching and sweeping graphs: a brief survey. Matematiche (Catania) 59(1–2), 5–37 (2004)
2. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM Journal on Algebraic and Discrete Mathematics 8, 277–284 (1993)
3. Baier, G., Erlebach, T., Hall, A., Köhler, E., Schilling, H., Skutella, M.: Length-bounded cuts and flows. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 679–690. Springer, Heidelberg (2006)
4. Bienstock, D.: Graph searching, path-width, tree-width and related problems (a survey). In: Reliability of computer and communication networks. DIMACS Ser. Discrete Math. Theoret. Comput. Sci, vol. 5, pp. 33–49. Amer. Math. Soc. (1991)
5. Bienstock, D., Seymour, P.D.: Monotonicity in graph searching. Journal of Algorithms 12(2), 239–245 (1991)
6. Bodlaender, H.L., Wolle, T., Koster, A.M.C.A.: Contraction and treewidth lower bounds. J. Graph Algorithms Appl. 10(1), 5–49 (2006)
7. Demaine, E.D., Hajiaghayi, M.T.: Linearity of grid minors in treewidth with applications through bidimensionality. Combinatorica 28(1), 19–36 (2008)
8. Dendris, N.D., Kirousis, L.M., Thilikos, D.M.: Fugitive-search games on graphs and related parameters. Theoret. Comput. Sci. 172(1–2), 233–254 (1997)
9. Diestel, R.: Graph Theory, 3rd edn. Springer, Heidelberg (2005)

10. Fomin, F.V., Petrov, N.N.: Pursuit-evasion and search problems on graphs. In: 27th Southeastern International Conference on Combinatorics, Graph Theory and Computing. Congr. Numer., vol. 122, pp. 47–58 (1996)
11. Fomin, F.V., Thilikos, D.M.: An annotated bibliography on guaranteed graph searching. Theoret. Comput. Sci. 399(3), 236–245 (2008)
12. Itai, A., Perl, Y., Shiloach, Y.: The complexity of finding maximum disjoint paths with length constraints. Networks 12(3), 277–286 (1982)
13. Kinnersley, N.G.: The vertex separation number of a graph equals its path-width. Information Processing Letters 42(6), 345–350 (1992)
14. Kirousis, L.M., Papadimitriou, C.H.: Interval graphs and searching. Discrete Math. 55(2), 181–184 (1985)
15. Koster, A.M.C.A., Wolle, T., Bodlaender, H.L.: Degree-based treewidth lower bounds. Technical Report UU-CS-2004-050, Department of Information and Computing Sciences, Utrecht University (2004)
16. Lovász, L., Neumann-Lara, V., Plummer, M.: Mengerian theorems for paths of bounded length. Periodica Math. Hungarica 9(4), 269–276 (1978)
17. Matula, D.W.: A min–max theorem for graphs with application to graph coloring. SIAM Reviews 10, 481–482 (1968)
18. Parsons, T.D.: Pursuit-evasion in a graph. In: Theory and Applications of Graphs, Proc. Internat. Conf., Western Mich. Univ., Kalamazoo MI, 1976. Lecture Notes in Math., vol. 642, pp. 426–441. Springer, Heidelberg (1978)
19. Petrov, N.N.: A problem of pursuit in the absence of information on the pursued. Differentsial'nye Uravneniya 18(8), 1345–1352, 1468 (1982)
20. Richerby, D.M., Thilikos, D.M.: Graph searching in a crime wave. In: Brandstädt, A., Kratsch, D., Müller, H. (eds.) WG 2007. LNCS, vol. 4769, pp. 21–32. Springer, Heidelberg (2007)
21. Robertson, N., Seymour, P.D., Thomas, R.: Quickly excluding a planar graph. J. Combin. Theory Ser. B 62(2), 323–348 (1994)
22. Seymour, P.D., Thomas, R.: Graph searching and a min-max theorem for tree-width. J. Combin. Theory Ser. B 58(1), 22–33 (1993)
23. Stamatiou, Y.C., Thilikos, D.M.: Monotonicity and inert fugitive search games. Electronic Notes in Discrete Mathematics 3, 184 (1999)
24. Takahashi, A., Ueno, S., Kajitani, Y.: Mixed searching and proper-path-width. Theoretical Computer Science 137(2), 253–268 (1995)
25. Wolle, T., Koster, A.M.C.A., Bodlaender, H.L.: A note on contraction degeneracy. Technical Report UU-CS-2004-042, Department of Information and Computing Sciences, Utrecht University (2004)

A Faster Shortest-Paths Algorithm for Minor-Closed Graph Classes

Siamak Tazari[1,⋆] and Matthias Müller-Hannemann[2]

[1] Technische Universität Darmstadt, Darmstadt, Germany
tazari@algo.informatik.tu-darmstadt.de
[2] Martin-Luther Universität Halle-Wittenberg, Halle (Saale), Germany
muellerh@informatik.uni-halle.de

Abstract. We generalize the linear-time shortest-paths algorithm for planar graphs with nonnegative edge-weights of Henzinger et al. (1994) to work for any proper minor-closed class of graphs. We argue that their algorithm can not be adapted by standard methods to all proper minor-closed classes. By using recent deep results in graph minor theory, we show how to construct an appropriate recursive division in linear time for any graph excluding a fixed minor and how to transform the graph and its division afterwards, so that it has maximum degree three. Based on such a division, the original framework of Henzinger et al. can be applied. Afterwards, we show that using this algorithm, one can implement Mehlhorn's (1988) 2-approximation algorithm for the Steiner tree problem in linear time on these graph classes.

1 Introduction

The single-source shortest-paths problem with nonnegative edge-weights is one of the most-studied problems in computer science, because of both its theoretical and practical importance. Dijkstra's classical algorithm [1] has ever since its discovery been one of the best choices in practice. Also from a theoretical point of view, until very recently, it had the best running time in the addition-comparison model of computation, namely $O(m+n\log n)$ using Fibonacci heaps [2] (we use n to denote the number of vertices of a graph and m for its number of edges). Pettie and Ramachandran [3] improved the running time in undirected graphs for the case when the ratio r between the largest and smallest edge-weight is not too large. They achieve a running time of $O(m\alpha(m,n) + \min\{n\log n, n\log\log r\})$, where $\alpha(m,n)$ is the very slowly growing inverse-Ackermann function. Goldberg [4] proposed an algorithm that runs on average in linear time. For the case of integer edge-weights, Thorup [5] presented a linear-time algorithm in the word RAM model of computation, where the bit-manipulation of words in the processor is allowed. Hagerup [6] extended and simplified Thorup's ideas to work for directed graphs in nearly linear time. But the question whether

⋆ This author was supported by the Deutsche Forschungsgemeinschaft (DFG), grant MU1482/3-1.

the standard addition-comparison model allows shortest-paths computation in worst-case linear-time is still open.

For planar graphs, Henzinger et al. [7] presented the first linear-time algorithm to calculate shortest-paths with nonnegative edge-weights. Their algorithm works on directed graphs. It is based on Frederickson's [8,9] work who gave an $O(n\sqrt{\log n})$-time algorithm for this case and whose idea was in turn based on planar separators [10] to decompose the graph. Henzinger et al. first decompose the graph into a recursive division and then use this division to relax the edges in a certain order that guarantees linear running time. They claim that their algorithm can be adapted to work for any proper minor-closed family of graphs where small separators can be found in linear time. Recently, Reed and Wood [11] improved the quadratic-time separator of Alon et al. [12] and showed that all proper minor-closed graph classes can be separated in linear time; so, we should be done. *However, both Frederickson's algorithm and Henzinger et al.'s algorithm assume that the graph has maximum degree* 3*; while this property can be achieved easily for planar graphs, we argue that it can not be achieved by standard methods for arbitrary minor-closed classes* (in particular, it can not be applied to apex graphs, i.e. planar graphs augmented by a "super-source"; these graphs have frequent application in the literature). We show how to build an appropriate recursive division of a graph from a proper minor-closed family in linear time by a non-trivial extension of the algorithm in [7]. Our algorithm works for graphs with arbitrary degrees. But even after having the recursive division, the shortest paths algorithm in [7] depends on the assumption that the graph has bounded degree (and contains only a single source labeled initially with distance zero, cf. apex graphs). Using our recursive division, we show how to transform the graph and its division to have maximum degree 3, so that Henzinger et al.'s shortest-paths algorithm can be applied. Our modifications lead to the *first linear-time shortest-paths algorithm for all proper minor-closed classes of graphs* in the addition-comparison model of computation.

We also consider the Steiner tree problem, namely finding the shortest tree that connects a given set of terminals in an undirected graph. The Steiner tree problem is also one of the most fundamental problems in computer science and of the first problems shown to be $\mathcal{NP}$-complete [13]. The best known approximability/inapproximability results are $1.55 + \epsilon$ (shown in [14]) and 1.01053 (shown in [15]), respectively. There exists a well-known 2-approximation algorithm for this problem [16,17] and Mehlhorn [18] showed how to implement it in time $O(m + n \log n)$. No better time complexity for such an approximation is known even for planar graphs. We show how to implement this algorithm in linear time on all proper minor-closed graph classes. An important observation that we made to improve this running time is that Mehlhorn's distance network is a minor of the given graph and thus, its minimum spanning tree can be calculated in linear time with the algorithm of Mares [19].

The area of graph minor theory has been constantly evolving ever since the graph minor theorem of Robertson and Seymour [20] was announced in 1988. Many important algorithms and meta-algorithms have been presented for large problem families on minor-closed graph classes and numerous theoretical concepts have been developed to handle them. We present the first linear-time algorithms for two fundamental graph-theoretic problems in these classes.

In Section 2, we review some needed concepts and previous work; in Section 3, we present our main result about shortest paths and in Section 4, the application to Steiner tree approximation.

2 Preliminaries

In this section, we review some concepts and some previous results that are needed in this work. These include graph minors, vertex partitioning, graph decomposition, and an overview of Henzinger et al.'s [7] single-source shortest-paths algorithm.

2.1 Graph Minors

A *minor* of a graph G is a graph that is obtained from a subgraph of G by contracting a number of edges. A class of graphs is *minor-closed* if it is closed under building minors. It is called a *proper* class if it is neither empty nor the class of all graphs. Examples of proper minor-closed graph families are planar graphs, bounded-genus graphs, and apex graphs. The seminal theorem of Robertson and Seymour [20] states that any proper minor-closed class of graphs can be characterized by a finite set of excluded minors. This is a very broad generalization of Kuratowski's theorem about planar graphs. Note that for a proper minor-closed class of graphs, we can always consider the number of vertices ℓ of the smallest excluded minor and conclude that the complete graph $\mathcal{K}_\ell$ is a particular excluded minor of the class. Thus, the class of $\mathcal{K}_\ell$-minor-free graphs includes the considered minor-closed class of graphs. In the rest of this work, we work with $\mathcal{K}_\ell$-minor-free graphs, where ℓ is a fixed constant.

It follows from a theorem of Mader [21] that $\mathcal{K}_\ell$-minor-free graphs have constant average degree, for some constant depending on ℓ. This, in turn, implies that these classes of graphs are sparse, i.e. we have $m = O(n)$. For planar graphs, we know by Euler's formula that the number of edges is at most only $3n - 6$.

2.2 Vertex Partitioning

In [8], Frederickson presented a simple algorithm called `FindClusters`, based on depth-first search, that given a parameter z and an undirected graph *with maximum degree* 3, partitions its vertices into *connected components* each having at least z and at most $3z$ vertices. Note that since the algorithm gives us connected components, we can contract each one of them and get a minor of the input graph with at most n/z vertices. He used this algorithm to derive fast algorithms for the minimum spanning tree and shortest-paths [9] problems. If a weighted graph does not have maximum degree 3, one can apply the following transformation: replace a vertex v of degree $d(v)$ with a zero-weight cycle of length $d(v)$, such that each edge incident to v is now incident to exactly one vertex of the cycle (a similar transformation can be applied to directed graphs, too). If the given graph is embedded in a surface, one can order the edges around a cycle in the same way they were ordered around the corresponding vertex in the given embedding. This way, the transformed graph will also be embedded in the same surface. However, for an arbitrary minor-closed class of graphs (e.g. apex graphs), it might not always be possible to remain in the class after transforming the graph this way, see Section 3. But Frederickson's `FindClusters` *depends* on the graph having bounded degree. Any constant bound would suffice for our purposes but in general such a bound does not exist for arbitrary minor-closed graph families.

Reed and Wood [11] introduced an alternative partitioning concept that can be applied to a graph $G = (V, E)$ with arbitrary degrees excluding a fixed minor. Consider some partitioning $\mathcal{P} = \{P_1, \dots, P_t\}$ of the vertex set V. Let $H = (V_H, E_H)$ be the graph obtained by collapsing every part P_i of G into a single vertex $v_i \in V_H$ $(1 \le i \le t)$ and removing loops and parallel edges. This way, there is an edge between two vertices v_i and v_j of H if and only if there is an edge between a vertex of P_i and a vertex of P_j in G $(1 \le i < j \le t)$. We say $\mathcal{P}$ is a *connected H-partition* of G if $v_i v_j \in E_H$ if and only if there is an edge of G between every connected component of P_i and every connected component of P_j. Reed and Wood proved the following lemma[1]:

Lemma 2.1 ([11]). *There is a linear-time algorithm that given a constant z and a graph G excluding a fixed $\mathcal{K}_\ell$-minor, outputs a connected H-partition $\mathcal{P} = \{P_1, \dots, P_t\}$ of G such that $t \le n/z$, and $|P_i| < c_0 \cdot z$ for all $1 \le i \le t$, where c_0 is a constant depending on ℓ.* □

Note that by contracting each connected component of each P_i in G to a single vertex, one gets a graph that contains an isomorphic copy of H as a subgraph and so, H is a minor of G and in particular, is also $\mathcal{K}_\ell$-minor-free. Hence, when dealing with graphs with no bounded degree, Lemma 2.1 can be used instead of `FindClusters` to partition the graph and reduce its size while keeping it free of some fixed minor.

2.3 Graph Decomposition

A *balanced node-separation* of a graph $G = (V, E)$ is given by two sets A and B, such that $A \cup B = V$, there is no edge between $A \setminus B$ and $B \setminus A$, and each one of A and B contains at most an α-fraction of the nodes (for some $1/2 \le \alpha < 1$). The *size* of the separation is $|A \cap B|$. For a function f, a subgraph-closed class of graphs is said to be *f-separable* if every n-node graph in the class has an $O(f(n))$-size separator. Reed and Wood [11] showed that all $\mathcal{K}_\ell$-minor-free graphs are f-separable in *linear* time for $f(n) = O(n^{2/3})$. For planar graphs, one can use the original planar separator theorem of Lipton and Tarjan [10] that delivers an $O(\sqrt{n})$-separator in linear time.

[1] In their lemma, we substitute $c_0 := 2^{\ell^2+\ell}$ and $z := 2k/c_0$.

An (r, s)-*division* of an n-node graph is a partition of the edges of the graph into $O(n/r)$ regions, each containing $r^{O(1)}$ nodes and each having at most s *boundary nodes* (i.e. nodes that occur in more than one region). For a nondecreasing positive integer function f and a positive integer sequence $\overline{r} = (r_0, r_1, \ldots, r_k)$, an $(\overline{r}, f)$-*recursive division* of an n-node graph is defined as follows: it contains one region R_G consisting of all of G. If G has more than one edge and $\overline{r}$ is not empty, then the recursive division also contains an $(r_k, f(r_k))$-division of G and an $(\overline{r}', f)$-recursive division of each of its regions, where $\overline{r}' = (r_0, r_1, \ldots, r_{k-1})$. A recursive division can be represented compactly by a *recursive division tree*, a rooted tree whose root represents the whole graph and whose leaves represent the edges of the graph. Every internal node represents a region, namely, the region induced by all the leaves in its subtree. The children of a node of the tree are its immediate subregions in the recursive division.

Using Frederickson's partitioning [8] and division [9] methods, Henzinger et al. [7] present a linear-time algorithm to find certain recursive divisions in planar graphs: they determine a vector $\overline{r}$ and an $(\overline{r}, cf)$-recursive division of the graph for some constant c, such that the inequality

$$\frac{r_i}{f(r_i)} \geq 8^i f(r_{i-1}) \log r_{i+1} (\sum_{j=1}^{i+1} \log r_j) \tag{1}$$

is satisfied for all r_i's exceeding a constant. The obtained recursive division tree has $O(n)$ nodes and its depth is roughly $O(\log^\star n)$.

2.4 Single-Source Shortest-Paths on Planar Graphs

Henzinger et al. prove the following theorem:

Theorem 2.2 ([7]). *Let a graph G with maximum in-/outdegree 2 and, for some constant c, an $(\overline{r}, cf)$-recursive division tree of G be given, such that inequality (1) is satisfied for all r_i's exceeding a constant. Then, the single-source shortest-paths problem with nonnegative edge-weights can be solved on G in linear time.* □

To prove this theorem, they use a complicated charging scheme that also depends on the graph having *a single source and bounded degree.* Together with the result from the previous subsection, it follows that single-source shortest-paths with nonnegative edge-weights can be calculated in linear-time on planar graphs.

3 Single-Source Shortest Paths on Minor-Closed Graph Classes

In this section, we prove our main theorem about shortest paths:

Theorem 3.1. *In every proper minor-closed class of graphs, single-source shortest-paths with nonnegative edge-weights can be calculated in linear time.*

First, we argue that the degree requirement of Henzinger et al.'s algorithm can not be fulfilled by standard methods for arbitrary minor-closed classes of graphs. By "standard methods" we mean splitting a vertex using zero-weight edges until the desired degree bound is reached. In Subsection 2.2 we discussed a particular way of splitting vertices that can be applied to embedded graphs. In this section, we show that there exist $\mathcal{K}_\ell$-minor-free graphs, so that *no matter how we split the vertices*, the resulting graph will include a minor whose size can not be bounded by a function in ℓ. The key lies in the observation that splitting an apex might introduce arbitrarily large minors. This is a well-known fact in graph minor theory [22]. For the sake of completeness, we include a short proof below. Apices are a fundamental part of minor-closed graph classes as is demonstrated by the powerful graph-decomposition theorem of Robertson and Seymour [22]. This theorem shows, in a sense, that at most a bounded number of apices are allowed in these classes; and intuitively, splitting an apex with unbounded degree might result in an unbounded number of apices and is thus not allowed in general.

Proposition 3.2. *For every $k \in \mathbb{N}$, there exists a $\mathcal{K}_6$-minor-free graph G_k, so that, if the vertices of G_k are split in any way to achieve a maximum degree of* 3*, the resulting graph G'_k includes a $\mathcal{K}_k$-minor.*

Proof (sketch). We define G_k to be a sufficiently large planar grid-graph augmented by an apex as follows: consider a sequence S of numbers between 1 and k, so that each possible pair of these k numbers is at least once adjacent in S. Let $t < k^2$ be the length of this sequence. Choose a set W of t vertices in the grid that are sufficiently far away from each other and add an apex v_0 connected to these t vertices. This completes the definition of G_k, which is clearly $\mathcal{K}_6$-minor-free. Now, no matter how we split the vertices of G_k, the apex v_0 will become a path of t vertices, each one connected to exactly one vertex of W. This path imposes an order on the vertices in W. We label the vertices in W according to this order using the sequence S. Let W_i be the set of vertices in W labeled by i ($1 \leq i \leq k$). For each i, construct a tree T_i that connects the vertices of W_i in the planar grid. Note that if the grid is sufficiently large and the vertices in W are sufficiently far away from each other, it is easily possible to choose the trees T_i to be all disjoint. Let U be the set of edges connecting the vertices in W with the path resulted from splitting v_0. Now, if we contract the trees T_i and

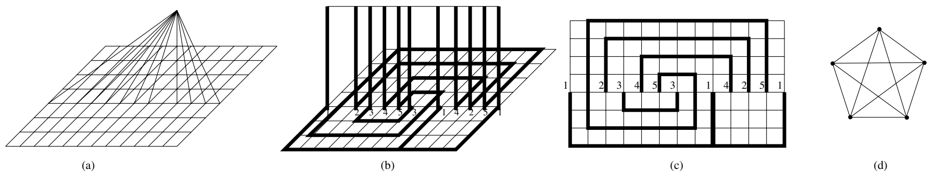


Fig. 1. A simplified example for the proof of Proposition 3.2: the apex of the graph in (a) is split, resulting in the graph (b); the vertices are labeled and connected according to the disjoint trees in (c); contracting the thick edges in (b) results in a $\mathcal{K}_5$-minor (d)

the edges in U and delete redundant edges, what remains is a $\mathcal{K}_k$-minor (see Fig. 1). □

Now we proceed with our modified algorithm and its proof of correctness.

3.1 Our Generalized Recursive Division Algorithm

The idea of Henzinger et al.'s[7] algorithm is as follows: first, iteratively reduce the size of the graph by partitioning the vertices of the graph and building minors; then, working backwards, find (r, s)-divisions of the smaller graphs (for appropriate values of r and s), imposing divisions on the larger graphs and at the same time building the recursive division tree. Since the time-consuming calculation of (r, s)-divisions is done on the smaller graphs, they succeed to prove that the overall time complexity is linear.

We present our modified algorithm in Alg. 1. Our modifications are only in three places but as we already discussed, they are essential to make the algorithm work for all proper minor-closed graph classes. These modifications along with the proof of correctness are presented in the following theorem:

Theorem 3.3. *There is a linear-time algorithm that given a $\mathcal{K}_\ell$-minor-free graph G, finds an $(\overline{r}, f)$-recursive division of G that satisfies inequality (1) for all r_i exceeding a constant and whose recursive division tree has $O(n)$ nodes.*

Proof. Consider Alg. 1 and let n_i be the number of vertices of G_i. Our modifications are as follows:

(i) In the first part of the algorithm, Henzinger et al. used Frederickson's `FindClusters` [8] to partition the graph into connected components and contract each one to obtain a minor G_{i+1} for each given G_i. We use instead Reed and Wood's connected H-partition [11] to achieve a similar effect without depending on the graph to have bounded degree; the only difference is that we have to deal with larger constants: the graph G_{i+1} will be a minor of G_i having at most n_i/z_i vertices, each representing at most $c_0 z$ vertices of G_i (cf. Subsection 2.2 and Lemma 2.1).

(ii) In the second part, the original algorithm makes use of the (r, s)-division procedure of Frederickson [9], which is based on the $O(\sqrt{n})$-planar separator of Lipton and Tarjan [10]. We replace the planar separator in this procedure with Reed and Wood's linear-time $O(n^{\frac{2}{3}})$-separation algorithm for $\mathcal{K}_\ell$-minor-free graphs. The `Divide` procedure takes parameters G, S, r, and ℓ and has now the following modified properties (see the journal version of this paper for a proof): it divides the edges of an n-node graph G and a node-subset S into at most $c_2(|S|/r^{\frac{2}{3}} + \frac{n}{r})$ regions, each one having at most r nodes and at most $c_1 r^{\frac{2}{3}}$ boundary nodes, where the nodes in S are counted as boundary nodes, too, and c_1 and c_2 are constants. The parameter ℓ is taken to indicate the excluded minor. If the input graph G has n nodes, the `Divide` procedure takes time $O(n \log n)$.

(iii) Finally, the definition of the sequence z_i also has to change. For the proof of inequality (1) to work, we need to reduce the exponent in the recursive definition of the z_i to $\frac{1}{7}$, i.e. define $z_{i+1} = 14^{z_i^{(1/7)}}$. Note that the choice of 14

Algorithm 1. Generalized Recursive Division Algorithm

```
Input  : An undirected graph $G = (V, E)$ excluding a $\mathcal{K}_\ell$-minor.
Output: A recursive division tree $T$ for $G$ satisfying inequality (1) for all $r_i$
        exceeding a constant.
begin
    // partition and contract the graph recursively
    let $G_0 := G$, $z_0 := 2$, $i := 0$;
    while the number of nodes in $G_i > \frac{n}{\log n}$ do
        let $G_{i+1} := H$-Partition$(G_i, z_i, \ell)$;
        let $z_{i+1} := 14^{z_i^{(1/7)}}$, $i := i + 1$;
    let $I := i - 1$;

    // divide the graphs and build recursive division tree
    let $v_G$ be the root of $T$;
    let $D_{I+1}$ be the trivial division of $G_{I+1}$ consisting of a single region;
    for $i := I$ downto 0 do
        for each region $R$ of $D_{i+1}$ do
            let $S_R$ be the boundary-nodes of $R$ in the division $D_{i+1}$;
            let $D_R :=$ Divide$(R, S_R, z_i, \ell)$;
            for each region $R'$ of $D_R$ do
                expand $R'$ into a region $R''$ of $G_i$ by expanding every vertex;
                assign each boundary edge to one of the regions it occurs in;
                create a child $v_{R''}$ of $v_R$ in $T$;
        let $D_i$ be the decomposition of $G_i$ consisting of the regions $R''$ above;

    // add the leaves
    for each edge $e$ of each region $R$ of $D_0$ do
        create a child $v_e$ of $v_R$ in $T$;
    return $T$;
end
```

as the base of the exponentiations above (and the choice of 7 in the original algorithm) is not arbitrary; these are the smallest values that ensure that the defined sequences grow (very rapidly) towards infinity.

With the changes given above, the proof of the correctness of the algorithm and all the calculations therein can be carried out in a similar way as is done in [7]; a number of subtle details have to be adapted, as is shown in the journal version of our work. The proof has four parts: First, it is shown that each region of the division D_i has at most $O(z_i^2)$ vertices and at most $O(z_i^{\frac{5}{3}})$ boundary vertices. Second, the number of regions in each division D_i is shown to be $O(n_i/z_i^2)$. Third, it is proven that the algorithm takes linear time and finally, the correctness of inequality (1) is asserted. In this version of the paper, we only present the proof of the last statement, namely, the correctness of (1):

First, note that combining the inequalities $n_{i+1} \leq n_i/z_i$, we obtain $n_i \leq n/\prod_{j<i} z_j$. Note moreover that each node of G_i expands to at most $\prod_{j<i} c_0 z_j$ nodes of G. Consider the division D_i of G_i, and the division it induces on G. The division D_i consists of $O(n_i/z_i^2)$ regions (see above), each having $O(z_i^2)$

vertices and $O(z_i^{\frac{5}{3}})$ boundary vertices. This induces $O(n_i/z_i^2)$ regions in G, each consisting of $O(z_i^2 \prod_{j<i} c_0 z_j)$ vertices and $O(z_i^{\frac{5}{3}} \prod_{j<i} c_0 z_j)$ boundary vertices. Let $r_i = z_i^2 \prod_{j<i} z_j$ and define $f(r_i) = z_i^{\frac{5}{3}} \prod_{j<i} c_0 z_j$. Then, the induced division of G has $O(n/r_i)$ regions each with $O(r_i c_0^i)$ vertices and $O(f(r_i))$ boundary vertices. Since $c_0^i = O(\prod_{j\le i} z_j)$, we get that the number of vertices per region is $O(r_i^2)$.

We have $\frac{r_i}{f(r_i)} = \frac{z_i^2}{z_i^{\frac{5}{3}} c_0^{i-1}} = \frac{z_i^{\frac{1}{3}}}{c_0^{i-1}}$. Using the definition of z_i, one can verify that $z_{i-1} = \theta(log^7 z_i)$ and $\prod_{j<i} z_j = O(log^8 z_i)$. Hence $f(r_{i-1}) = c_0^{i-2} z_{i-1}^{\frac{5}{3}} \prod_{j<i-1} z_j = c_0^{i-2} O(\log^{\frac{35}{3}} z_i \log^8 \log z_i)$. We also have

$$\log r_{i+1} = \log(z_{i+1}^2 \prod_{j\le i} z_j) = O(\log(z_{i+1}^2 \log^8 z_{i+1})) = O(\log z_{i+1}) = O(z_i^{\frac{1}{7}})$$

and consequently $\sum_{j=1}^{i+1} \log r_j = O(z_i^{\frac{1}{7}})$. For a sufficiently large constant i_0, we have for all $i \ge i_0$,

$$\begin{aligned} 8^i f(r_{i-1}) \log r_{i+1} (\sum_{j=1}^{i+1} \log r_j) &\le 8^i c_0^{i-2} O(\log^{\frac{35}{3}} z_i \log^8 z_i) O(z_i^{\frac{1}{7}}) O(z_i^{\frac{1}{7}}) \\ &= 8^i c_0^{i-2} O(z_i^{\frac{2}{7}} \log^{20} z_i) \le \frac{z_i^{\frac{1}{3}}}{c_0^{i-1}} = \frac{r_i}{f(r_i)}, \end{aligned} \tag{2}$$

since the z_i grow much faster than any exponential function having a constant in the base; specifically, a simple calculation shows that $z_i^{\frac{1}{21}} \ge g_0^i \log^{20} z_i$ for any constant $g_0 \ge 0$ if i is larger than a constant. So, inequality (1) is fulfilled for all r_i exceeding the constant r_{i_0}. □

3.2 Establishing the Degree Requirement

After having computed a recursive division, we still have to transform the graph to have maximum degree 3; otherwise, Lemma 2.2 can not be applied, see Subsection 2.4. We can achieve this, using our recursive division, by the following lemma. Note that according to Proposition 3.2 the resulting graph might not be $\mathcal{K}_\ell$-minor-free but it will still serve our purpose of finding shortest paths in linear time, since it is now accompanied by a recursive division satisfying equation (1).

Lemma 3.4. *Let G be an edge-weighted directed graph excluding a fixed minor and let T be a recursive division tree representing an $(\overline{r}, f)$-recursive division of G. Then one can replace every vertex of G with a zero-weight cycle to obtain a graph G' and at the same time modify T into a tree T', so that G' has in-/outdegree at most 2 and T' represents an $(\overline{r}, f)$-recursive division of G'. This modification takes linear time.*

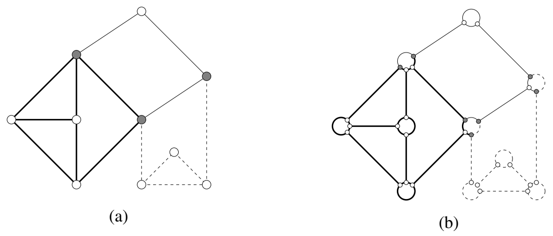


Fig. 2. (a) a given graph with 3 regions indicated by different line-styles and shaded boundary nodes; (b) the transformed graph, such that every node has degree at most 3; the number of boundary nodes of each region has exactly doubled

Proof. Recall that the leaves of T represent the edges of G and that internal nodes of T correspond to regions of G, namely, the region induced by all the leaves in the subtree of that node. We modify G and T at the same time. An in-order traversal of T induces an order on the edges of G. Replace each vertex of G with a zero-weight cycle, so that the edges around each cycle are ordered according to this order. For a splitted copy v' of a vertex v, add the outgoing edge of v' in the zero-weight cycle as a sibling of the original edge adjacent to v' to T. This gives a complete definition of G' and T'. They can be computed by a single in-order traversal of T in linear time.

Now consider an internal node q' of T'. It represents a region R' of G' and corresponds to a node q of T, representing a region R of G. R has $r^{O(1)}$ vertices and $O(f(r))$ boundary-nodes. The number of edges of R' is at most 3-times as much as in R and the number of vertices is proportional to the number of edges of R. But R is a subgraph of G, excludes the same fixed minor and thus, the number of its edges is linear in the number of its vertices. Hence, R' still has $r^{O(1)}$ vertices and edges. Also, since R is represented by the subtree rooted at q, its edges were traversed in order while building T' and G'. So, every vertex v in R is replaced by a path $v_i, v_{i+1}, \ldots, v_j$ with $1 \leq i \leq j \leq d(v)$ in R'. Thus, if v is a boundary node of R, then instead, we have v_i and v_j as boundary nodes of R'. So R' has at most twice as many boundary nodes as R, i.e. still $O(f(r))$ (see Fig. 2). So, T' represents an $(\overline{r}, f)$-recursive division of G'. □

Proof (Proof of Theorem 3.1). Note that up to the choice of the start- and endvertex inside the zero-weight cycles of G', shortest paths in G and G' correspond one-to-one to each other. G' fulfills all the requirements of Theorem 2.2 and combining this with Theorem 3.3, and Lemma 3.4, we obtain our main theorem, namely, Theorem 3.1. □

4 Steiner Tree Approximation

Theorem 4.1. *There is a linear-time algorithm that calculates a 2-approximation for the Steiner minimum tree problem in any proper minor-closed class of graphs.*

Due to space constraints, we leave the full proof of Theorem 4.1 for the journal version of this paper. The basic idea is to implement Mehlhorn's algorithm [18] in linear time using Theorem 3.1 together with the following observation:

Observation 4.2. *The distance network defined in [18] is a minor of the input graph.*

As far as we know, this observation has not been stated in the literature yet, not even for planar graphs. Mehlhorn constructs this distance network using a single-source shortest-paths computation and then determines and returns its minimum spanning tree, requiring $O(n \log n)$ time in total. Using Theorem 3.1, the construction of the network takes linear time on minor-closed graph classes and by Observation 4.2, its minimum spanning tree can also be calculated in linear time using the algorithm of Mares [19].

References

1. Dijkstra, E.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)
2. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. Journal of the ACM 34(3), 596–615 (1987)
3. Pettie, S., Ramachandran, V.: A shortest path algorithm for real-weighted undirected graphs. SIAM Journal on Computing 34(6), 1398–1431 (2005)
4. Goldberg, A.V.: A simple shortest path algorithm with linear average time. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 230–241. Springer, Heidelberg (2001)
5. Thorup, M.: Undirected single-source shortest paths with positive integer weights in linear time. Journal of the ACM 46(3), 362–394 (1999)
6. Hagerup, T.: Improved shortest paths on the word RAM. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 61–72. Springer, Heidelberg (2000)
7. Henzinger, M.R., Klein, P.N., Rao, S., Subramanian, S.: Faster shortest-path algorithms for planar graphs. Journal of Computer and System Sciences 55(1), 3–23 (1997); STOC 1994: Proceedings of the 26th Annual ACM Symposium on Theory of Computing, pp. 27–37. ACM Press, New York
8. Frederickson, G.N.: Data structures for on-line updating of minimum spanning trees, with applications. SIAM Journal on Computing 14(4), 781–798 (1985)
9. Frederickson, G.N.: Fast algorithms for shortest paths in planar graphs, with applications. SIAM Journal on Computing 16(6), 1004–1022 (1987)
10. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM Journal on Applied Mathematics 36(2), 177–189 (1979)
11. Reed, B., Wood, D.R.: Fast separation in a graph with an excluded minor. In: EuroComb 2005: 2005 European Conference on Combinatorics, Graph Theory and Applications. DMTCS Proceeding. Discrete Mathematics and Theoretical Computer Science, vol. AE, pp. 45–50 (2005)
12. Alon, N., Seymour, P., Thomas, R.: A separator theorem for nonplanar graphs. Journal of the American Mathematical Society 3(4), 801–808 (1990)
13. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103. Plenum Press (1972)

14. Robins, G., Zelikovsky, A.: Improved Steiner tree approximation in graphs. In: SODA 2000: Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 770–779 (2000)
15. Chlebík, M., Chlebíková, J.: Approximation hardness of the Steiner tree problem on graphs. In: Penttonen, M., Schmidt, E.M. (eds.) SWAT 2002. LNCS, vol. 2368, pp. 170–179. Springer, Heidelberg (2002)
16. Choukhmane, E.A.: Une heuristique pour le problème de l'arbre de Steiner. Recherche Opèrationelle 12, 207–212 (1978)
17. Plesnik, J.: A bound for the Steiner problem in graphs. Mathematica Slovaca 31, 155–163 (1981)
18. Mehlhorn, K.: A faster approximation algorithm for the Steiner problem in graphs. Information Processing Letters 27, 125–128 (1988)
19. Mares, M.: Two linear time algorithms for MST on minor closed graph classes. Archivum Mathematicum 40(3), 315–320 (2004)
20. Robertson, N., Seymour, P.D.: Graph minors. XX. Wagner's conjecture. Journal of Combinatorial Theory Series B 92(2), 325–357 (2004)
21. Mader, W.: Homomorphieeigenschaften und mittlere Kantendichte von Graphen. Mathematische Annalen 174, 265–268 (1967)
22. Robertson, N., Seymour, P.: Graph minors. XVI. Excluding a non-planar graph. Journal of Combinatorial Theory Series B 89(1), 43–76 (2003)

Local Construction and Coloring of Spanners of Location Aware Unit Disk Graphs (Extended Abstract)

Andreas Wiese[1,⋆,⋆⋆] and Evangelos Kranakis[2,⋆⋆⋆]

[1] Technische Universität Berlin, Institut fr Mathematik, Germany
[2] School of Computer Science, Carleton University, 1125 Colonel By Drive, Ottawa, Ontario, Canada K1S 5B6

Abstract. We investigate the problem of locally coloring and constructing special spanners of location aware Unit Disk Graphs (UDGs). First we present a local approximation algorithm for the vertex coloring problem in UDGs which uses at most four times as many colors as required by an optimal solution. Then we look at the colorability of spanners of UDGs. In particular we present a local algorithm for constructing a 4-colorable spanner of a unit disk graph. The output consists of the spanner and the 4-coloring. The computed spanner also has the properties that it is planar, the degree of a vertex in the spanner is at most 5 and the angles between two edges are at least $\pi/3$. By enlarging the locality distance (i.e. the size of the neighborhood which a vertex has to explore in order to compute its color) we can ensure the total weight of the spanner to be arbitrarily close to the weight of a minimum spanning tree.

We prove that a local algorithm cannot compute a bipartite spanner of a unit disk graph and therefore our algorithm needs at most one color more than any local algorithm for the task requires. Moreover, we prove that there is no local algorithm for 3-coloring UDGs or spanners of UDGs, even if the 3-colorability of the graph (or the spanner respectively) is guaranteed in advance.

1 Introduction

In the case of ad hoc networks, where there is no global entity which could assign channels (colors) to the nodes, we are interested in local algorithms. The decision about what color a local algorithm assigns to a vertex v depends only on the vertices which are a constant number of hops (edges) away from v (with the constant being independent of the size of the network). This ensures that

⋆ Research conducted while the authors were visiting the School of Computing Science at Simon Fraser University, Vancouver.

⋆⋆ Research supported by a scholarship from DAAD (German Academic Exchange Service).

⋆⋆⋆ Research supported in part by NSERC (Natural Science and Engineering Research Council of Canada). Research supported in part by MITACS (Mathematics of Information Technology and Complex Systems).

messages do not propagate uncontrollably far through the network. Locality is advantageous in dynamically changing networks, since if only local changes occur we do not have to recompute the entire solution, but only parts of it. E.g. in the event of a disaster recovery we can take advantage of the fact that we can recover parts of the solution without having to repeat the computation for the entire graph.

Unit Disk Graphs (UDGs) are widely used for modeling wireless networks. In these graphs connectivity between two nodes is established if and only if their Euclidean distance is not larger than one unit, i.e. we assume that the wireless devices have an identical transmission range. In the graph model used in this paper we also assume that each node knows about its geographic position in the plane, e.g. from a GPS receiver or from virtual coordinates assigned by another source.

1.1 Related Work

Graph coloring is a well studied subject in the literature. For general graphs it is NP-complete and even approximating it within a constant ratio is NP-hard [16]. It is even true that for any $\epsilon > 0$, the problem cannot be approximated to within $O\left(n^{1-\epsilon}\right)$ (where n is the number of nodes) unless the complexity classes NP and ZPP coincide [9].

For unit disk graphs the problem remains NP-complete [6], even when it is restricted to a fixed number of colors $k \geq 3$ [11]. However, for UDGs it can be approximated within a constant factor. Marathe et al. [17] present an offline-coloring algorithm with an approximation factor of 3 and an online-coloring algorithm with an approximation ratio of 6. Both algorithms do not need the embedding of the graph as part of the input. In [11] it is stated that Peeters in [21] has shown that this method applied to a "lexicographic" vertex ordering colors a unit disk graph G with at most $3\omega(G) - 2$ colors (where $\omega(G)$ is the clique number of G).

Gräf et al. present a factor 3 approximation algorithm [11] for the case where the embedding of the graph is known. Their algorithm exploits the topology of the graph. For the setting of location aware nodes no algorithm has been known which outperforms the online algorithm mentioned above (whose idea could be applied in this setting).

The problem of constructing spanning subgraphs (spanners) of geometric graphs has been studied widely in the literature. There are many optimization results for trade-offs between size, diameter, maximum degree and stretch factor of the computed spanner, e.g. Eppstein [8], Arya et al. [2], Narasimhan and Smid [20] and Bose et al. [3]. However, all these algorithms are global, i.e. they need the whole graph as the input.

Linial introduced the model of local computation in [15] and proved bounds for local vertex coloring algorithms. In [19] Naor and Stockmeyer provide a framework for local algorithms for Locally Checkable Labeling Problems (LCL). Vertex coloring is an LCL problem. When looking for local algorithms for constructing spanners of unit disk graphs, Bose et al. [4] address this problem by constructing a planar spanner using the Gabriel test [10]. Li et al. [12] present a local algorithm which computes a planar spanner with constant stretch factor. In [22,13] Li and Wang introduce local algorithms which compute planar spanners with constant stretch factor and a constant maximum degree. However, the resulting maximum degree can be up to 20 and the weight of the edges can be much higher than in a minimum spanning tree (MST). Li et al. [14] present a local algorithm which computes a planar spanner of a unit disk graph with a node degree bounded by 6. Chavez et al. [5] further analyzed this algorithm when operating on quasi unit disk graphs, proved an upper bound for the weight of the spanner in comparison with an MST, and improved the maximum node degree to 5 for the case of unit disk graphs. Every planar graph and therefore every planar spanner of a unit disk graph can be colored with at most 4 colors due to the well known Four-Color-Theorem [1]. However, the algorithm presented there cannot be implemented as a local algorithm. Czyzowicz et al. [7] present a local algorithm which colors a given planar spanner of a unit disk graph with at most 7 colors.

1.2 Main Results and Outline of the Paper

In this paper we present a local algorithm with polynomial processing time which colors the vertices of a unit disk graph and needs at most most 4 times as many colors as an optimal coloring requires. It is the first local algorithm for this task. Its approximation ratio is better than the ratio of 6 which is guaranteed by the online algorithm [17], but a bit higher than the performance ratio of 3 which is achieved by the best global polynomial time algorithms [17,11]. We also have a local algorithm which uses at most 3 times the optimal number of colors but it requires exponential processing time.

We also present a local algorithm which computes a 4-colorable spanner of a unit disk graph as well as a 4-coloring for the spanner. By employing the local algorithm presented in [5] for preprocessing, we can guarantee that our spanner is planar, the maximum node degree is bounded by 5 and any angle between two edges is at least $\pi/3$. As described in [5] we can also ensure its weight to be at most $(k+1)/(k-1)$ times the weight of a minimum spanning tree for an arbitrary large k. The locality distance (the size of the neighborhood which a vertex has to explore in order to compute its color) of the algorithm is $136 + k$. This is the first local algorithm which computes a spanner of a unit disk graph while computing a coloring for it. Using at most 4 colors, we need fewer colors than the local 7-coloring algorithm in [7] which colors an arbitrary planar spanner of a unit disk graph. It is possible to reduce the locality distance of our algorithm to $34 + k$ by using one additional color while still ensuring the properties discussed above.

Further we show that there is no local algorithm for computing 2-colorable (i.e., bipartite) spanners, even if we do not compute the coloring but only the spanner itself. We also show that there is no local algorithm for coloring 3-colorable unit disk graphs or 3-colorable spanners of unit disk graphs using at most 3 colors.

Due to the space constraints we cannot give full proofs of all our theorems. For all details we refer to the technical report [23] and give sketches of the proofs instead.

2 Preliminaries

All algorithms presented in this paper are local algorithms for unit disk graphs. An undirected graph $G = (V, E)$ is a *unit disk graph* if there is an embedding in the plane for G such that two vertices u and v are connected by an edge if and only if the Euclidean distance between them is at most 1. The graph G we consider for all our algorithms is a connected unit disk graph. For two vertices u and v let $d(u, v)$ be the hop-distance between u and v, that is the number of edges on a shortest path between these two vertices. Denote by $N^r(v) = \{u \in V \mid d(u, v) \leq r\}$ the r-th neighborhood of a vertex v. For ease of notation we set $N^0(v) := \{v\}$, $N(v) := N^1(v)$ and for a set $V' \subseteq V$ we define $N(V') = \bigcup_{v' \in V'} N(v')$. Note that $v \in N(v)$. We define the diameter of a set of vertices $V' \subseteq V$ as $diam(V') := \max_{u,v \in V'} d(u, v)$. It is assumed that in all our algorithms an embedding for G is given. For a vertex v we denote by v_x its x-coordinate and by v_y its y-coordinate. We denote by $ht(G)$ the *height* of G, defined by $ht(G) := \max_{u,v \in V}\{u_y - v_y\}$.

We denote by the *locality distance* (or short the *locality*) of an algorithm the minimum α such that the status of any vertex v (e.g. its color, whether or not it is in a computed set etc.) depends only on the vertices in $N^\alpha(v)$. In the graph model which we use we assume that each vertex v is aware of its geographic position in the plane. We also assume that each vertex v can find out the geographic position of the vertices which are at most α hops away from v by message passing.

A *coloring* of a graph G is a map $color : V \to \{1, ..., c\}$ such that $(v_1, v_2) \in E \Rightarrow color(v_1) \neq color(v_2)$. For ease of notation we define $|color| := c$. We denote by $\chi(G)$ the chromatic number of G. That is the minimum number of colors that is needed for a coloring of G. We define $\omega(G)$ to be the clique number, i.e. the size of the largest clique in G.

We denote by $\triangle(G)$ the maximum degree of a node in G. If G is a geometric graph, we define $cost(G)$ as the sum of Euclidean lengths of the edges of G. For a graph G we denote by $E(G)$ the set of its edges and by $V(G)$ the set of its vertices. For a set of vertices V' we denote by $G[V']$ the subgraph of G induced by V'. If an embedding of G is given, then for a rectangle R in the plane we denote by $G[R]$ the subgraph of G induced by the vertices in R.

3 Local $4 \cdot \chi(G)$ Vertex Coloring

In this section we present a local approximation algorithm for vertex coloring in a unit disk graph G. We prove that it achieves a competitive ratio of 4 and that the processing time for each vertex is bounded by a polynomial. We employ a result by Grf et al. [11] that enables us to use an algorithm by Mhring [18] as a subroutine which colors a unit disk graph optimally in polynomial time when its height is at most $\sqrt{3}/2$. Before we present the algorithm we introduce a tiling of the plane that we are going to use.

3.1 Tiling of the Plane

We divide the plane into rectangles and assign a class number to each rectangle. The tiling achieves the following properties:

- Each vertex of G is in exactly one rectangle.
- The height of each rectangle is smaller than $\sqrt{3}/2$.
- Each rectangle has a class number between 1 and 4.
- The Euclidean distance between any two points in two different rectangles with the same class number is strictly greater than one.

We achieve these properties as follows: We divide the plane into a grid where each grid cell is a rectangle with height $1/2 + \epsilon$ and width $1 + \epsilon$. We choose ϵ such that $0 < \epsilon \leq \frac{1}{64}$. We place tiles of rectangles into the grid. Figure 1 shows one tile. The rectangles of class 1 have the size of 1 grid cell, the rectangles of classes $2, 3$ and 4 have the size of 5 grid cells (later, we use the different sizes of rectangles in order to achieve a lower locality distance in our algorithm). The class numbers of the rectangles are assigned according to Figure 1 (white=class 1, black=class 2, dark gray=class 3 and light gray=class 4). We tile the whole plane with such tiles, starting at an arbitrary position. Figure 2 shows a part of this tiling.

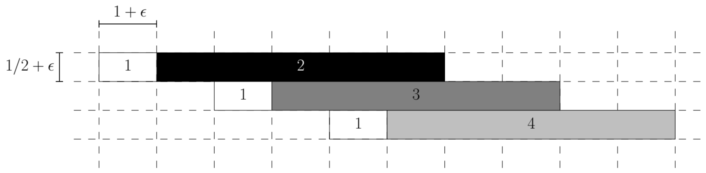


Fig. 1. One tile of the tiling. The numbers in the rectangles indicate the class number of the respective rectangle.

Each vertex of G is contained in exactly one rectangle. Ambiguities caused by vertices on the border of a rectangle are resolved by assigning them to the rectangle with the lowest class number which contains them. From the construction it follows that two different rectangles of the same class have an Euclidean distance of strictly more than one. So we conclude that two vertices in different rectangles of the same class are not adjacent. We also observe that every vertex can determine its class number in constant time by only using its coordinates.

3.2 The Algorithm

Now we present our algorithm. The main idea is to solve the coloring problem optimally for the rectangles of each class separately. First we color the vertices

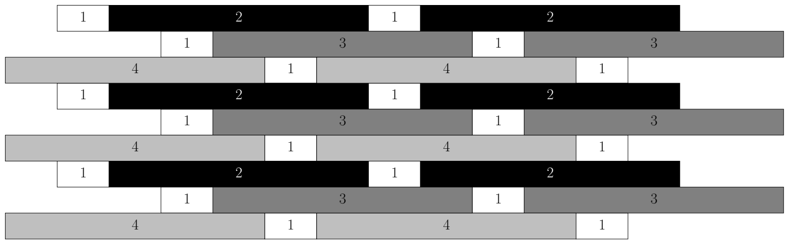


Fig. 2. A part of the tiling of the plane used in Algorithm 1

in all class 1 rectangles optimally. Then we solve the problem for each connected component C in each class 2 rectangle R optimally under the condition that we are not allowed to use colors that have been used by vertices which are adjacent to any vertex in C. Then we do the same for all class 3 rectangles and then for all class 4 rectangles.

Now we present our algorithm in detail. We start with a coloring *color* defined by $color(v) := 0$ for all $v \in V$. For $i := 1, 2, 3, 4$ we do the following: Consider a connected component C in a rectangle R of class i and denote its vertices by V_C. Denote by G_C the subgraph induced by V_C. By construction of the tiling the height of R is smaller than $\sqrt{3}/2$ and thus $ht(G_C) \leq \sqrt{3}/2$. In [11] Grf et al. state an algorithm which computes an optimal coloring for a unit disk graph G_C in time $O\left(|V_C|\,\omega\left(G_C\right)^2\right)$ if $ht\left(G_C\right) \leq \sqrt{3}/2$ (they call such graphs $\sqrt{3}/2$-stripes). We use this algorithm to compute an optimal coloring $color_C$ for G_C. We might not be able to use the assignment of colors in $color_C$ directly in the coloring *color* which has been computed so far since a vertex $v \in C$ might be adjacent to a vertex v' (in another rectangle) such that $color_C(v) = color(v')$. Let c be the highest number of a color that has already been assigned to any vertex in $N(V_C)$ by *color* (i.e. $c = \max_{v \in N(V_C)} color(v)$). We define $color(v_C) := color_C(v_C) + c$ for all $v_C \in V_C$. We do this for all connected components in all rectangles of class i. As two vertices in two different connected components in rectangles of the same class number are not adjacent the order in which the connected components are being processed does not matter. We output the coloring *color*. We refer to the above as Algorithm 1.

In the following theorem we prove that Algorithm 1 is a local algorithm that computes a valid coloring with a competitive ratio of 4.

Theorem 1. *Algorithm 1 has the following properties:*

1. *The computed coloring is a valid coloring for G.*
2. *It holds that $|color| \leq 4 \cdot \chi(G)$.*
3. *The color of a vertex v depends only on the vertices which are at most* 71 *hops away from v, i.e. Algorithm 1 is local.*
4. *The processing time for a vertex v is bounded by a cubic polynomial in the number of vertices which are at most* 71 *hops away from v.*

Proof. Due to the lack of space we just give a sketch here. For each rectangle we compute an optimal coloring. Then we skip all colors that have already been used in adjacent vertices. Therefore, the overall coloring is valid. Since we have four different classes of rectangles we obtain an approximation ratio of 4. The diameter of a connected component in a rectangles is bounded. Using this it is possible to prove that the locality distance of Algorithm 1 is at most 71. The processing time is dominated by the computation of the colorings for the rectangles. Using the algorithm stated in [11] this can be done in time $O\left(|V_C|\,\omega\left(G_C\right)^2\right)$.

3.3 Local $3 \cdot \chi(G)$ Vertex Coloring

Using the tiling technique which we are going to present in Section 4 we can construct a local algorithm for vertex coloring which uses at most $3 \cdot \chi(G)$ colors and has a locality distance of 42. However, its processing time is exponential in the number of vertices at most 42 hops away from a given vertex. (We omit details for lack of space.)

4 Local Construction of 4 Colorable Spanners

In this section we present a local algorithm for computing a 4-colorable spanner of a given unit disk graph. It computes the spanner and the 4-coloring for it. For preprocessing the graph we employ the local algorithm presented in [5,14]. This ensures that the resulting spanner is planar, it does not contain any angle smaller than $\pi/3$ and the degree of any node is at most 5. For arbitrarily large k this subroutine can also guarantee the weight of the spanner to be at most $\frac{k+1}{k-1}$ times the weight of a minimum spanning tree. The locality distance of the algorithm is in $O(k)$.

We consider a unit disk graph $G = (V, E)$. We compute a set $E' \subseteq E$ such that $G' = (V, E')$ forms a spanner for G. We also compute a coloring *color* : $V \rightarrow \{1, 2, 3, 4\}$ for G'. The algorithm has three steps:

1. A planar spanner G_k of G is created using the algorithm presented in [5,14].
2. The plane is divided into rectangles. A bipartite spanner for the vertices in each rectangle is computed and the vertices are colored with colors 1 and 2.
3. Collisions (two adjacent vertices with the same color) between vertices in different rectangles are being resolved by using two more colors.

4.1 The Algorithm

We consider a unit disk graph $G = (V, E)$. First we present the tiling of the plane which we are going to use. Then we present the three steps of the algorithm as outlined above. We will compute a set $E' \subseteq E$ such that $G' = (V, E')$ forms a spanner for G with certain properties. We will also compute a coloring *color* for G' with $|color| \leq 4$.

Tiling of the Plane. The plane is divided into tiles of rectangles. Figure 3 (left) shows one tile. A class number between 1 and 3 is assigned to each rectangle as shown in Figure 3 (left). The whole plane is tiled with such tiles, starting at an arbitrary position. Figure 3 (right) shows a part of this. The width of each rectangle is $6 + \epsilon$, the height of each rectangle is $3 + \epsilon$ for any fixed ϵ with $0 < \epsilon \leq \frac{1}{128}$. Each vertex is assigned to the rectangle which contains it. Ambiguities caused by vertices on the edge of rectangles are being resolved by assigning them to the rectangle with the smallest class number which contains them (any other resolving method works as well). We observe that two rectangles of the same class have a Euclidean distance of strictly more than three. So we conclude with the following propositions:

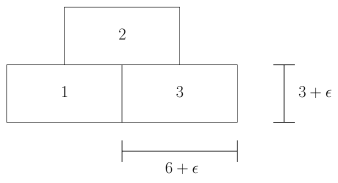


Fig. 3. One tile for the tiling of the plane and an extract of the whole tiling of the plane

Proposition 1. *Two vertices in different rectangles of the same class are not adjacent.*

Proposition 2. *For a rectangle R let R^+ denote the area of R plus a surrounding belt of width one. Let R_1 and R_2 be two rectangles of the same class. Then there is no edge in G which connects two vertices in $G[R_1^+]$ and $G[R_2^+]$.*

Each vertex can compute the rectangle that it belongs to from its coordinates in the plane.

Step 1: Computing the Spanner G_k. In this step fix an integer $k \geq 2$ and compute a spanner G_k for G (the value of k which we choose will determine the weight of our spanner in comparison with a minimum spanning tree for G). This routine is taken from [5,14]. As the spanner which we output later will be a subgraph of G_k, our spanner will inherit some properties from G_k. First we need to define an ordering on the edges of G.

Definition 1. (Compatible Linear Order). *Each edge (u, v) is assigned a 5-tuple $(|u, v|, x_1, y_1, x_2, y_2)$, where $|u, v|$ is the Euclidean length of the edge, x_1, y_1 and x_2, y_2 are the coordinates of the endnodes of the edge with either $x_1 > x_2$ or both $x_1 = x_2$ and $y_1 > y_2$. Clearly this gives a unique 5-tuple to any edge, and 5-tuples assigned to any two edges are distinct. The linear order $\prec$ is defined using the lexicographic ordering of the assigned 5-tuples.*

A graph may have several minimum spanning trees (MST) when the Euclidean length of the edges is the cost function. However, if we break the ties when an edge is chosen in the MST-algorithm (e.g. in Kruskal's algorithm) by the linear order $\prec$, then the graph has a unique MST (which can be computed for example by Kruskal's algorithm).

In step 1 of our algorithm we do the following: First we fix an integer $k \geq 2$. Then we compute the spanner G_k as explained in the description of the algorithm. It was originally presented in [5,14].

Step 1 of Algorithm 2: Computing a planar spanner with certain properties

```
1 // Algorithm is executed independently by each node;
2 // the parameter $k$ is fixed
3 Learn your distance $k$ neighborhood $N^k(v)$;
4 Constuct locally the unique MST $T^k(v)$ of $N^k(v)$;
5 Broadcast in $N^k(v)$ the edges of $N^1(v)$ which have been retained in $T^k(v)$ (i.e.
  $N^1(v) \cap T^k(v)$);
6 The output spanner $G_k$ is defined as follows: an edge is selected into $G_k$ if and
  only if it was retained by both of its incident nodes;
```

Step 2: Bipartite Spanner for each Rectangle. We compute a set E' and a map $color : V \rightarrow \mathbb{N}$. Note that after this step the map $color$ will not necessarily be a valid coloring for $G' = (V, E')$.

From now on, only edges that are part of G_k are considered. All other edges are ignored. Let R be a rectangle. Let $G[R]$ be the restriction of G to the vertices in R. For each connected component C in $G[R]$ we do the following: Compute a spanning tree T_C and a two-coloring $color_C$ for T_C which uses only colors 1 and 2. Assign all edges in T_C to E'. Define $color(v) := color_C(v)$ for all vertices v in C. Do this for all rectangles R which contain vertices of G. Finally we assign all edges to E' which connect vertices in different rectangles.

Step 3: Resolving Collisions. By a collision we denote an edge whose adjacent vertices have the same color. From the size of the rectangles in the tiling of the plane we conclude that such an edge must connect two vertices which are in adjacent rectangles (as the length of an edge is at most one). We first resolve all collisions where vertices in rectangles of class 1 are involved (step 3a). For that we need one additional color. Then we resolve collisions between vertices in rectangles of class 2 and 3 (step 3b). We use a fourth color for this.

We start with step 3a. Consider a rectangle R of class 1. Denote by V'' all vertices which are adjacent to at least one vertex in R. Denote by V' vertices in R which are adjacent to vertices in V''. Denote by $G_{coll}[R]$ the subgraph induced by $V' \cup V''$. For each connected component $C_{coll}[R]$ in $G_{coll}[R]$ we compute a spanning tree $T_{coll}[R]$. We compute a two-coloring $color_T$ for T_{coll}. Assume $color_T$ uses the colors $T1$ and $T2$. For all vertices v with $color_T(v) = T1$ we define $color(v) := 3$. Then we remove all edges $E(C_{coll}[R]) \setminus E(T_{coll}[R])$ from E'. Do this for all rectangles of class 1.

Resolving collisions between vertices in class 2 and 3 rectangles in step 3b works similarly: Consider a rectangle R of class 2. Denote by V'' all vertices in rectangles of class 3 which are adjacent to at least one vertex in R. Denote by V' vertices in R which are adjacent to vertices in V''. Denote by $G_{coll}[R]$ the subgraph induced by $V' \cup V''$. For each connected component $C_{coll}[R]$ in $G_{coll}[R]$ we compute a spanning tree $T_{coll}[R]$. We compute a two-coloring $color_T$ for $T_{coll}[R]$. Assume $color_T$ uses the colors $T1$ and $T2$. For all vertices v with $color_T(v) = T1$ we define $color(v) := 4$. Then we remove all edges $E(C_{coll}[R]) \setminus E(T_{coll}[R])$ from E'. Do this for all rectangles of class 2. This ensures the connectivity of the spanner while only using four colors to color it. We summarize the whole algorithm in Algorithm 2.

Algorithm 2. Local Algorithm for computing a spanner and a 4-coloring for the spanner

```
1 Fix an integer $k \geq 2$;
2 Compute the spanner $G_k$;
3 For all vertices $v$ compute $color(v)$ and compute the spanner $E'$ according to step
  2;
4 For all vertices $v$ check whether $color(v)$ is changed in step 3 and what edges of
  $E'$ remain after step 3;
```

We prove the correctness of Algorithm 2 and the properties of the computed spanner G' and the coloring *color* in Theorem 2.

Theorem 2. *Let $k \geq 2$ be the integer which was fixed at the beginning of Algorithm 2. For the computed spanner $G' = (V, E')$ and the computed coloring color it holds that*

1. *color is a valid coloring for G' which uses at most 4 colors*
2. *G' is connected,*
3. *G' is planar, $\triangle(G') \leq 5$, and no angle between two edges in G' is smaller than $\pi/3$,*
4. *for a minimum spanning tree T for G it holds that $cost(G') \leq \frac{k+1}{k-1} \cdot cost(T)$ and*
5. *the locality distance of Algorithm 2 is bounded by $136 + k$, i.e. Algorithm 2 is local.*

Proof. Due to space constraints we refer for the proof to our technical report[23].

4.2 Local Construction of 5 Colorable Spanners

Using a similar technique as described above we can find a local algorithm with a locality of $34+k$ which constructs a 5-colorable spanner of a unit disk graph and a 5-coloring for it. The spanner has the same additional properties (planarity, bounded degree, angles greater than $\pi/3$ and weight at most $\frac{k+1}{k-1}$ times the weight of a minimum spanning tree) as the spanners constructed by Algorithm 2. (We omit details here for lack of space.)

5 Impossibility Results

In this section we investigate the limits of local algorithms in terms of possible approximation ratios for vertex coloring and the numbers of colors of needed for a computed spanner. Due to the lack of space we give only the statements of the theorems and refer to our technical report [23] for the full proofs.

Theorem 3. *There is no local algorithm for computing connected bipartite spanners of unit disk graph.*

Theorem 4. *Let $\mathcal{A}$ be a local algorithm for vertex coloring. The approximation ratio of $\mathcal{A}$ is at least* $3/2$.

Theorem 5. *There is no local algorithm for coloring 3-colorable unit disk graphs using at most 3 colors. Moreover, there is no local algorithm for coloring 3-colorable planar spanners of unit disk graphs with at most 3 colors.*

References

1. Appel, K., Haken, W.: A proof of the four color theorem. Discrete Math. 16, 179–180 (1976)
2. Arya, S., Das, G., Mount, D.M., Salowe, J.S., Smid, M.: Euclidean spanners: short, thin, and lanky. In: ACM (ed.) Proceedings of the twenty-seventh annual ACM Symposium on Theory of Computing, Las Vegas, Nevada, May 29–June 1, pp. 489–498. ACM Press, New York (1995)
3. Bose, P., Gudmundsson, J., Smid, M.H.M.: Constructing plane spanners of bounded degree and low weight. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 234–246. Springer, Heidelberg (2002)
4. Bose, P., Morin, P., Stojmenovic, I., Urrutia, J.: Routing with guaranteed delivery in ad hoc wireless networks. Wireless Networks 7(6), 609–616 (2001)
5. Chávez, E., Dobrev, S., Kranakis, E., Opatrny, J., Stacho, L., Urrutia, J.: Local construction of planar spanners in unit disk graphs with irregular transmission ranges. In: Correa, J.R., Hevia, A., Kiwi, M.A. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 286–297. Springer, Heidelberg (2006)
6. Clark, B.N., Colbourn, C.J., Johnson, D.S.: Unit disk graphs. Discrete Math. 86(1-3), 165–177 (1990)
7. Czyzowicz, J., Dobrev, S., González-Aguilar, H., Kralovic, R., Kranakis, E., Opatrny, J., Stacho, L., Urrutia, J.: Local 7-coloring for planar subgraphs of unit disk graphs. In: Agrawal, M., Du, D.-Z., Duan, Z., Li, A. (eds.) TAMC 2008. LNCS, vol. 4978, pp. 170–181. Springer, Heidelberg (2008)
8. Eppstein, D.: Spanning trees and spanners. In: Sack, J.-R., Urrutia, J. (eds.) Handbook of Computational Geometry. Elsevier, Amsterdam (2000)
9. Feige, U., Kilian, J.: Zero knowledge and the chromatic number. In: Homer, S., Cai, J.-Y. (eds.) Proceedings of the 11th Annual IEEE Conference on Computational Complexity, CCC 1996, Philadelphia, Pennsylvania, Washington, D.C, May 24-27, 1996, pp. 278–287. IEEE Computer Society, Los Alamitos (1996)
10. Gabriel, K.R., Sokal, R.R.: A new statistical approach to geographic variation analysis. Systematic Zoology 18, 259–278 (1969)

11. Gräf, A., Stumpf, M., Weißenfels, G.: On coloring unit disk graphs. Algorithmica 20(3), 277–293 (1998)
12. Li, X.-Y., Calinescu, G., Wan, P.-J.: Distributed construction of planar spanner and routing for ad hoc wireless networks. In: Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Society (INFOCOM 2002), Piscataway, NJ, USA, June 23–27, 2002. Proceedings IEEE INFOCOM 2002, vol. 3, pp. 1268–1277. IEEE Computer Society, Los Alamitos (2002)
13. Li, X.-Y., Wang, Y.: Efficient construction of low weight bounded degree planar spanner. In: Warnow, T., Zhu, B. (eds.) COCOON 2003. LNCS, vol. 2697, pp. 374–384. Springer, Heidelberg (2003)
14. Li, X.-Y., Wang, Y., Song, W.-Z.: Applications of k-Local MST for Topology Control and Broadcasting in Wireless Ad Hoc Networks. IEEE Transactions on Parallel and Distributed Systems 15(12), 1057–1069 (2004)
15. Linial, N.: Locality in distributed graph algorithms. SIAM J. Comput. 21(1), 193–201 (1992)
16. Lund, C., Yannakakis, M.: On the hardness of approximating minimization problems. J. ACM 41(5), 960–981 (1994)
17. Marathe, M.V., Breu, H., Hunt III, H.B., Ravi, S.S., Rosenkrantz, D.J.: Simple heuristics for unit disk graphs. Networks 25(1), 59–68 (1995)
18. Möhring, R.H.: Algorithmic aspects of comparability graphs and interval graphs. In: Graphs and order, Banff, Alta. NATO Adv. Sci. Inst. Ser. C Math. Phys. Sci, vol. 147, pp. 41–101. Reidel, Dordrecht (1984)
19. Naor, M., Stockmeyer, L.: What can be computed locally? In: STOC 1993: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing, pp. 184–193. ACM Press, New York (1993)
20. Narasimhan, G., Smid, M.: Approximating the stretch factor of Euclidean graphs. SIAM Journal on Computing 30(3), 978–989 (2000)
21. Peeters, R.: On coloring j-unit sphere graphs. Technical report, Department of Economics, Tilburg University (1991)
22. Wang, Y., Li, X.-Y.: Localized construction of bounded degree and planar spanner for wireless ad hoc networks. In: DIALM-POMC, pp. 59–68. ACM, New York (2003)
23. Wiese, A., Kranakis, E.: Local construction and coloring of spanners of location aware unit disk graphs. Technical Report TR-07-18, Carleton University, School of Computer Science, Ottawa (December 2007)

Author Index